GUÍA AL CUERPO DE CONOCIMIENTO DE LA INGENIERÍA DEL SOFTWARE

VERSIÓN 2004

SWEBOK

UN PROYECTO DEL COMITÉ DE LA PRÁCTICA PROFESIONAL DEL IEEE COMPUTER SOCIETY

BORRADOR - ESPAÑOL

GUÍA AL CUERPO DE CONOCIMIENTO DE LA INGENIERÍA DEL SOFTWARE

VERSIÓN 2004

SWEBOK

Directores ejecutivos
Alain Abran, École de Technologie Superieure
James W. Moore, The Mitre Corp.

Directores
Pierre Bourque, École De Technologie Superieure
Robert Dupuis, Universite Du Quebec A Montreal

Jefe de proyecto
Leonard L. Tripp, Chair, Professional Practices Committee,
IEEE Computer Society (2001-2003)

IEEE COMPUTER SOCIETY
http://computer.org

Los Alamitos, California

Washington • Brussels • Tokyo

Número de Pedido C2330 de la Sociedad de Computadores IEEE
ISBN 0-7695-2330-7
Biblioteca del Congreso Número 2005921729

Copias adicionales deben ser pedidas desde:

Sociedad de Computadores IEEE
Centro de Servicio al Consumidor
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: +1-714-821-8380
Fax:+1-714-821-4641
E-mail: cs.books@computer.org

Centro de Servicio IEEE
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: +1-732-981-0060
Fax: +1-732-981-9667
http://shop.ieee.org/store/customer-service@ieee.org

Sociedad de Computadores IEEE
Oficina de Asia/Pacífico
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minatu-ku, Tokio 107-0062
Japón
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553
Tokio.ofc@computer.org

Publicadora: Ángela Burgess
Editora del Grupo de Gestión, Prensa CS: Deborah Plummer
Publicidad/Promociones: Tom Fink
Production Editor: Bob Werner
Impreso en los Estados Unidos de América

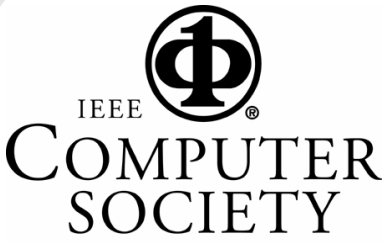


TABLA DE CONTENIDOS

PRÓLOGO

En esta guía, el IEEE Computer Society establece por primera vez una base para el cuerpo de conocimiento del campo de la ingeniería del software, y el trabajo, y el trabajo cubre parcialmente la responsabilidad de la *Sociedad* de promover el avance de la teoría y práctica de este campo. Llevando a cabo esta tarea, la Sociedad ha sido guiada por la experiencia de otras disciplinas más maduras sin ligarse a sus problemas o soluciones.

Nótese que la Guía no pretende definir el cuerpo de conocimiento, sino servir como un compendio y guía al cuerpo de conocimiento que se ha desarrollado y evolucionado durante 4 décadas. Es más, este cuerpo de conocimiento no es estático. La Guía debe necesariamente desarrollarse y evolucionar según madura la ingeniería del software. Sin embargo, constituye un valioso elemento a la infraestructura de la ingeniería del software.

En 1958, el renombrado matemático estadístico John Tukey, acuño el término *software*. El término Ingeniería del Software se utilizó por primera vez en el título de una conferencia de la OTAN celebrada en Alemania en 1968. La IEEE Computer Society publicó las primeras Transacciones en Ingeniería del Software (*Transactions on Software Engineering*) en 1972. El comité creado por la IEEE Computer Society para el desarrollo de estándares de ingeniería del software se fundó en 1976.

La primera vista global de la ingeniería del software emerge del trabajo del equipo liderado por Fletcher Buckley para desarrollar el estándar IEEE Std 730 para la calidad del software finalizado en 1979. El objetivo de dicho estándar fue el de proporcionar un mínimo de requerimientos aceptables y uniformes para el aseguramiento de la calidad. Este estándar influyó en el desarrollo de otros estándares posteriores relacionados con la gestión de la configuración, pruebas de software, requerimientos del software, diseño del software y verificación y validación del software.

Durante los años 1981-1985, la IEEE Computer Society organizó una serie de talleres de trabajo sobre la aplicación de los estándares de ingeniería del software en donde los profesionales que participaron compartieron sus experiencias con los estándares existentes y planificaron futuros estándares incluyendo uno sobre medición y métricas para procesos y productos de la ingeniería del software. El resultado fue el IEEE Std 1002, Taxonomía de estándares de ingeniería del software (1986), el cual proporciono una visión global de la ingeniería del software. El estándar describe la forma y contenido de una taxonomía de estándares de ingeniería del software. Explica los diferentes tipos de estándares de ingeniería del software, sus relaciones externas y funcionales, y el rol de las distintas funciones en el ciclo de vida del software.

En 1990 comenzó una planificación para un estándar internacional con una visión general. La planificación se centró en reconciliar las vistas de los procesos del software del estándar IEEE Std 1074 y el estándar 2167A del departamento de defensa (DoD, en sus siglas en inglés) de los EE.UU. La revisión del estándar se publicó como DoD Std 498. El estándar fue completado en 1995 como ISO/IEC 12207, Estándar para los procesos del ciclo de vida del software. Este estándar proporciono un importante punto de arranque para el cuerpo de conocimiento contenido en este libro.

La aprobación de la moción por parte de la junta de gobierno del IEEE Computer Society supuso el arranque por parte de Fletcher Buckley para la escritura de este libro. El consejo de la ACM (Association for Computing Machinery) aprobó la moción en agosto de 1993. Las dos mociones llevaron a la creación de un comité conjunto dirigido por Mario Barbacci y Stuart Zweben. La misión del comité conjunto fue: “establecer un conjunto apropiado de criterios y normas para la práctica profesional de la ingeniería del software sobre las que puedan basarse decisiones industriales, certificaciones profesionales y planes de estudios. El comité directivo organizó el trabajo en las siguientes áreas:

1. Definir el cuerpo de conocimiento y prácticas recomendadas.
2. Definir estándares éticos y profesionales
3. Definir planes de estudios para cursos de grado, postgrado y formación continua.

Este libro proporciona el primer punto, el cuerpo de conocimiento necesario y prácticas recomendadas.

El código ético y conducta profesional para la Ingeniería del software fue terminado en 1998 y aprobado por la juntas de gobierno de la ACM e IEEE Computer Society. Ha sido adoptado por numerosas empresas y organizaciones y se incluye en libros de texto recientes.

El plan de estudios para estudios de grado fue completado en 2004.

Aunque no siempre definidas de manera precisa, cada profesión se basa en un cuerpo de conocimiento y prácticas recomendadas. En muchos casos, están formalmente documentadas, generalmente en un formato que permite que sean usadas para acreditaciones de planes académicos, desarrollo de cursos y programas educativos, certificaciones profesionales o licencias profesionales. Generalmente, una sociedad profesional o cuerpo relacionado mantiene la custodia de la definición formal. En los casos en los que no existe esa formalidad, el cuerpo de conocimiento y prácticas recomendadas son "generalmente reconocidas" por profesionales y pueden ser codificadas en una variedad de formas por los distintos usuarios.

Se espera que los lectores encuentren este libro útil como guía para el conocimiento y recursos necesarios en el desarrollo de su profesión como profesionales de la ingeniería del software.

Este libro está dedicado a Fletcher Buckley en reconocimiento a su empeño en el promover la ingeniería del software como una disciplina profesional y su excelente profesionalidad como ingeniero de software para radares.

Leonard L. Tripp, IEEE Fellow 2003
Chair, Professional Practices Committee, IEEE Computer Society (2001-2003)
Chair, Joint IEEE Computer Society and ACM Steering Committee
for the Establishment of Software Engineering as a Profession (1998-1999)
Chair, Software Engineering Standards Committee, IEEE Computer Society (1992-1998)

EDITORES ASOCIADOS

Las siguientes personas trabajaron como directores asociados bien para la versión de prueba 2001 o bien la versión 2004.

Requerimientos del Software

Peter Sawyer and Gerald Kotonya, Computing Department, Lancaster University, Reino Unido, {p.sawy er, g.kotonya}@lancaster.ac.uk

Diseño del Software

Guy Tremblay, Département d'informatique, UQAM, Canadá, tremblay.guy@uqam.ca

Construcción del software

Steve McConnell, Construx Software, EEUU, Steve.McConnell@construx.com
Terry Bollinger, the MITRE Corporation, EEUU, terry@mitre.org
Philippe Gabrini, Département d'informatique, UQAM, Canadá, gabrini.philippe@uqam.ca
Louis Martin, Département d'informatique, UQAM, Canadá, martin.louis@uqam.ca

Pruebas del software

Antonia Bertolino y Eda Marchetti, ISTI-CNR, Italia, {antonia.bertolino}{eda.marchetti}@isti.cnr.it

Mantenimiento del software

Thomas M. Pigoski, Techsoft Inc., EEUU, tmpigoski@techsoft.com
Alain April, École de technologie supérieure, Canadá, aapril@ele.etsmtl.ca

Gestión de la configuración del software

John A. Scott, Lawrence Livermore National Laboratory, EEUU, scott7@llnl.gov
David Nisse, EEUU, nissed@worldnet.att.net

Gestión en la ingeniería del software

Dennis Frailey, Raytheon Company, EEUU, DJFrailey@Raytheon.com
Stephen G. MacDonell, Auckland University of technology, Nueva Zelanda, smacdone@aut.ac.nz
Andrew R. Gray, University of Otago, Nueva Zelanda

Procesos de ingeneniería del software

Khaled El Emam, Canadian National Research Council, Canadá, khaled.el-emam@nrc-cnrc.gc.ca

Herramientas y métodos en la ingeniería del software

David Carrington, School of Information Technology and Electrical Engineering, The University of Queensland, Australia, davec@itee.uq.edu.au

Calidad del software

Alain April, École de technologie supérieure, Canadá, aapril@ele.etsmtl.ca
Dolores Wallace, retired from the National Institute of Standards and Technology, EEUU, Dolores.Wallace@nist.gov
Larry Reeker, NIST, EEUU, Larry.Reeker@nist.gov

Coordinador de referencias

Marc Bouisset, Département d'informatique, UQAM, Bouisset.Marc@uqam.ca

COMITÉ EJECUTIVO PROFESIONAL

A fecha de publicación, las personas nombradas a continuación formaron el comité ejecutivo profesional:

Mario R. Barbacci, Software Engineering Institute, representando al IEEE Computer Society

Carl Chang, representando a Computing Curricula 2001

François Coallier, École de technologie supérieure, como director de ISO/IEC JTC 1/SC7

Charles Howell, The MITRE Corporation

Anatol Kark, National Research Council of Canadá

Philippe Kruchten, University of British Columbia, como representante de Rational Software

Laure Le Bars, SAP Labs (Canadá)

Steve McConnell, Construx Software

Dan Nash, Raytheon Company

Fred Otto, Canadian Council of Professional Engineers (CCPE)

Richard Metz, The Boeing Company

Larry Reeker, National Institute of Standards and Technology, Department of Commerce, EEUU

Las siguientes personas trabajaron con este comité en el control de cambios de la edición del 2004:

Donald Bagert, Rose-Hulman Institute of Technology, representing the IEEE Computer Society Professional Practices Committee

Ann Sobel, Miami University, representado a la junta ejecutiva del Computing Curricula Software Engineering

PANEL DE EXPERTOS

Las siguientes personas trabajaron como panel de expertos en la preparación de la versión de prueba (Trial) de la Guía:

Steve McConnell, Construx Software

Roger Pressman, R.S. Pressman and Associates

Ian Sommerville, Lancaster University, Reino Unido

REVISORES

Las siguientes personas trabajaron como directores asociados en la versión de prueba (Trial) 2001 y/o la versión 2004.

Abbas, Rasha, Australia
Abran, Alain, Canadá
Accioly, Carlos, Brasil
Ackerman, Frank, EEUU
Akiyama, Yoshihiro, Japón
Al-Abdullah, Mohammad, EEUU
Alarcon, Miren Idoia, España
Alawy, Ahmed, EEUU
Alleman, Glen, EEUU
Allen, Bob, Canadá
Allen, David, EEUU
Amorosa, Franciasco, Italia
Amyot, Daniel, Canadá
Andrade, Daniel, Brasil
April, Alain, Canadá
Arroyo-Figueror, Javier, EEUU
Ashford, Sonny, EEUU
Atsushi, Sawada, Japón
Backitis Jr., Frank, EEUU
Bagert, Donald, EEUU
Baker, Jr., David, EEUU
Baker, Theodore, EEUU
Baldwin, Mark, EEUU
Bales, David, Reino Unido
Bamberger, Judy, EEUU
Banerjee, Bakul, EEUU
Barber, Scott, EEUU
Barker, Harry, Reino Unido
Barnes, Julie, EEUU
Barney, David, Australia
Barros, Rafael, Colombia
Bastarache, Louis, Canadá
Bayer, Steven, EEUU
Beaulac, Adeline, Canadá
Beck, William, EEUU
Beckman, Kathleen, EEUU
Below, Doreen, EEUU
Benediktsson, Oddur, Islandia
Ben-Menachem, Mordechai, Israel
Bergeron, Alain, Canadá
Berler, Alexander, Grecia
Bernet, Martin, EEUU
Bernstein, Larry, EEUU
Bertram, Martin, Alemania
Bialik, Tracy, EEUU
Bielikova, Maria, Eslovaquia
Bierwolf, Robert, The Países Bajos
Bisbal, Jesus, Irlanda
Boivin, Michel, Canadá
Bolton, Michael, Canadá
Bomitali, Evelino, Italia
Bonderer, Reto, Suiza
Bonk, Francis, EEUU
Booch, Grady, EEUU
Booker, Glenn, EEUU
Börstler, Jürgen, Suecia
Borzovs, Juris, Latvia
Botting, Richard, EEUU
Bourque, Pierre, Canadá
Bowen, Thomas, EEUU
Boyd, Milt, EEUU
Boyer, Ken, EEUU
Brashear, Phil, EEUU
Briggs, Steve, EEUU
Bright, Daniela, EEUU
Brosseau, Jim, Canadá
Brotbeck, George, EEUU
Brown, Normand, Canadá
Bruhn, Anna, EEUU
Brune, Kevin, EEUU
Bryant, Jeanne, EEUU
Buglione, Luigi, Italia
Bullock, James, EEUU
Burns, Robert, EEUU
Burnstein, Ilene, EEUU
Byrne, Edward, EEUU
Calizaya, Percy, Perú
Carreon, Juan, EEUU
Carroll, Sue, EEUU
Carruthers, Kate, Australia
Caruso, Richard, EEUU
Carvalho, Paul, Canadá
Case, Pam, EEUU
Cavanaugh, John, EEUU
Celia, John A., EEUU
Chalupa Sampaio, Alberto Antonio, Portugal
Chan, F.T., Hong Kong
Chan, Keith, Hong Kong
Chandra, A.K., India
Chang, Wen-Kui, Taiwan
Chapin, Ned, EEUU
Charette, Robert, EEUU
Chevrier, Marielle, Canadá
Chi, Donald, EEUU
Chiew, Vincent, Canadá
Chilenski, John, EEUU
Chow, Keith, Italia
Ciciliani, Ricardo, Argentina
Clark, Glenda, EEUU
Cleavenger, Darrell, EEUU
Cloos, Romain, Luxembourg
Coallier, François, Canadá
Coblentz, Brenda, EEUU
Cohen, Phil, Australia
Collard, Ross, Nueva Zelanda
Collignon, Stephane, Australia
Connors, Kathy Jo, EEUU
Cooper, Daniel, EEUU
Councill, Bill, EEUU
Cox, Margery, EEUU
Cunin, Pierre-Yves, Francia
DaLuz, Joseph, EEUU
Dampier, David, EEUU
Daneva, Maya, Canadá
Daneva, Maya, Canadá
Daughtry, Taz, EEUU
Davis, Ruth, EEUU
De Cesare, Sergio, Reino Unido
Dekleva, Sasa, EEUU
Del Castillo, Federico, Perú
Del Dago, Gustavo, Argentina
DeWeese, Perry, EEUU
Di Nunno, Donn, EEUU
Diaz-Herrera, Jorge, EEUU
Dieste, Oscar, España
Dion, Francis, Canadá
Dixon, Wes, EEUU
Dolado, Javier, España
Donaldson, John, Reino Unido
Dorantes, Marco, Mexico
Dorofee, Audrey, EEUU
Douglass, Keith, Canadá
Du, Weichang, Canadá
Duben, Anthony, EEUU
Dudash, Edward, EEUU
Duncan, Scott, EEUU
Duong, Vinh, Canadá
Durham, George, EEUU

Dutil, Daniel, Canadá
Dutton, Jeffrey, EEUU
Ebert, Christof, Francia
Edge, Gary, EEUU
Edwards, Helen Maria, Reino Unido
El-Kadi, Amr, Ejipto
Endres, David, EEUU
Engelmann, Franz, Suiza
Escue, Marilyn, EEUU
Espinoza, Marco, Perú
Fay, Istvan, Hungría
Fayad, Mohamed, EEUU
Fendrich, John, EEUU
Ferguson, Robert, EEUU
Fernandez, Eduardo, EEUU
Fernandez-Sanchez, Jose Luis, España
Filgueiras, Lucia, Brasil
Finkelstein, Anthony, Reino Unido
Flinchbaugh, Scott, EEUU
Forrey, Arden, EEUU
Fortenberry, Kirby, EEUU
Foster, Henrietta, EEUU
Fowler, Martin, EEUU
Fowler, John Jr., EEUU
Fox, Christopher, EEUU
Frankl, Phyllis, EEUU
Freibergs, Imants, Latvia
Frezza, Stephen, EEUU
Fruehauf, Karol, Suiza
Fuggetta, Alphonso, Italia
Fujii, Roger, EEUU
FUSCHI, David Luigi, Italia
Fuschi, David Luigi, Italia
Gabrini, Philippe, Canadá
Gagnon, Eric, Canadá
Ganor, Eitan, Israel
Garbajosa, Juan, España
Garceau, Benoît, Canadá
Garcia-Palencia, Omar, Colombia
Garner, Barry, EEUU
Gelperin, David, EEUU
Gersting, Judith, Hawaii
Giesler, Gregg, EEUU
Gil, Indalecio, España
Gilchrist, Thomas, EEUU
Giurescu, Nicolae, Canadá
Glass, Robert, EEUU
Glynn, Garth, Reino Unido
Goers, Ron, EEUU
Gogates, Gregory, EEUU
Goldsmith, Robin, EEUU
Goodbrand, Alan, Canadá
Gorski, Janusz, Polonia
Graybill, Mark, EEUU
Gresse von Wangenheim, Christiane, Brasil
Grigonis, George, EEUU
Gupta, Arun, EEUU
Gustafson, David, EEUU
Gutcher, Frank, EEUU
Haas, Bob, EEUU
Hagar, Jon, EEUU
Hagstrom, Erick, EEUU
Hailey, Victoria, Canadá
Hall, Duncan, Nueva Zelanda
Haller, John, EEUU
Halstead-Nussloch, Richard, EEUU
Hamm, Linda, EEUU
Hankewitz, Lutz, Alemania
Harker, Rob, EEUU
Hart, Hal, EEUU
Hart, Ronald, EEUU
Hartner, Clinton, EEUU
Hayeck, Elie, EEUU
He, Zhonglin, Reino Unido
Hedger, Dick, EEUU
Hefner, Rick, EEUU
Heinrich, Mark, EEUU
Heinze, Sherry, Canadá
Hensel, Alan, EEUU
Herrmann, Debra, EEUU
Hesse, Wolfgang, Alemania
Hilburn, Thomas, EEUU
Hill, Michael, EEUU
Ho, Vinh, Canadá
Hodgen, Bruce, Australia
Hodges, Brett, Canadá
Hoffman, Douglas, Canadá
Hoffman, Michael, EEUU
Hoganson, Tammy, EEUU
Hollocker, Chuck, EEUU
Horch, John, EEUU
Howard, Adrian, Reino Unido
Huang, Hui Min, EEUU
Hung, Chih-Cheng, EEUU
Hung, Peter, EEUU
Hunt, Theresa, EEUU
Hunter, John, EEUU
Hvannberg, Ebba Thora, Islandia
Hybertson, Duane, EEUU
Ikiz, Seckin, Turkey
Iyengar, Dwaraka, EEUU
Jackelen, George, EEUU
Jaeger, Dawn, EEUU
Jahnke, Jens, Canadá
James, Jean, EEUU
Jino, Mario, Brasil
Johnson, Vandy, EEUU
Jones, Griffin, EEUU
Jones, James E., EEUU
Jones, Alan, Reino Unido
Jones, James, EEUU
Jones, Larry, Canadá
Jones, Paul, EEUU
Ju, Dehua, China
Juan-Martinez, Manuel-Fernando, España
Juhasz, Zoltan, Hungría
Juristo, Natalia, España
Kaiser, Michael, Suiza
Kambic, George, EEUU
Kamthan, Pankaj, Canadá
Kaner, Cem, EEUU
Kark, Anatol, Canadá
Kasser, Joe, EEUU
Kasser, Joseph, Australia
Katz, Alf, Australia
Kececi, Nihal, Canadá
Kell, Penelope, EEUU
Kelly, Diane, Canadá
Kelly, Frank, EEUU
Kenett, Ron, Israel
Kenney, Mary L., EEUU
Kerievsky, Joshua, EEUU
Kerr, John, EEUU
Kierzyk, Robert, EEUU
Kinsner, W., Canadá
Kirkpatrick, Harry, EEUU
Kittiel, Linda, EEUU
Klappholz, David, EEUU
Klein, Joshua, Israel
Knight, Claire, Reino Unido
Knoke, Peter, EEUU
Ko, Roy, Hong Kong
Kolewe, Ralph, Canadá
Komal, Surinder Singh, Canadá
Kovalovsky, Stefan, Austria
Krauth, Péter, Hungría
Krishnan, Nirmala, EEUU
Kromholz, Alfred, Canadá
Kruchten, Philippe, Canadá
Kuehner, Nathanael, Canadá
Kwok, Shui Hung, Canadá
Lacroix, Dominique, Canadá
LaMotte, Stephen W., EEUU
Land, Susan, EEUU
Lange, Douglas, EEUU
Laporte, Claude, Canadá
Lawlis, Patricia, EEUU
Le, Thach, EEUU
Leavitt, Randal, Canadá
LeBel, Réjean, Canadá
Leciston, David, EEUU
Lee, Chanyoung, EEUU
Lehman, Meir (Manny), Reino Unido

Leigh, William, EEUU
Lembo, Jim, EEUU
Lenss, John, EEUU
Leonard, Eugene, EEUU
Lethbridge, Timothy, Canadá
Leung, Hareton, Hong Kong
Lever, Ronald, Países Bajos
Levesque, Ghislain, Canadá
Ley, Earl, EEUU
Linders, Ben, Países Bajos
Linscomb, Dennis, EEUU
Little, Joyce Currie, EEUU
Logan, Jim, EEUU
Long, Carol, Reino Unido
Lounis, Hakim, Canadá
Low, Graham, Australia
Lutz, Michael, EEUU
Lynch, Gary, EEUU
Machado, Cristina, Brasil
MacKay, Stephen, Canadá
MacKenzie, Garth, EEUU
MacNeil, Paul, EEUU
Magel, Kenneth, EEUU
Mains, Harold, EEUU
Malak, Renee, EEUU
Maldonado, José Carlos, Brasil
Marcos, Esperanza, España
Marinescu, Radu, Rumanía
Marm, Waldo, Perú
Marusca, Ioan, Canadá
Matlen, Duane, EEUU
Matsumoto, Yoshihiro, Japón
McBride, Tom, Australia
McCarthy, Glenn, EEUU
McChesney, Ian, Reino Unido
McCormick, Thomas, Canadá
McCown, Christian, EEUU
McDonald, Jim, EEUU
McGrath Carroll, Sue, EEUU
McHutchison, Diane, EEUU
McKinnell, Brian, Canadá
McMichael, Robert, EEUU
McMillan, William, EEUU
McQuaid, Patricia, EEUU
Mead, Nancy, EEUU
Meeuse, Jaap, Países Bajos
Meier, Michael, EEUU
Meisenzahl, Christopher, EEUU
Melhart, Bonnie, EEUU
Mengel, Susan, EEUU
Meredith, Denis, EEUU
Meyerhoff, Dirk, Alemania
Mili, Hafedh, Canadá
Miller, Chris, Países Bajos
Miller, Keith, EEUU
Miller, Mark, EEUU
Miranda, Eduardo, Canadá
Mistrik, Ivan, Alemania
Mitasiunas, Antanas, Lituania
Modell, Howard, EEUU
Modell, Staiger,EEUU
Modesitt, Kenneth, EEUU
Moland, Kathryn, EEUU
Moldavsky, Symon, Ucrania
Montequín, Vicente R., España
Moreno, Ana Maria, España
Mosiuoa, Tseliso, Lesotho
Moudry, James, EEUU
Msheik, Hamdan, Canadá
Mularz, Diane, EEUU
Mullens, David, EEUU
Müllerburg, Monika, Alemania
Murali, Nagarajan, Australia
Murphy, Mike, EEUU
Napier, John, EEUU
Narasimhadevara, Sudha, Canadá
Narawane, Ranjana, India
Narayanan, Ramanathan, India
Navarro Ramirez, Daniel, México
Navas Plano, Francisco, España
Navrat, Pavol, Eslovaquia
Neumann, Dolly, EEUU
Nguyen-Kim, Hong, Canadá
Nikandros, George, Australia
Nishiyama, Tetsuto, Japón
Nunn, David, EEUU
O'Donoghue, David, Irlanda
Oliver, David John, Australia
Olson, Keith, EEUU
Oskarsson, Östen, Suecia
Ostrom, Donald, EEUU
Oudshoorn, Michael, Australia
Owen, Cherry, EEUU
Pai, Hsueh-Ieng, Canadá
Parrish, Lee, EEUU
Parsons, Samuel, EEUU
Patel, Dilip, Reino Unido
Paulk, Mark, EEUU
Paella, Jan, República Checa
Pavlov, Vladimir, Ucrania
Pawlyszyn, Blanche, EEUU
Pecceu, Didier, Francia
Perisic, Branko, Yugoslavia
Perry, Dale, EEUU
Peters, Dennis, Canadá
Petersen, Erik, Australia
Pfahl, Dietmar, Alemania
Pfeiffer, Martin, Alemania
Phillips, Dwayne, EEUU
Phipps, Robert, EEUU
Phister, Paul, EEUU
Phister, Jr., Paul, EEUU
Piattini, Mario, España
Piersall, Jeff, EEUU
Pillai, S.K., India
Pinder, Alan, Reino Unido
Pinheiro, Francisco A., Brasil
Plekhanova, Valentina, Reino Unido
Poon, Peter, EEUU
Poppendieck, Mary, EEUU
Powell, Mace, EEUU
Predenkoski, Mary, EEUU
Prescott, Allen, EEUU
Pressman, Roger, EEUU
Price, Art, EEUU
Price, Margaretha, EEUU
Pullum, Laura, EEUU
Purser, Keith, EEUU
Purssey, John, Australia
Pustaver, John, EEUU
Quinn, Anne, EEUU
Radnell, David, Australia
Rae, Andrew, Reino Unido
Rafea, Ahmed, Ejipto
Ramsden, Patrick, Australia
Rao, N.Vyaghrewara, India
Rawsthorne, Dan, EEUU
Reader, Katherine, EEUU
Reddy, Vijay,EEUU
Redwine, Samuel, EEUU
Reed, Karl, Australia
Reedy, Ann, EEUU
Reeker, Larry, EEUU
Rethard, Tom, EEUU
Reussner, Ralf, Alemania
Rios, Joaquin, España
Risbec, Philippe, Francia
Roach, Steve, EEUU
Robillard, Pierre, Canadá
Rocha, Zalkind, Brasil
Rodeiro Iglesias, Javier, España
Rodriguez-Dapena, Patricia, España
Rogoway, Paul, Israel
Rontondi, Guido, Italia
Roose, Philippe, Francia
Rosca, Daniela, EEUU
Rosenberg, Linda, EEUU
Rourke, Michael, Australia
Rout, Terry, Australia
Rufer, Russ, EEUU
Ruiz, Francisco, España
Ruocco, Anthony, EEUU
Rutherfoord, Rebecca, EEUU
Ryan, Michael, Irlanda
Salustri, Filippo, Canadá

Salustri, Filippo, Canadá
Salwin, Arthur, EEUU
Sanden, Bo, EEUU
Sandmayr, Helmut, Suiza
Santana Filho, Ozeas Vieira, Brasil
Sato, Tomonobu, Japón
satyadas, antony, EEUU
Satyadas, Antony, EEUU
Schaaf, Robert, EEUU
Scheper, Charlotte, EEUU
Schiffel, Jeffrey, EEUU
Schlicht, Bill, EEUU
Schrott, William, EEUU
Schwarm, Stephen, EEUU
Schweppe, Edmund, EEUU
Sebern, Mark, EEUU
Seffah, Ahmed, Canadá
Selby, Nancy, EEUU
Selph, William, EEUU
Sen, Dhruba, EEUU
Senechal, Raymond, EEUU
Sepulveda, Christian, EEUU
Setlur, Atul, EEUU
Sharp, David, EEUU
Shepard, Terry, Canadá
Shepherd, Alan, Alemania
Shillato, Rrobert W, EEUU
Shintani, Katsutoshi, Japón
Silva, Andres, España
Silva, Andres, España
Singer, Carl, EEUU
Sinnett, Paul, Reino Unido
Sintzoff, André, Francia
Sitte, Renate, Australia
Sky, Richard, EEUU
Smilie, Kevin, EEUU
Smith, David, EEUU
Sophatsathit, Peraphon, Tailandia
Sorensen, Reed, EEUU
Soundarajan, Neelam, EEUU
Sousa Santos, Frederico, Portugal
Spillers, Mark, EEUU
Spinellis, Diomidis, Grecia
Splaine, Steve, EEUU
Springer, Donald, EEUU
Staiger, John, EEUU
Starai, Thomas, EEUU
Steurs, Stefan, Belgium
St-Pierre, Denis, Canadá
Stroulia, Eleni, Canadá
Subramanian, K.S., India
Sundaram, Sai, Reino Unido
Swanek, James, EEUU
Swearingen, Sandra, EEUU
Szymkowiak, Paul, Canadá
Tamai, Tetsuo, Japón
Tasker, Dan, Nueva Zelanda
Taylor, Stanford, EEUU
Terekhov, Andrey A., Russian Federation
Terski, Matt, EEUU
Thayer, Richard, EEUU
Thomas, Michael, EEUU
Thompson, A. Allan, Australia
Thompson, John Barrie, Reino Unido
Titus, Jason, EEUU
Tockey, Steve, EEUU
Tovar, Edmundo, España
Towhidnejad, Massood, EEUU
Trellue, Patricia, EEUU
Trèves, Nicolas, Francia
Troy, Elliot, EEUU
Tsui, Frank, EEUU
Tsuneo, Furuyama, Japón
Tuohy, Kenney, EEUU
Tuohy, Marsha P., EEUU
Turczyn, Stephen, EEUU
Upchurch, Richard, EEUU
Urbanowicz, Theodore, EEUU
Van Duine, Dan, EEUU
Van Ekris, Jaap, Países Bajos
Van Oosterhout, Bram, Australia
Vander Plaats, Jim, EEUU
Vegas, Sira, España
Verner, June, EEUU
Villas-Boas, André, Brasil
Vollman, Thomas, EEUU
Walker, Richard, Australia
Walsh, Bucky, EEUU
Wang, Yingxu, Suecia
Wear, Larry, EEUU
Weigel, richard, EEUU
Weinstock, Charles, EEUU
Wenyin, Liu, China
Werner, Linda, EEUU
Wheeler, David, EEUU
White, Nathan, EEUU
White, Stephanie, EEUU
Whitmire, Scott, EEUU
Wijbrans, Klaas, The Países Bajos
Wijbrans-Roodbergen, Margot, Países Bajos
Wilkie, Frederick, Reino Unido
Wille, Cornelius, Alemania
Wilson, Charles, EEUU
Wilson, Leon, EEUU
Wilson, Russell, EEUU
Woechan, Kenneth, EEUU
Woit, Denise, Canadá
Yadin, Aharon, Israel
Yih, Swu, Taiwan
Young, Michal, EEUU
Yrivarren, Jorge, Perú
Znotka, Juergen, Alemania
Zuser, Wolfgang, Austria
Zvegintzov, Nicholas, EEUU
Zweben, Stu, EEUU

La siguiente moción fue unánimemente aprobada por el comité ejecutivo profesional el 6 de febrero de 2004.

El comité ejecutivo profesional afirma que el cuerpo de conocimiento de la ingeniería del software iniciado en 1998 se ha completado satisfactoriamente y endorsa la versión 2004 de la guía al SWEBOK y lo recomienda su aprobación a junta de gobierno de *IEEE Computer Society*.

La siguiente moción fue unánimemente aprobada por el junta de gobierno del IEEE Computer Society en febrero del 2004.

La junta de gobierno de IEEE Computer Society aprueba la edición del 2004 de la Guía al cuerpo de conocimiento de la ingeniería del software y autoriza al director del comité de prácticas profesionales a proceder con su impresión.

PREFACIO

La ingeniería del software es una disciplina emergente y hay tendencias que indican un incremento en su nivel de madurez:

- Varias universidades en el mundo ofrecen titulaciones en ingeniería del software. Ejemplos de tales titulaciones se incluyen: University of New South Wales (Australia), McMaster University (Canadá), Rochester Institute of Technology (EE.UU.), University of Sheffield (Reino Unido), etc.
- En los EE.UU., la comisión de acreditación de ingenierías de ABET (*Accreditation Board for Engineering and Technology*) es el responsable de la acreditación de planes de estudios de grado de ingeniería del software.
- La *Canadian Information Processing Society* ha publicado los criterios para acreditar programas universitarios de grado de ingeniería del software.
- El CMM (Capability Maturiry Model) y el CMMI (Capability Maturiry Model Integration) del SEI (Software Enginering Institute) se emplean para evaluar la capacidad/madurez de la ingeniería del software en las empresas. Los estándares de gestión de calidad ISO 9000 han sido aplicados a la ingeniería del software en el estándar ISO/IEC 90003.
- La Junta de Ingenieros Profesionales de Texas ha comenzado a emitir licencias a los profesionales de la ingeniería del software.
- La APEGBC (Association of Professional Engineers and Geoscientists of British Columbia) inscribe a ingenieros del software y la PEO (Professional Engineers of Ontario) ha anunciado los requerimientos para las licencias.
- La ACM (Association for Computing Machinery) y el IEEE Computer Society han desarrollado conjuntamente y adoptado el código ético y conducta profesional[1].
- El IEEE Computer Society ofrece el Certificado de desarrollador de software profesional. El ICCP (Institute for Certification of Computing Professionals) lleva tiempo ofreciendo una certificación a profesionales de la informática.

Todos estos esfuerzos están basados en la presunción de que existe un cuerpo de conocimiento que debería ser dominado por los profesionales del software. Dicho cuerpo de conocimiento existe en la literatura acumulada en los últimos 30 años. Este libro proporciona una guía al cuerpo de conocimiento.

[1] El código ético y conducta profesional está disponible en español en: http://www.sc.ehu.es/jiwdocoj/codeacm.htm
e inglés: http://www.computer.org/certification/ethics.htm

PROPÓSITO

El propósito de la guía al conocimiento de la ingeniería del software es proporcional una caracterización validada y consensuada de los límites de la disciplina de la ingeniería del software, y proporcionar un acceso a los temas del cuerpo de conocimiento apoyando la disciplina. El cuerpo de conocimiento está dividido en 10 áreas del conocimiento (AC), KA, Knowledge Areas) más un capítulo adicional proporcionando una perspectiva general de las AC. Las descripciones de las AC están diseñadas para discernir entre los varios conceptos importantes, permitiendo al lector encontrar rápidamente los temas de interés. Una vez encontrado el tema de interés, el lector es referido a artículos clave o capítulos de libro seleccionados por presentar el conocimiento de manera sucinta.

Con una ojeada a la guía, los lectores notarán que el contenido es sustancialmente diferente de la "ciencia de la informática". Al igual que el ingeniero electrónico se basa en física, el ingeniero del software se debería basar, entre otras cosas, en la informática. En estos dos casos, el énfasis es necesariamente diferente. Los científicos (físicos, informáticos) incrementan nuestro conocimiento de las leyes de la naturaleza, sin embargo, los ingenieros aplican esas leyes para construir artefactos útiles bajo ciertas condiciones. Por tanto, el énfasis de esta guía se centra en la construcción de artefactos de software útiles bajo ciertas restricciones.

Los lectores también notarán en muchos aspectos importantes de la tecnología de la información pueden constituir conocimientos importantes en la ingeniería del software aún no cubiertos en esta guía, por ejemplo, lenguajes de programación específicos, bases de datos relacionales y redes. Todo esto es consecuencia del punto de vista de la ingeniería de software tomado. En todas las áreas – no solamente en informática – los diseñadores de los planes de estudios en ingeniería se han dado cuenta que tecnologías específicas están siendo reemplazadas mucho más rápidamente que la vida laboral de los ingenieros que las utilizan. Por tanto, los ingenieros de tener un conocimiento en el que basar la selección de la tecnología apropiada en cierto momento y bajo unas circunstancias determinadas. Por ejemplo, un programa puede ser construido en Fortran utilizando descomposición funcional o en C++ utilizando técnicas orientadas a objeto. Aunque las técnicas para configuración de dichos sistemas serían bastante diferentes, los principios y objetivos en la gestión de configuración son los mismos. La guía, por tanto, no se centra en el constante cambio de tecnologías, sino en los principios generales relevantes descritos en las áreas de conocimiento.

Estas exclusiones demuestran que la guía es necesariamente incompleta. La guía cumbre conocimiento sobre la ingeniería del software que es condición necesaria pero no suficiente para los ingenieros del software. Los profesionales de la ingeniería del software necesitarán tener amplios conocimientos de, por ejemplo, informática, gestión de proyectos, y la ingeniería de sistemas que están fuera del cuerpo de conocimiento caracterizado en esta guía; sin embargo decir que todos esos conceptos deberían ser conocidos por los ingenieros del software no es lo mismo que decir que este conocimiento cae dentro de las fronteras de la ingeniería del software. No obstante, sí que se afirma que los ingenieros del software necesitan conocimientos de otras disciplinas. Ese es el punto de vista adoptado en esta guía. Por tanto, esta guía caracteriza el cuerpo de conocimiento que cae dentro del ámbito de la ingeniería de software y proporciona referencias a otros conocimientos relevantes en otras disciplinas. Un capítulo de la guía proporciona una visión global taxonómica de disciplinas relacionadas derivadas de importantes fuentes.

El énfasis en la práctica de la ingeniería lleva a la guía a tener una fuerte relación con literatura normativa. La mayoría de la literatura en la informática, tecnologías de la información e ingeniería de software proporciona información útil a los ingenieros del software, pero sólo una pequeña parte de ella es normativa. Una referencia normativa prescribe que debería hacer un ingeniero en una situación específica en vez de proporcionar información que podría ser útil. La literatura normativa está validada por consenso entre los profesionales, y se concentra en estándares y documentos relacionados. Desde la concepción del SWEBOK, éste fue concebido con una fuerte relación a la literatura normativa en la ingeniería del software. Las dos entidades más importantes en la creación de estándares de ingeniería de software IEEE Computer Society e ISO/IEC JCT1/SC7 están siendo representadas en el proyecto. En última instancia, esperamos que los estándares en la práctica de la ingeniería del software contengan los principios contenidos en esta guía.

A QUIÉN VA DIRIGIDO

La guía está orientada hacia una gran variedad de audiencias en todo el mundo. Intenta proporcionar a organismos públicos o privados con una visión estable de la ingeniería de software para definir requerimientos en educación, competencias para trabajos, desarrollo de políticas de evaluación, o la especificación de tareas de desarrollo de software. También se dirige a la práctica o gestión, ingenieros de software y responsables de políticas relacionadas con licencias y guías de profesionales. Además, se beneficiarán del SWEBOK las sociedades profesionales y académicos que definen las normas de certificación, políticas de acreditación para planes de estudio universitarios y guías para la práctica profesional. Finalmente, estudiantes de ingeniería de software y académicos relacionados con la definición de planes de estudio y contenidos de asignaturas.

EVOLUCIÓN DE LA GUÍA

Desde 1993 hasta el 2000, el IEEE Computer Society y la ACM cooperaron en promover la profesionalización de la ingeniería de software a través de un comité para la coordinación de la ingeniería del software (*Software Engineering Coordinating Committee* – SWECC). El código de ética y conducta profesional fue completado bajo la tutela de SWECC en1998 a través de esfuerzos voluntarios. El SWEBOK fue iniciado por el SWECC en 1998.

El ámbito del proyecto SWEBOK, la gran variedad de comunidades involucradas, y la necesidad de una amplia participación sugirieron la necesidad de una gestión a tiempo completo en lugar de esfuerzos voluntarios y altruistas. Con este fin, el *IEEE Computer Society* contrato con el laboratorio de investigación de ingeniería de software de la Universidad de Québec en Montreal (Université du Québec à Montréal -UQAM) para gestionar el proyecto. En los últimos años, UQAM se ha unido a la Escuela de Tecnología Superior de Montréal, Quebec (ÉTS – École technologie supérieure).

El proyecto se dividió en tres fases: *Hombre de Paja*, *Hombre de Piedra* y *Hombre de Hierro*. Un prototipo inicial, *Hombre de Paja*, demostró como el proyecto podría ser organizado. La publicación de la ampliamente circulada versión de prueba de la guía en el 2001 (Trial Version) marcó fin de la fase *Hombre de Piedra* e inicio un periodo de prueba en uso. La guía actual marca el fin de del periodo *Hombre de Hierro* proporcionando una guía que ha alcanzado un consenso mediante revisiones y pruebas.

El equipo que desarrolló el proyecto se basó en dos principios fundamentales para el desarrollo de la guía: transparencia y consenso. Por transparencia, se quiere decir que el proceso de desarrollo en sí mismo está documentado, publicado y publicitado de manera que las decisiones importantes y progreso son visibles a todas las partes constituyentes por consenso. Por consenso, queremos decir que el único método práctico para legitimar afirmaciones es a través de una amplia participación y acuerdos por todos los sectores relevantes en la comunidad.

Literalmente cientos de copartícipes, revisores y usuarios durante el periodo de prueba han contribuido en la producción del actual documento.

Como cualquier proyecto de software, el en SWEBOK contribuyeron muchos participantes – algunos de los cuales están formalmente representados. El proyecto ha sido financiado por un *comité ejecutivo profesional* compuesto por representantes de la empresa (Boeing, Construx Software, the MITRE Corporation, Rational Software, Raytheon Systems, and SAP Labs-Canadá), institutos de investigación (National Institute of Standards

and Technology, National Research Council of Canadá), CCPE (Canadian Council of Professional Engineers) y el IEEE Computer Society. Su generosidad nos ha permitido proporcionar el SWEBOK sin coste alguno (ver http://www.swebok.org/). El *comité ejecutivo profesional* se complementa con los directores de ISO/IEC JTC1/SC7 y con la iniciativa *Computing Curricula 2001*. El comité revisa y aprueba los planes del proyecto, dándole credibilidad. En general, se asegura que el esfuerzo del proyecto es relevante a las necesidades del mundo real.

La versión de prueba de la guía ha sido el resultado de extensivas revisiones y comentarios. En tres ciclos de revisión pública, un total de unos 500 revisores de 42 países proporcionaron aproximadamente 9.000 comentarios, los cuales están todos disponibles en `http://www.swebok.org/`. Para producir la versión actual, la versión de prueba (*Trial Version*) se utilizo ampliablemente durante un periodo durante el cual se generaron 17 artículos describiendo los beneficios aspectos de la guía al igual que otros aspectos en los que se debía mejorar. Una encuesta a través de la Web, recopilo información adicional: 573 personas de 55 países se registraron para la encuesta; 124 revisores de 51 países proporcionaron 1.020 comentarios. Se sugirieron otras mejoras a través de otros proyectos relacionados: *IEEE-CS/ACM Computing Curricula*, le proyecto *IEEE CS Certified Software Development Professional*, ISO/IEC JTC1/SC7 (estándares de ingeniería del software y sistemas); el comité de estándares de ingeniería del software del IEEE, la división de software de la ASQS (American Society for Quality), y una sociedad de ingenieros profesionales, la CCPE (Canadian Council of Professional Engineers).

CAMBIOS DESDE LA VERSIÓN DE PRUEBA

El objetivo general de la versión actual fue mejorar la legibilidad, consistencia y usabilidad de la guía. Esto implicó una reescritura todo el texto para hacer el estilo más consistente a lo largo de todo el documento. En varias ocasiones la descomposición de las áreas de conocimiento (AC) fue reordenada para hacerla más usable, pero siendo cuidadoso en que incluyese la información aprobada por consenso. Se actualizó la lista de referencias de manera que fuesen más fáciles de conseguir.

Tras el período de prueba se recomendó la reescritura de 3 AC. Se destacó que el AC de la construcción del software era difícil de aplicar en un contexto práctico. El AC de gestión fue percibido como próximo al concepto de gestión en general y no lo suficientemente específico a la ingeniería del software. Del AC de calidad se percibió que mezclaba la calidad en el proceso y calidad en el producto, y por tanto, también fue revisada.

Finalmente algunas AC fueron revisadas para quitar material contenido en otras AC.

LIMITACIONES

Aunque la guía ha pasado un elaborado proceso de desarrollo y revisión, las siguientes limitaciones en este proceso deben ser consideradas y expuestas:

- La ingeniería del software continua siendo infundida con nuevas tecnologías y nuevas prácticas. La aceptación de las nuevas técnicas crece y las antiguas descartadas. Los "temas globalmente aceptados" que se que han incluido en esta guía han sido cuidadosamente seleccionados. Aunque inevitablemente, esta selección tendrá que ser actualizada.

- El volumen de literatura publicado sobre la ingeniería del software es abundante y las referencias incluidas en esta guía no deberían ser consideradas como la selección definitiva sino como una selección razonable. Obviamente, hay excelentes autores y excelentes referencias que no están incluidos en esta guía. En este trabajo, la referencias seleccionadas lo fueron por estar en inglés, fácilmente accesibles, recientes, fáciles de leer y por cubrir los temas de las AC.

- Hay referencias importantes y relevantes que no han sido escritas en inglés que han sido omitidas del material seleccionado.

Además uno debe considerar que:

- La ingeniería del software es una disciplina emergente, esto es especialmente cierto si se compara con otras disciplinas más maduras. Por tanto, las fronteras entre las AC de la ingeniería del software y entre la ingeniería del software y otras disciplinas relacionadas continúan evolucionando.

Los contenidos de esta guía deben ser vistos como una informada y razonable caracterización del cuerpo de conocimiento de la ingeniería de software, y base de su futura evolución. Además, nótese que la guía no es un intento, ni intenta reemplazar o corregir ninguna de las leyes, normas o procedimientos definidos por organismos oficiales con respecto a la práctica y definición de la ingeniería, y de la ingeniería del sobre el particular.

Alain Abran
École de technologie supérieure

Coordinadores ejecutivos de la Guía al cuerpo de conocimiento de la ingeniería del software

James W. Moore
The MITRE Corporation

Pierre Bourque
École de Technologie Supérieure

Coordinadores de la Guía al cuerpo de conocimiento de la ingeniería del software

Robert Dupuis
Université du Québec à Montréal

Leonard Tripp
1999 President
IEEE Computer Society

***Director del* comité ejecutivo profesional, IEEE Computer Society (2001-2003)**

Diciembre 2004
La Web del proyecto SWEBOK es: http://www.swebok.org/

RECONOCIMIENTOS

El equipo de coordinación del SWEBOK reconoce la gratitud por la ayuda por parte de los miembros del comité ejecutivo profesional. La financiación de este proyecto se ha debido a las siguientes organizaciones: ACM, Boing, CCPE (Canadian Council of Professional Engineers), Construx Software, IEEE Computer Society, the MITRE Corporation, NIST (National Institute of Standards and Technology), NRC Canadá (National Research Council of Canadá), Rational Software, Raytheon Company, y SAP Labs (Canadá). El equipo también agradece a los miembros del panel de expertos. También queremos expresar nuestro agradecimiento por el trabajo inicial en la descripción de las áreas de conocimiento completadas por Imants Freibergs, Stephen Frezza, Andrew Gray, Vinh T. Ho, Michael Lutz, Larry Reeker, Guy Tremblay, Chris Verhoef, y Sybille Wolff. El equipo de coordinación también agradece a los cientos de revisores su valiosa contribución.

El equipo de coordinación también quiere agradecer por su contribución al proyecto a las siguientes personas: Mark Ardis, Yussef Belkebir, Michel Boivin, Julie Bonneau, Simon Bouchard, François Cossette, Vinh Duong, Gilles Gauthier, Michèle Hébert, Paula Hawthorn, Richard W. Heiman, Julie Hudon, Idrissa Konkobo, Rene Köppel, Lucette Lapointe, Claude Laporte, Luis Molinié, Hamdan Msheik, Iphigénie N'Diyae, Serge Oligny, Suzanne Paquette, Keith Paton, Dave Rayford, Normand Séguin, Paul Sinnett, Denis St-Pierre, Dale Strok, Pascale Tardif, Louise Thibaudeau, Dolores Wallace, Évariste Valery Bevo Wandji, y Michal Young.

Finalmente, estamos seguros que hay otras personas que han contribuido a esta guía directa o indirectamente, cuyos nombres han sido omitidos inadvertidamente. A estas personas, ofrecemos nuestro agradecimiento tácito y disculpas por haber omitido un reconocimiento explicito.

CAPITULO 1
INTRODUCCIÓN A LA GUÍA

A pesar de los millones de profesionales del software en el mundo y de la presencia ubicua del software en nuestra sociedad, sólo recientemente la ingeniería del software ha alcanzado el estado de disciplina ingenieril y reconocida profesión.

Alcázar un consenso por la profesión en un núcleo del cuerpo de conocimiento es un hito clave en todas las disciplinas y ha sido identificado por el IEEE Computer Society como crucial para la evolución hacia un status profesional. Esta guía, escrita bajo los auspicios del comité del ejercicio profesional, es parte de un proyecto multi-anual para alcanzar tal consenso.

¿QUÉ ES LA INGENIERÍA DEL SOFTWARE?

El IEEE Computer Society define la ingeniería del software como:
"(1) Aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software, es decir, la aplicación de la ingeniería al software.
(2) El estudio de los métodos en (1)". [2]

¿QUÉ ES UNA PROFESIÓN RECONOCIDA?

Para que la ingeniería del software sea una legítima disciplina y reconocida profesión, es imperativo un consenso sobre el cuerpo de conocimiento. Este hecho es bien ilustrado por Starr cuando define que puede considerarse como legítima disciplina reconocida profesión. En su libro, ganador del premio Pulitzer, sobre la historia de la profesión médica de los EE.UU, indica:

"La legitimización de una autoridad profesional envuelve tres afirmaciones distintivas: primero, todo conocimiento y competencias de un profesional han sido validadas por una comunidad de compañeros de profesión; segundo, el conocimiento validado consensuadamente se basa en fundamentos científicos racionales; y tercero, las opiniones profesionales y consejos están orientadas hacia valores fundamentales como la salud. Estos aspectos de legitimidad corresponden con los tipos de atributos - - generalmente cubiertos en el término 'profesión' "[3].

¿CUÁLES SON LAS CARACTERÍSTICAS DE UNA PROFESIÓN?

Gary Ford y Norman Gibbs estudió varias profesiones, incluyendo medicina, derecho, ingeniería y contabilidad. Concluyeron que la profesión de la ingeniería está caracterizada por varios componentes:

- Una educación profesional inicial en un currículum validado por una sociedad de acreditación.
- Registro de la correcta práctica por medio de una certificación voluntaria o licencia obligatoria.
- Habilidad espacial de desarrollo y una continua educación profesional.
- Soporte comunitario por medio de una sociedad profesional.
- Un compromiso con las normas de conducta a menudo prescritas en un *código de ética*.

Esta Guía contribuye a los primeros tres de estos componentes. La articulación del Cuerpo del Conocimiento es un paso esencial hacia el desarrollar una profesión porque representa un amplio consenso en lo que respecta a qué debería conocer un profesional de la ingeniería del software. Sin tal consenso, ningún examen de licenciatura puede ser validado, ningún plan de estudios puede preparar a una persona para un examen, y no se puede formular unos criterios para la acreditación de dicho plan de estudios. El desarrollo de un consenso es también un prerrequisito para la adopción de unas habilidades coherentes de desarrollo y de un programa de educación continua para los profesionales de una organización.

¿CUÁLES SON LOS OBJETIVOS DEL PROYECTO SWEBOK?

La guía no debería ser confundida con el cuerpo de conocimiento en sí mismo, el cual en la literatura publicada. El propósito de la guía es describir que parte del cuerpo de conocimiento es generalmente aceptada, organizar esa parte y proporcionar acceso a los temas de interés. Información adicional de que se entiende por "generalmente aceptado" se describe a continuación y en el Apéndice A.

La guía al cuerpo de conocimiento de ingeniería del software se estableció con los siguientes 5 objetivos:

1. Promover una visión consistente de la ingeniería del software en el mundo.
2. Clarificar la situación – y definir fronteras – de la ingeniería del software con respecto a otras disciplinas como la informática, gestión de proyectos, ingeniería informática y matemáticas.
3. caracterizar los contenidos de la disciplina de la ingeniería del software
4. Proporcionar al cuerpo de conocimiento de la ingeniería del software con los temas de interés

[2] "IEEE Standard Glossary of Software Engineering Terminology," IEEE std 610.12-1990, 1990.
[3] P. Starr, The Social Transformation of American Medicine, Basic Books, 1982, p. 15.

5. Proporcionar una base para el desarrollo planes de estudio, certificaciones individuales y materiales para licencias.

El primero de estos objetivos una visión consistente de la ingeniería del software fue proporcionada por el proceso de desarrollo consistente en aproximadamente 500 revisores de 42 países en la fase *Hombre de Piedra*, (1998-2001), la cual condujo a la versión de prueba; 120 revisores de 21 países en la fase *Hombre de Hierro* (2003, la cual genero la versión 2004. Información adicional sobre el proceso, se encuentra disponible en Prefacio y en la Web (http://www.swebok.org/). Se contactaron sociedades profesionales y agencias públicas involucradas con la ingeniería del software para que fueran conscientes del proyecto y se les invitó a participar en el proceso de desarrollo. Se reclutaron coordinadores asociados en América del Norte, los países del pacífico y Europa. Se hicieron presentaciones del proyecto en acontecimientos internacionales y se planificaron otros para el año posterior.

El segundo de los objetivos, el deseo de definir fronteras para la ingeniería del software, motiva la organización fundamental de esta guía. El material reconocido como perteneciente a la Ingeniería del software está organizado en las 10 áreas de conocimiento (AC) enumeradas en la Tabla 1. Cada una de estas AC es tratada como un capítulo de la guía.

Tabla 1 Áreas de conocimiento del SWEBOK

Requerimientos del software
Diseño del software
Construcción del software
Pruebas del software
Mantenimiento del software
Gestión de la configuración del software
Gestión en la ingeniería del software
Métodos y Herramientas de la ingeniería del software
Calidad del software

Al establecer la frontera, también es importante identificar que disciplinas comparten tal frontera, y a menudo, una intersección común con la ingeniería del software, A tal respecto, la guía reconoce otras 8 disciplinas relacionadas, enumeradas en la Tabla 2 (ver Capítulo 12, Disciplinas relacionadas con la ingeniería del software). Sin embargo, no es objetivo de la guía del SWEBOK caracterizar el conocimiento de las disciplinas relacionadas, sino el conociendo que es visto como específico a la ingeniería del software

Tabla 2 Disciplinas relacionadas

Ingeniería informática
Informática
Gestión
Matemáticas
Gestión de proyectos
Gestión de calidad
Ergonomía del software
Ingeniería de sistemas

ORGANIZACIÓN JERÁRQUICA

La organización de las AC en los capítulos proporciona el tercer objetivo del proyecto – una caracterización de los contenidos de la ingeniería del software. Especificaciones detalladas por el equipo de coordinadores editoriales a los coordinadores asociados en encuentra en el Apéndice A.

La guía utiliza una organización jerárquica para descomponer cada área de conocimiento en un conjunto de temas catalogados. Una descomposición en 2 o 3 niveles proporciona una manera razonable de encontrar los temas. La guía trata los temas seleccionados de una manera compatible con las escuelas de pensamiento mayoritarias y con las descomposiciones encontradas en las organizaciones, literatura y estándares. La descomposición no presupone ningún dominio de aplicación particular, forma de negocio, filosofía de gestión, métodos de desarrollo, etc. La extensión de cada tema es la justa para entender la naturaleza de los temas y para que el lector pueda referirse a la literatura de forma satisfactoria. Después de todo, el cuerpo de conocimiento se encuentra en el material referenciado y no en la guía en sí misma.

MATERIALES DE REFERENCIA Y MATRICES

Para proporcionar el acceso a los temas de interés de la guía – el cuarto de los objetivos del proyecto – la guía identifica material de referencia para cada AC, incluyendo capítulos de libro, artículos u otras fuentes de reconocido prestigio. Cada AC incluye una matriz relacionando la literatura con los temas. El total de la literatura citada intenta ser el adecuado para un graduado con 4 años de experiencia.

En esta edición de la guía, las referencias de todas las AC forman unas 500 páginas de material, lo cual era una de las especificaciones a la hora de crear el SWEBOK. Se puede argumentar que algunas AC, por ejemplo el diseño del software, se merecen más referencias que otras, y puede que este criterio se aplique a futuras ediciones de la guía.

Además, nótese que la guía no busca la completitud en sus referencias. Existe mucho material importante y relevante que no se ha sido incluido. El material fue en parte seleccionado por que cubre los temas descritos.

PROFUNDIDAD DEL TRATAMIENTO

Una de las cuestiones que surgieron desde el comienzo del proyecto, fue el nivel de detalle que la Guía debería proporcionar. El equipo del proyecto adaptó un enfoque que ayuda con el quinto de los requerimientos – proveer la base para el desarrollo curricular, certificaciones y licencias. El equipo editorial aplico el criterio de conocimiento generalmente aceptado, conocimiento consensuado, distinguiéndolo de conocimientos avanzados o de investigación (en base a la madurez) y de conocimiento especializado (en base a la generalidad de aplicación). La definición viene del *Project Management Institute* (PMI): "El conocimiento generalmente aceptado se aplica a la mayoría de los proyectos la mayoría del tiempo, y su amplio consenso valida su valor y efectividad"[4].

Especializado Prácticas utilizadas solamente en ciertos tipos de software	Generalmente aceptado Prácticas tradicionales establecidas que son recomendadas por muchas organizaciones.
	Avanzados y de investigación Practicas innovadoras probadas y usadas solo por algunas organizaciones y conceptos que están todavía siendo desarrollados y probados en organizaciones de investigación

Sin embargo, el término "generalmente aceptado", no implica que el conocimiento nombrado deba aplicarse uniformemente a todos los proyectos software – cada proyecto necesita determinarlos – pero implica que todos los ingenieros competentes deberían de estar equipados con estos conocimientos. Siendo más precisos, el conocimiento generalmente aceptado debería ser incluido en los materiales de estudio de licencias que los graduados deberían de adquirir tras cuatro años de experiencia laboral. Aunque este criterio es específico al estilo de educación en los EEUU y puede que no se aplique a otros países, se considera útil. Sin embargo, las dos definiciones de conocimiento generalmente aceptado deberían ser vistas como complementarias.

LIMITACIONES RELACIONADAS CON EL FORMATO DEL LIBRO

El formato en el cual este libro ha sido concebido tiene sus limitaciones. La naturaleza de los contenidos podría mostrarse mejor en formato de hipertexto, donde cada tema podría ser unido a otros temas que no sean el anterior y posterior a una lista.

[4] A Guide to the Project Management Body of Knowledge, 2000 ed., Project Management Institute, http://www.pmi.org/

Algunas fronteras entre AC, subáreas, etc. son a veces, arbitrarias, pero no hay que darle mucha importancia. En lo posible, se dan enlaces en el texto done son relevantes y útiles.

Los enlaces entre las AC, no son del tipo entrada-salida. Las áreas de conocimiento proporcionan vistas al conocimiento que uno debería poseer con respecto a cada AC en la ingeniería del software. La descomposición de la disciplina dentro de cada AC y el orden en el cual las áreas de conocimiento son presentadas, no tienen por qué integrarse con ningún método o modelo particular. Los métodos son descritos en las apropiadas AC dentro de la guía, y la guía en sí misma no es parte de ellos.

LAS ÁREAS DE CONOCIMIENTO (AC)

La Figura 1 describe los 11 capítulos y los temas importantes de cada uno de ellos. Las 5 primeras áreas de conocimiento son presentadas siguiendo el tradicional ciclo de vida en cascada. Sin embargo, esto no implica que la guía adopta o fomenta el ciclo de vida en cascada o ningún otro. Las subsecuentes AC se presentan en orden alfabético, y las disciplinas relacionadas se presentan en el ultimo capitulo.

ESTRUCTURA DE LAS DESCRIPCIONES DE LAS AC

Las áreas de conocimiento se estructuran como se describe a continuación.

En la introducción, de cada AC se proporciona una breve definición y una visión general de su ámbito y relaciones con otras áreas de conocimiento.

La descomposición de los temas constituye la parte fundamental de cada AC en áreas, subáreas y subtemas. Para cada tema o subtema, se da una breve descripción y referencias.

El material de referencia fue seleccionado por considerar que constituía la mejor presentación del conocimiento relacionado con un tema teniendo en cuenta las restricciones impuestas en la limitación de las referencias (ver arriba). Una matriz enlaza los temas con las referencias.

La última parte de la descripción de las AC es una lista de referencias recomendadas. El apéndice A de cada AC incluye sugerencias para los lectores que quieran profundizar en un tema particular. Los apéndices B presentan la lista de estándares más relevantes para cada área de conocimiento. Nótese que las citas entre corchetes "[…]" en el texto indican las referencias recomendadas, mientras que las que están en paréntesis "(…)" identifican las referencias usuales usadas al escribir o argumentar el texto. Las primeras se encuentran en la sección correspondiente del AC y las ultimas en el Apéndice A de cada AC.

A continuación se resumen las áreas de conocimiento y apéndices.

REQUERIMIENTOS DEL SOFTWARE (FIGURA 2, COLUMNA A)

Un requerimiento se define como una propiedad que debe exhibir el software para resolver algún problema del mundo real.

La primera subárea de conocimiento es *Fundamentos de los Requerimientos del Software*. Incluye las definiciones de requerimientos del software y sus principales tipos: producto vs. proceso, funcional vs. no funcional, propiedades emergentes. La subárea además describe la importancia de que los requerimientos cuantificables y distingue entre sistemas y requerimientos software.

La segunda subárea de conocimiento son los requerimientos del proceso, el cual introduce el proceso y orienta las 5 subáreas restantes y cómo la ingeniería de requerimientos encaja en otras con otros procesos de la ingeniería del software. Describe los modelos de proceso, actores del proceso, procesos de soporte y gestión, y la calidad mejora del proceso.

La tercera subárea, es la C*aptura de requisitos*, la cual se centra en de dónde vienen los requerimientos y cómo el ingeniero de software puede obtenerlos. Incluye las fuentes de los requerimientos y las técnicas de captura.

La cuarta subárea, *Análisis de requerimientos*, se centra en los procesos de análisis de requerimientos para:

- Detectar y resolver conflictos entre requerimientos
- Descubrir las fronteras del software y como deben interactuar con el entorno
- Elaborar los requerimientos del sistema a requerimientos software.

El análisis de requerimientos incluye su clasificación, el modelado conceptual, diseño arquitectural, asignación de requerimientos y negociación de requerimientos.

La quitan subárea es la *Especificación de requerimientos*, que típicamente se refiere a la producción de un documento o su equivalente electrónico, que puede ser sistemáticamente revisado, evaluado y aprobado. Para sistemas complejos, particularmente los sistemas con una parte substancial de componentes no-software, se pueden producir hasta 3 tipos diferentes de documentos: definiciones del sistema, especificación de requerimientos del sistema, y especificación de requerimientos software. La subárea, describe los 3 documentos y actividades asociadas.

La sexta subárea es la *Validación de Requerimientos*, cuyo objetivo, es descubrir problemas antes de asignar recursos a abordar los requerimientos. La validación de requerimientos concierne proceso de examinar los documentos de requerimientos y asegurarse de que definen el sistema correcto, es decir, el sistema que los usuarios esperan. Esta subdividido en las descripciones para llevar a cabo revisiones de requerimientos, prototipos y validación y aceptación de pruebas.

La séptima subárea son las *Consideraciones Practicas*, la cual describe los temas que necesitan ser entendidos en la práctica. El primer tema es la naturaleza del proceso iterativo de los requerimientos. Los tres temas siguientes son sobre la gestión del cambio, el mantenimiento de los requerimientos que reflejen el sistema a construir o ya construido. Incluye gestión de cambios, atributos de los requerimientos y trazas de los requerimientos. El último tema es la medición de los requerimientos.

DISEÑO DEL SOFTWARE (FIGURA 2, COLUMNA B)

Según la definición de IEEE [IEEE 610.12-90], el diseño es "el proceso de definir la arquitectura, componentes, interfaces y otras características de un sistema o componente" y "el resultado de [ese] proceso". El AC está dividido en 6 subáreas.

La primera subárea presenta los *Fundamentos del Diseño del Software*, el cual forma la base para entender el rol y ámbito del diseño del software. Estos son conceptos generales del software, el contexto del diseño del software, el proceso del diseño del software, las técnicas que permiten el diseño del software.

La segunda subárea agrupa los *Temas Clave en el Diseño del Software*. Incluye concurrencia, control y manejo de eventos, distribución de componentes, manejo de errores, excepciones y tolerancia a fallos, interacción y presentación y persistencia de datos.

La tercera subárea es la *Estructura del Software y la Arquitectura*, los temas son las estructuras arquitecturales y los puntos de vista, estilos arquitecturales, patrones de diseño, y finalmente, familias de programas y *frameworks*.

La cuarta subárea describe la *Calidad Evaluación de las del Diseño del Software*. Aunque existe una AC dedicada exclusivamente a la calidad del software, esta subárea se centra en los aspectos específicos del diseño. Estos aspectos son atributos de calidad, análisis de calidad y técnicas de evaluación y medición.

La quinta subárea son las *Notaciones del Diseño del Software*, que se dividen en descripciones estructurales y de comportamiento.

La última subárea describe las *Estrategias y Métodos del diseño del Software*. Primero, se describen estrategias generales, seguidas por métodos funcionales, métodos de

diseño orientados a objetos, métodos de diseño centrados en la estructura de datos, diseño basado en componentes y otros.

CONSTRUCCIÓN DEL SOFTWARE (FIGURA 2, COLUMNA C)

La construcción del software se refiere la creación de software mediante una combinación de codificación, verificación, pruebas unitarias, pruebas de integración, y depuración. Está dividida en tres subáreas.

La primera subárea son los *Fundamentos de la Construcción del Software*. Los tres primeros temas son los principios básicos de la construcción: minimización de la complejidad, anticipación al cambio, y la construcción y verificación.

La segunda subárea describe la *Gestión de la Construcción*. Los temas son la construcción de modelos, planificación de la construcción y la medición de la construcción.

La tercera subárea cubre las *Consideraciones Practicas*. Los temas son el diseño de la construcción, lenguajes de construcción, codificación, pruebas de construcción, reutilización, calidad de construcción e integración.

PRUEBAS DEL SOFTWARE (FIGURA 2, COLUMNA D)

Las pruebas de software se componen de la verificación dinámica del comportamiento de un programa con un conjunto finito de casos de pruebas, adecuadamente seleccionados del un infinito número de posibles ejecuciones del dominio.

Comienza con una descripción de los *Fundamentos de las Pruebas del Software*. Primero, se presenta la terminología relacionada con las pruebas, después se presentan los aspectos fundamentales de las pruebas, y finalmente, la relación de las pruebas con otras actividades.

La segunda subárea son los *Niveles de Pruebas*. Estos están divididos entre el objeto de la prueba y los objetivos de las pruebas.

La tercera subárea son las *Técnicas para Pruebas*. La primera categoría incluye las pruebas basadas en la intuición del probador/a y experiencia. El segundo grupo comprende las técnicas basadas en la especificación, seguido de las técnicas basadas en el código, y las técnicas relativas a la naturaleza de la aplicación. También se presenta cómo seleccionar y combinar las técnicas más apropiadas.

La cuarta subárea cubre las *Medidas relacionadas con las Pruebas*. Las medidas se agrupan en aquellas relacionadas con la evaluación del programa que se está probando y la evaluación de las pruebas realizadas.

La última subárea describe el *Proceso de Pruebas* e incluye las consideraciones prácticas y las actividades de pruebas.

MANTENIMIENTO DEL SOFTWARE (FIGURA 2, COLUMNA E)

Una vez en producción, se descubren anomalías, los entornos de trabajo cambian y aparecen nuevos requerimientos de trabajo. La fase del ciclo de vida mantenimiento comienza una vez entregado el sistema, sin embargo, las actividades de mantenimiento ocurren mucho antes. El AC de mantenimiento del software está dividido en 4 subáreas.

La primera presenta los *Fundamentos del Mantenimiento del Software*: definiciones y terminología, la naturaleza del mantenimiento, la necesidad del mantenimiento, los costes, la evolución del software, y las categorías de mantenimiento.

La segunda subárea agrupa los *Temas Clave del Mantenimiento del Software*. Estos comprenden temas técnicos, de gestión, estimación del coste de mantenimiento y la medición del mantenimiento.

La tercera subárea describe el *Proceso de Mantenimiento*. Los temas aquí presentados son el proceso de mantenimiento y las actividades de mantenimiento.

Las Técnicas para el Mantenimiento constituye la cuarta subárea. Estas incluyen la compresión del software, la reingeniería y la ingeniería inversa.

GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE (FIGURA 3, COLUMNA F)

La *Gestión de la Configuración del Software* (GCS) es la disciplina de la identificación del software en distintos puntos en el tiempo con el propósito de controlar los cambios sistemáticamente, y del mantenimiento de la integridad y trazabilidad de la configuración durante todo el ciclo de vida. Esta AC incluye 6 subáreas.

La primera subárea es la *Gestión del proceso de la GCS*. Cubre los temas del contexto organizacional para la GSC, restricciones y guías para la GCS, el plan de SGC, y el control del GCS.

La segunda subárea es la *Identificación en la Configuración del Software*, la cual establece los ítems a ser controlados, establece la identificación de esquemas, para los ítems y sus versiones, y establece las herramientas y técnicas usadas en la adquisición y gestión de los ítems controlados. Los primeros temas en esta subárea son la identificación de los ítems a ser controlados y las bibliotecas de software.

La tercera subárea es el *Control de Configuración del Software*, trata la gestiona de cambios durante el ciclo de

vida del software. Las áreas son: primero, solicitud, evaluación y aprobación de cambios; segundo, implementación de cambios de software; tercero, desviaciones y remisiones.

La cuarta área es *Registro del Estado de la Configuración*. Sus temas son la información sobre el estado de la configuración e informes sobre el estado de la configuración.

La quita área es *Auditoria de Configuración Software*. Se compone de auditoría de la configuración funcional, auditoría de la configuración física y auditorías de una Línea Base de software.

GESTIÓN DE LA INGENIERÍA DEL SOFTWARE (FIGURA 2, COLUMNA G)

El AC de de la Gestión de la Ingeniería del Software trata la gestión y medición de la ingeniería del software. Mientras la medición es aspecto importante en todas las AC, es aquí donde se presenta el tema de programas de medición. Hay seis subáreas en la gestión de ingeniería del software. Las 5 primeras cubren la gestión de proyectos software y la última describe los programas de medición software.

La primera subárea es *Iniciación y Definición del Alcance*, la cual comprende la determinación y negociación de los requisitos, estudios de viabilidad, y consideración y revisión de requisitos.

La segunda subárea es la *Planificación de Proyectos Software* e incluye la planificación del proceso, determinación de los entregables, esfuerzo, plazos y estimación de costes, asignación de recursos, gestión de riesgos, gestión de la calidad y gestión de planes.

La tercera subárea es la *Promulgación del Proyecto Software*. Los temas son los planes de implementación, gestión de contratos con proveedores, implementación de procesos de medición, monitorización del proceso, control del proceso e informes.

La cuarta subárea es *Revisión y Evolución*, la cual incluye los temas de determinación de la satisfacción de requisitos, y revisión y evaluación de la ejecución.

La quita subárea describe el Cierre: determinación del cierre y actividades del cierre.

Finalmente, la sexta subárea describe la *Medición de la Ingeniería del Software*, más concretamente, los programas de medición. Las mediciones de procesos y productos se describen en el AC de los procesos de la Ingeniería del Software. Muchas de las otras AC también describen mediciones específicas a sus AC. Los temas de esta subárea incluyen el establecimiento y mantenimiento de la medición, realización del proceso de medición y evaluación de las mediciones.

PROCESO DE LA INGENIERÍA DEL SOFTWARE (FIGURA 2, COLUMNA H)

El AC de proceso de la ingeniería del software se centra en la definición, implementación, evaluación, gestión, cambio y mejora del proceso de ingeniería del software. Está dividido en cuatro subáreas.

La primera subárea presenta el *Proceso de Implementación y Cambios*. Aquí los temas son infraestructura del proceso, ciclo de gestión del proceso software, modelos para el proceso de implementación y cambios y consideraciones practicas.

La segunda subárea trata la *Definición del Procesos* Incluye los modelos del ciclo de vida del software, procesos del ciclo de vida del software, notaciones para la definición de procesos, adaptación de procesos y automatización.

La tercera subárea es la *Evolución del Proceso*. Los temas incluyen modelos de evaluación del proceso y los métodos de evaluación del proceso.

La cuarta subárea describe las *Mediciones de Proceso y de Producto*. El proceso de ingeniería del software cubre genéricamente la medición de producto y proceso. Mediciones específicas a cada AC se describen en las respectivas AC. Los temas son la medición del proceso, la medición de productos software, calidad de los resultados de las medición, modelos de información software y técnicas de medición de procesos.

HERRAMIENTAS Y MÉTODOS EN LA INGENIERÍA DEL SOFTWARE (FIGURA 2, COLUMNA I)

El AC de Herramientas y Métodos de la ingeniería del software incluye ambas, herramientas de la ingeniería del software y métodos de la ingeniería del software.

Las subárea de *Herramientas de la Ingeniería del Software* utiliza la misma estructura que la guía en sí misma, con un tema por cada una de las otras nueve AC de la ingeniería del software. Se añade un tema adicional: cuestiones varias sobre herramientas como técnicas de integración de herramientas que son potencialmente aplicables a todo tipo de herramientas.

La subárea de *Métodos de Ingeniería del Software* se divide en cuatro subsecciones: métodos heurísticos que tratan aproximaciones informales, métodos formales basados en aproximaciones matemáticas, métodos de prototipado tratando varias formas de prototipados.

CALIDAD DEL SOFTWARE (FIGURA 2, COLUMNA J)

El AC de la calidad del software se ocupa de las consideraciones sobre la calidad del software las cuales transcienden los procesos del ciclo de vida del software. Al ser la calidad del software un tema ubicuo a toda la

ingeniería del software, también es considerada en muchas otras AC, por lo que el lector notará referencias a otras AC en toda esta AC.

La primera subárea describe los *Fundamentos de la Calidad del Software* como la ética y cultura de la ingeniería del software, valor y coste de la calidad, modelos y características de la calidad y la mejora de la calidad.

La segunda subárea cubre los *Procesos de Gestión de la Calidad del Software*. Aquí los temas son el aseguramiento de la calidad, verificación y validación, y revisión y auditorias.

La tercera y última subárea describe las consideraciones prácticas relacionadas con la calidad del software. Los temas son requerimientos de calidad del software, caracterización de defectos, técnicas de gestión de la calidad del software, y medición de la calidad del software.

DISCIPLINAS RELACIONADAS DE LA INGENIERÍA DEL SOFTWARE (FIGURA 2, COLUMNA K)

El último capítulo se titula Disciplinas relacionadas de la Ingeniería del Software. Para circunscribir la ingeniería del software, es necesario identificar las disciplinas con las que la ingeniería del software comparte una frontera común. Este capítulo identifica en orden alfabético, estas disciplinas relacionadas. Para cada disciplina relacionada, y mediante consenso, se identifica:

- Una definición informativa (donde sea posible)
- Una lista de Áreas de Conocimiento

Las disciplinas relacionadas son

Ingeniería de ordenadores	Gestión de proyectos
Informática	Gestión de la Calidad
Gestión	Ergonomía
Matemáticas	Ingeniería de sistemas

APÉNDICES

APÉNDICE A. DESCRIPCIÓN DE LAS ESPECIFICACIONES EN LAS AC

El apéndice describe las especificaciones proporcionadas por el equipo editorial a los editores asociados para el contenido, referencias recomendadas, formato, y estilo de las descripciones de las AC.

APÉNDICE B. EVOLUCIÓN DE LA GUÍA

El segundo apéndice describe la propuesta de la evolución de la Guía. La Guía del 2004 es la edición actual, la cual continuara evolucionando para alcanzar las necesidades de la comunidad de la ingeniería del software. La planificación de la evolución no está todavía completada, pero un esquema provisional se proporciona en el apéndice. En el momento de esta escritura, este proceso ha sido aprobado por el Comité Ejecutivo Profesional y comunicado a la junta de gobierno del *IEEE Computer Society*; sin embargo no ha sido ni financiada ni implementada.

APÉNDICE C. ASIGNACIÓN DE ESTÁNDARES A AC

El tercer apéndice es una tabla comentada de los estándares más relevantes, principalmente de IEEE e ISO, asignados a las AC de la Guía del SWEBOK.

APÉNDICE D. ÍNDICES DE BLOOM

Como ayuda, principalmente a los creadores de planes de estudio (y otros usuarios), para alcanzar el quito objetivo, el cuarto apéndice clasifica cada tema con una de las categorías pedagógicas atribuidas a Benjamin Bloom. El concepto es que los objetivos educacionales pueden ser clasificados en seis categorías incrementando la profundidad: conocimiento, comprensión, aplicación, análisis, síntesis y evaluación. Los resultados de este ejercicio para todas las AC se encuentran en el Apéndice D. Este apéndice no debe ser visto como una clasificación definitiva, sino como un punto de partida.

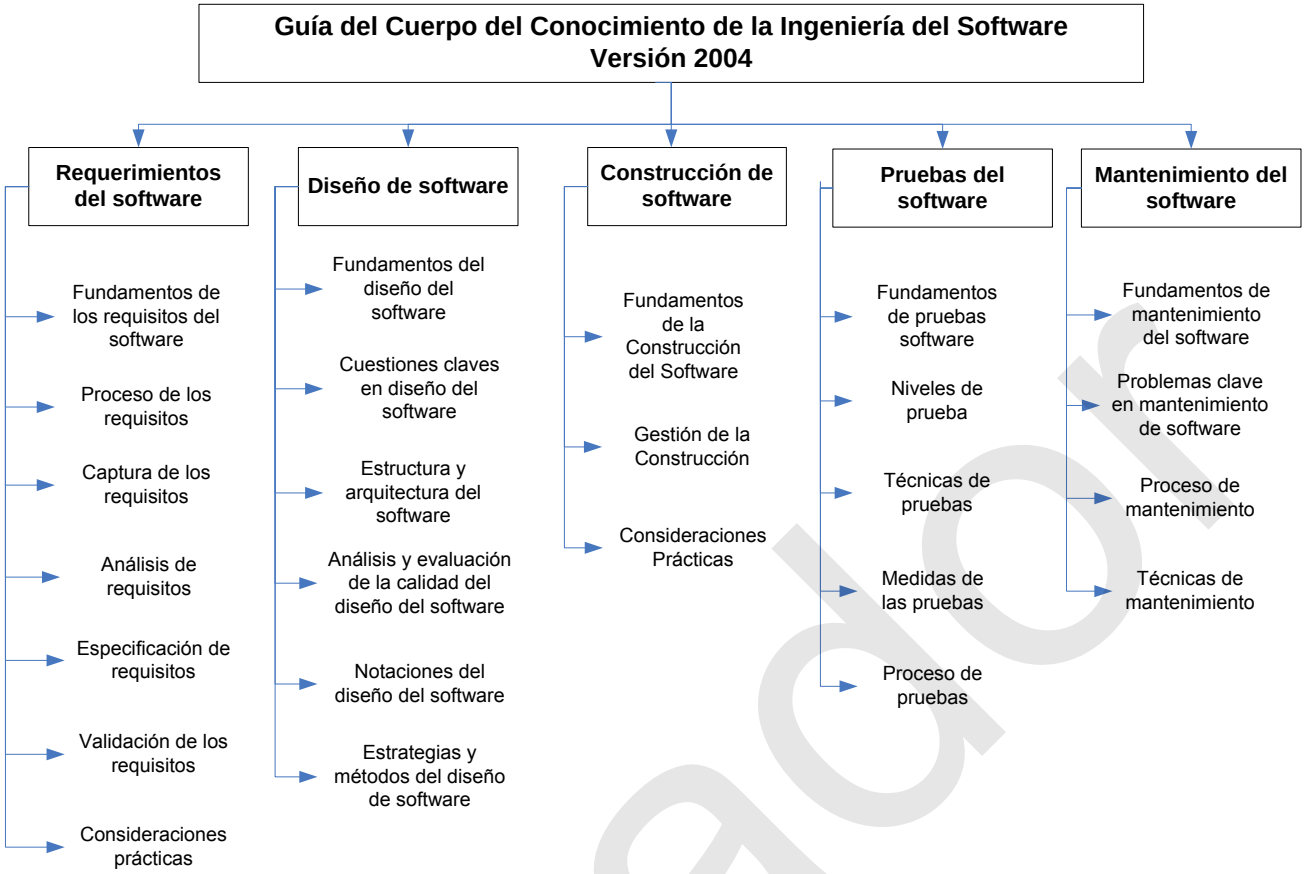


Figura 2: Las 5 primeras áreas de conocimiento

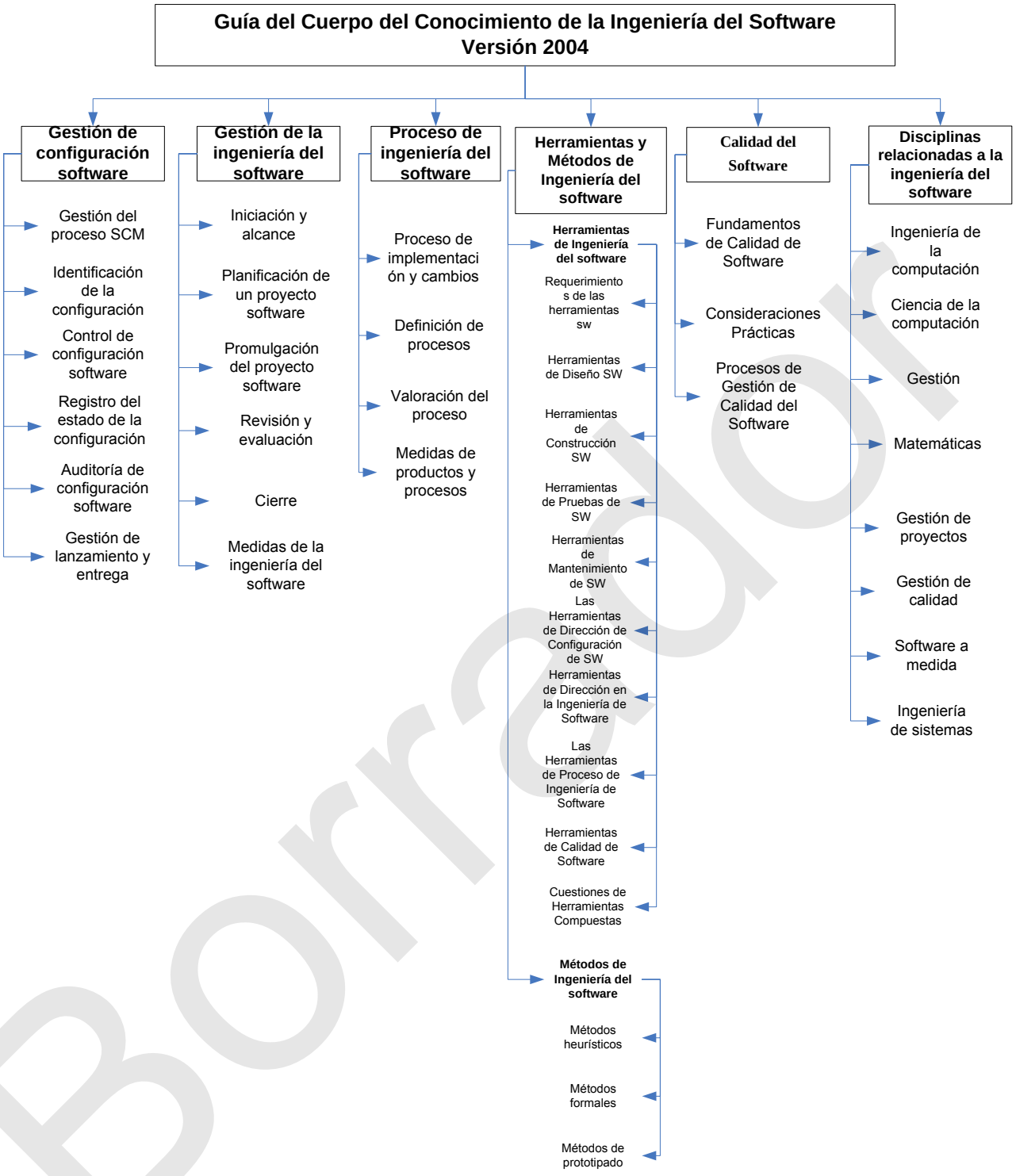


Figura 3: Las 6 últimas áreas de conocimiento

CAPITULO 2
REQUERIMIENTOS DE SOFTWARE

ACRÓNIMOS

DAG	Grafo Acíclico Dirigido complejo
FSM	Medida Funcional del Tamaño
INCOSE	Consejo Internacional sobre la Ingeniería de Sistemas
SADT	Análisis Estructurados y Técnicas de Diseño

INTRODUCCIÓN

El área del conocimiento de los requisitos del software (KA) se refiere al análisis, a la especificación, y a la validación de los requisitos del software. Está extensamente reconocido dentro de la industria del software que los proyectos de la ingeniería de software son críticamente vulnerables cuando estas actividades se realizan mal.

Los requisitos del software expresan las necesidades y los apremios colocados en un producto de software que contribuye a la solución de un cierto problema del mundo real. [Kot00] El término "ingeniería de requisitos" es ampliamente utilizado en el campo para denotar la dirección sistemática de requisitos. Aunque por razones de consistencia, este término no será utilizado en la guía, pues se ha decidido que el uso del término "ingeniería" para las actividades con excepción de la tecnología de dotación lógica debe ser evitado en esta edición de la guía.

Por la misma razón, tampoco se utilizará "ingeniero de los requisitos," un término que aparece en algo de la literatura. En su lugar se utilizará el término "Ingeniero de Software" o, en algunos casos específicos, "especialista de los requisitos", el último donde el papel en la pregunta es realizado generalmente por un individuo con excepción de una Ingeniería de Software. Esto no implica, sin embargo, que una Ingeniería de Software no podría realizar la función.

La interrupción de KA es ampliamente compatible con secciones de IEEE 12207 que se refieren a actividades de requisitos. (IEEE12207.1-96)

Un riesgo inherente en la interrupción propuesta es que un proceso en cascada puede ser deducido. Para evitar esto, los procesos de requisitos de la subzona 2, se diseñan para proporcionar una descripción de alto nivel del proceso de los requisitos precisando los recursos y los apremios bajo los cuales el proceso funciona y los cuales actúan para configurarlo.

Una descomposición alternativa podía utilizar una estructura basada en producto (requisitos del sistema, requisitos del software, prototipos, casos de uso, y así sucesivamente). Las interrupciones basadas en procesos reflejan el hecho de que el proceso de los requisitos, si es acertado, se debe considerar como proceso que implica actividades complejas, firmemente unidas (ambas secuencial y concurrentemente), más que una única actividad discreta realizada al principio de un proyecto de desarrollo de software.

El KA de los requisitos del software se relaciona de cerca con el Diseño del software, pruebas, mantenimiento del software, gerencia de la configuración del software, tecnología de dotación lógica, proceso de la tecnología de dotación lógica, y KAs de Calidad de software.

INTERRUPCIÓN DE LOS ASUNTOS PARA LOS REQUISITOS DEL SOFTWARE

1. Fundamentos de los requisitos del software

1.1. Definición de un requisito del software

Básicamente, un requisito del software es una característica que se debe exhibir para solucionar un cierto problema en el del mundo real. La guía se refiere a requisitos de "software" porque se refiere a los problemas que se tratarán por el software. Por lo tanto, un requisito del software es una característica que se debe exhibir por el software desarrollado o adaptado para solucionar un problema particular. El problema puede ser automatizar la parte de una tarea de alguien que utilizará el software, para apoyar los procesos del negocio de la organización que ha comisionado el software, a corregir los defectos del software existente, al control de dispositivos, y muchos más. El funcionamiento de los usuarios, los procesos del negocio, y los dispositivos es típicamente complejo. Por extensión, por lo tanto, los requisitos de software son típicamente una combinación compleja de requisitos de diversa gente en diversos niveles de una organización y del ambiente en el cual el software funcionará.

Una característica esencial de todos los requisitos del software es que sean comprobables. Puede ser difícil o costoso verificar ciertos requisitos del software. Por ejemplo, la verificación del requisito del rendimiento de procesamiento en el centro de la llamada puede hacer necesario el desarrollo del software de la simulación. Los requisitos del software y el personal de la calidad del software deben asegurarse de que los requisitos se puedan verificar dentro de los apremios disponibles del recurso.

Los requisitos tienen otras cualidades además de las características del comportamiento que expresan. Ejemplos comunes incluyen una tasa de prioridad para permitir compensaciones frente a recursos finitos y un valor del estado para permitir que el progreso del proyecto sea supervisado. Típicamente, los requisitos de software se identifican únicamente de modo que puedan estar sujetos al control de configuración del software y manejados sobre el ciclo vital entero del software. [Kot00; Pfl01; Som05; Tha97]

1.2. Producto y requisitos del proceso

Se puede dibujar una distinción entre los parámetros del producto y los parámetros del proceso. Los parámetros del producto son requisitos en software para ser convertido (por ejemplo, "El software verificará que un estudiante resuelva todos los requisitos previos antes de que él o ella se coloque para un curso.").

Un parámetro de proceso es esencialmente un constreñimiento en el desarrollo del software (por ejemplo, "el software será escrito en el Ada."). Éstos se conocen a veces como requisitos de proceso.

Algunos requisitos del software generan requisitos de proceso implícitos. La opción de la técnica de la verificación es un ejemplo. Otro puede ser el uso de técnicas rigurosas de análisis (tales como métodos formales de la especificación) para reducir las averías que pueden conducir a una inadecuada confiabilidad. Los requisitos de proceso pueden también ser impuestos directamente por la organización del desarrollo, su cliente, o terceros tales como un regulador de seguridad [Kot00; Som97].

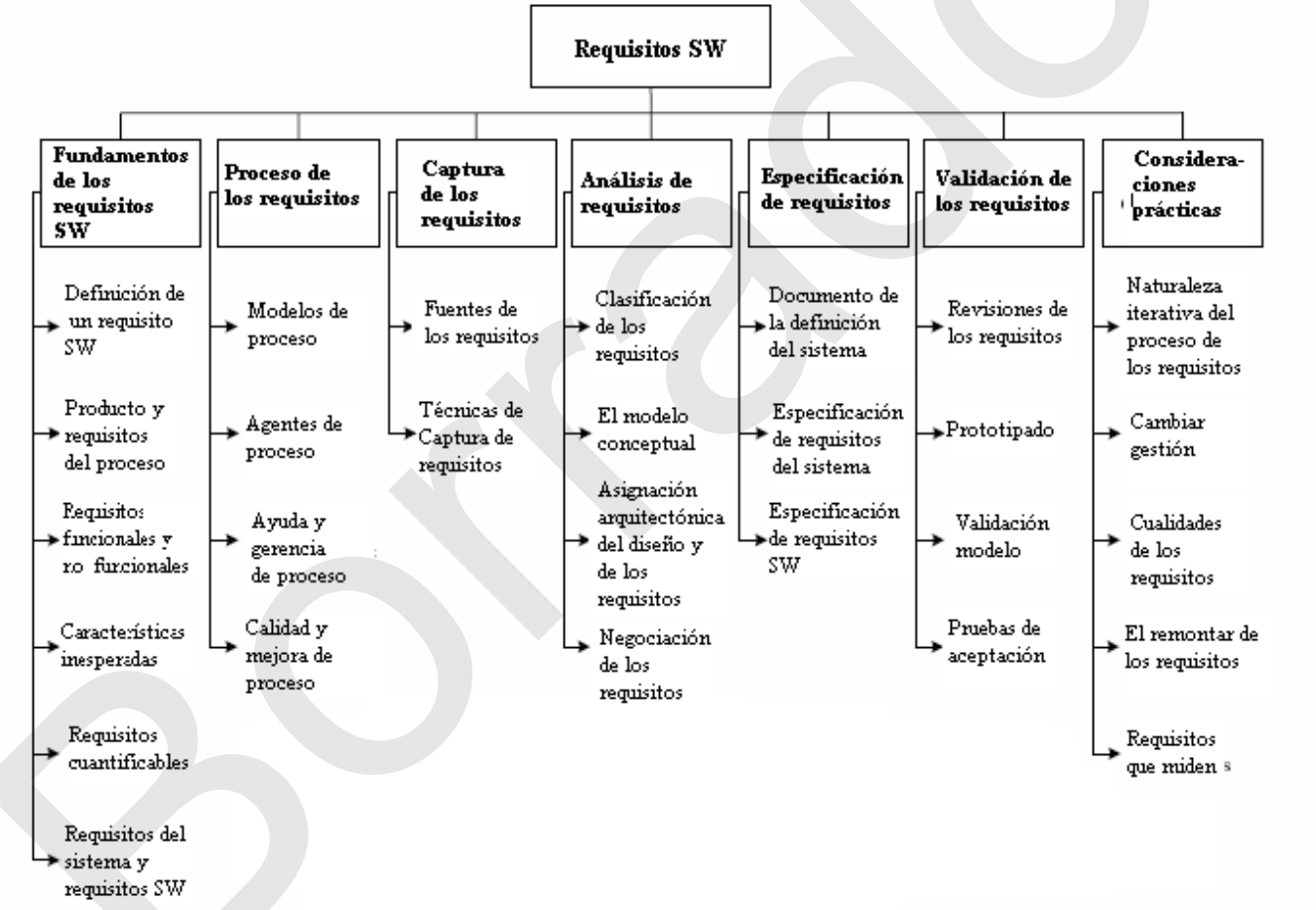


Figura 1: Descomposición de materias para la KA Requisitos del Software

1.3. Requisitos funcionales y no funcionales

Los requisitos funcionales describen las funciones que el software va a ejecutar; por ejemplo, ajustarse a un formato de texto o modular una señal. Se conocen también como capacidades.

Los requisitos no funcionales son los que actúan para obligar la solución. Los requisitos no funcionales se conocen a veces como apremios o requisitos de calidad. Pueden ser clasificados más a fondo según si son requisitos de funcionamiento, requisitos de capacidad de mantenimiento, requisitos de seguridad, requisitos de confiabilidad, o uno de muchos otros tipos de requisitos del software. Estos asuntos también se discuten en KA de la calidad del software. [Kot00; Som97]

1.4. Características inesperadas

Algunos requisitos representan características inesperadas del software- esto es, los requisitos que no pueden ser tratados por un solo componente, pero que su satisfacción va a depender de cómo todos los componentes de software ínter operan. El requisito del rendimiento de procesamiento para un centro de llamadas, por ejemplo, dependería de cómo el sistema de teléfono, el sistema de información, y los operadores obraron recíprocamente bajo condiciones de funcionamiento reales. Las características inesperadas son crucialmente dependientes en la arquitectura del sistema. [Som05]

1.5. Requisitos cuantificables

Los requisitos del software se deben indicar tan clara e inequívocamente como sea posible, y cuantitativamente. Es importante evitar requisitos vagos e inverificables que dependen para su interpretación del juicio subjetivo ("el software será confiable"; "el software será de uso fácil"). Esto es particularmente importante para los requisitos no funcionales. Dos ejemplos de requisitos cuantificados son los siguientes: el software para un centro de llamadas debe aumentar el rendimiento del procesamiento del centro un 20%; y un sistema tendrá una probabilidad de generar un error fatal durante cualquier hora de operación menos de $1 * 10^{-8}$. El requisito de rendimiento de procesamiento está a un alto nivel y necesitará ser derivado en un número de requisitos detallados. El requisito de la confiabilidad determinará firmemente la arquitectura del sistema. [Dav93; Som05]

1.6. Requisitos del sistema y requisitos del software

En este asunto, el sistema significa "una combinación recíproca de los elementos para lograr un objetivo definido. Éstos incluyen el hardware, software, soporte lógico inalterable, gente, información, técnicas, instalaciones, servicios, y otros elementos de apoyo," según lo definido por el consejo internacional sobre la ingeniería de sistemas (INCOSE00).

Los requisitos del sistema son los requisitos para el sistema en su totalidad. En un sistema que contiene componentes de software, los requisitos del software se derivan de los requisitos del sistema.

La literatura sobre requisitos llama a veces a los requisitos del sistema "exigencias del consumidor." La guía define "exigencias del consumidor" de una manera restricta como requisitos de los clientes o de los usuarios finales del sistema. Los requisitos del sistema, por el contrario, abarcan los requisitos del usuario, requisitos de otros tenedores de apuestas (por ejemplo autoridades reguladoras), y requisitos sin un origen identificable.

2. Proceso de los requisitos

Esta sección introduce el proceso de los requisitos del software, orientando las cinco subzonas restantes y demostrando cómo el proceso de los requisitos encaja con el proceso de ingeniería del software. [Dav93; Som05]

2.1. Modelos de proceso

El objetivo de este asunto es proporcionar una comprensión de que el proceso de los requisitos

- No es una actividad anticipada discreta del ciclo vital del software, sino un proceso iniciado en principio de un proyecto y a continuación refinado a través del ciclo vital
- Identifica los requisitos del software como elementos de configuración, y los maneja usando las mismas prácticas de gerencia de la configuración del software como otros productos de los procesos del ciclo vital del software
- Necesita ser adaptado a la organización y al contexto del proyecto

Particularmente, el asunto se refiere a cómo las actividades de análisis, especificación, y la validación se configuran para diversos tipos de proyectos y apremios. El asunto también incluye las actividades que proporcionan la entrada en el proceso de los requisitos, por ejemplo estudios de la comercialización y de viabilidad. [Kot00; Rob99; Som97; Som05]

2.2. Agentes de proceso

Este asunto introduce los papeles de la gente que participa en el proceso de los requisitos. Este proceso es fundamental interdisciplinario, y el especialista de los requisitos necesita mediar entre el dominio del tenedor de apuestas y el de la tecnología de dotación lógica. Hay mucha gente implicada además del especialista de los requisitos, cada uno de ellos tiene una función en el software. Los tenedores de apuestas variarán según los proyectos, pero incluyen siempre usuarios/operadores y clientes (quiénes no necesitan ser iguales). [Gog93]

Los ejemplos típicos de los tenedores de apuestas del software incluyen (pero no restringen)

- Usuarios: Este grupo abarca a los que utilizan el software. Es a menudo un grupo heterogéneo que abarca a gente con diversos papeles y requisitos.
- Clientes: Este grupo abarca a los que han comisionado el software o quién representa el blanco de mercado del software.
- Analistas de mercado: Un producto del mass-market no tendrá un cliente que comisiona, de modo que la gente de comercialización a menudo necesita establecer lo que el mercado necesita y actuar como clientes.
- Reguladores: Muchos dominios de aplicación como por ejemplo el transporte público o la banca son regulados. El software en estos

dominios debe conformarse con los requisitos de las autoridades reguladoras.

- Ingenieros de software: Estos individuos tienen un interés legítimo en beneficiarse del desarrollo del software, por ejemplo, reutilizando componentes en otros productos. Si, en este escenario, un cliente de un producto particular tiene requisitos específicos que comprometen el potencial para la reutilización de componentes, los ingenieros de software deben pesar cuidadosamente sus propios intereses contra los del cliente.

No será posible satisfacer perfectamente los requisitos de cada stakeholder, y es el trabajo de los ingenieros de software negociar las compensaciones que son aceptables a los stakeholder principales y dentro de presupuestario, técnico, regulador, y otros apremios. Un requisito previo para esto es que identifiquen a todos los stakeholder, la naturaleza de su "interés" analizada, y sus requisitos sacados. [Dav93; Kot00; Rob99; Som97; You01]

2.3. Ayuda y gerencia de proceso

Este asunto introduce los recursos de la gerencia de proyecto requeridos y consumidos por el proceso de los requisitos. Él establece el contexto para la primera subzona (definición de la iniciación y del alcance) de la tecnología de dotación lógica KA de la gerencia. Su propósito principal es hacer el acoplamiento entre las actividades de proceso identificadas en 2.1 y los temas de coste, recursos humanos, entrenamiento, y herramientas. [Rob99; Som97; You01]

2.4. Calidad y mejora de proceso

Este asunto se refiere al gravamen de la calidad y de la mejora del proceso de los requisitos. Su propósito es acentuar el papel dominante que los requisitos procesan en términos de coste y puntualidad de un producto de software, y de la satisfacción del cliente con ello [Som97]. Ayudará a orientar el proceso de los requisitos con estándares de calidad y modelos de mejora de proceso para el software y los sistemas. El proceso de calidad y la mejora se relacionan de cerca con la calidad del software y la tecnología de dotación lógica KA. De interés particular están las aplicaciones de calidad del software, sus cualidades y medida, y la definición de proceso de software. Este punto abarca

- Cobertura de proceso de los requisitos mediante estándares y modelos de la mejora
- Medidas de proceso de los requisitos y benchmarking
- Plan de mejora y puesta en práctica [Kot00; Som97; You01]

3. Captura de los requisitos

[Dav93; Gog93; Lou95; Pfl01]

La captura de los requisitos se refiere a de donde vienen los requisitos del software y cómo el ingeniero de software puede recogerlos. Es la primera etapa en la construcción de una comprensión del problema que el software requiere solucionar. Es fundamental una actividad humana, y es donde identifican a los stakeholders y las relaciones se establecen entre el equipo del desarrollo y el cliente. También se conoce como "descubrimiento de los requisitos," y "adquisición de los requisitos."

Uno de los aspectos fundamentales de la buena ingeniería de software es que haya buena comunicación entre los usuarios del software y los ingenieros. Antes de que comience el desarrollo, los especialistas de requisitos pueden formar el conducto para esta comunicación. Deben mediar entre el dominio de los usuarios del software (y otros stakeholders) y el mundo técnico del ingeniero de software.

3.1. Fuentes de los requisitos
[Dav93; Gog93; Pfl01]

Los requisitos tienen muchas fuentes en software típico, y es esencial que todas las fuentes potenciales estén identificadas y evaluadas para su impacto en él. Este asunto se diseña para promover el conocimiento de las varias fuentes de los requisitos del software y de los armazones para manejarlos. Los puntos principales cubiertos son

- Metas: El término meta (a veces llamada "preocupación de negocio" o "factor crítico del éxito") se refiere a los objetivos totales, de alto nivel del software. Las metas proporcionan la motivación para el software, pero a menudo son formuladas vagamente. Es necesidad que los ingenieros de software presten atención particular a determinar el valor (concerniente a prioridad) y coste de metas. Un estudio de viabilidad es una manera relativamente barata de hacer esto. [Lou95].
- Conocimiento del dominio: Los ingenieros de software necesitan adquirir, o tener disponible, conocimiento sobre el uso del dominio. Esto permite deducir el conocimiento que los stakeholders no articulan, determinar las compensaciones que serán necesarias en requisitos que están en conflicto, y, a veces, para actuar como campeón del "usuario".
- Stakeholders (véase a agentes de proceso del asunto 2.2). Mucho software se ha probado insatisfactorio porque había tensionado los requisitos de un grupo de stakeholders a expensas de los de otros. Por lo tanto, se entrega el software que es difícil de utilizar o que derriba las estructuras culturales o políticas de la organización del cliente. Los ingenieros de software necesitan identificar, representar, y manejar "puntos de vista" de muchos diversos tipos de tenedores de apuestas. [Kot00]
- El entorno operacional. Los requisitos serán derivados del ambiente en el cual el software será ejecutado. Éstos pueden ser, por ejemplo, el medir el tiempo en tiempo real del software o de la interoperabilidad en un ambiente de la oficina. Éstos deben ser buscados activamente, porque pueden afectar grandemente a la viabilidad y el coste del software, y restringen las opciones de diseño. [Tha97]
- El entorno de la organización. El software se requiere a menudo para apoyar un proceso del negocio, la selección del cual se puede condicionar por la estructura, cultura, y política interna de la organización. Los ingenieros de software necesitan ser sensibles a éstos, puesto que, el nuevo software no debe forzar generalmente cambios imprevistos en el proceso del negocio.

3.2. Técnicas de Captura de requisitos
[Dav93; Kot00; Lou95; Pfl01]

Una vez que se hayan identificado las fuentes de los requisitos, ingenieros de software pueden comenzar a sacar requisitos de ellos. Este asunto se concentra en las técnicas para conseguir que los stakeholders articulen sus requisitos. Es un área muy difícil e ingenieros de software necesitan sensibilizarse al hecho que (por ejemplo) los usuarios pueden tener dificultad para describir sus tareas, puede dejar la información importante sin especificar, o pueden estar poco dispuestos o cooperar. Es particularmente importante entender que la captura no es una actividad pasiva, y que, ingenieros de software tienen que trabajar difícilmente para sacar la información adecuada. Existe un número de técnicas para hacer esto, las principales son [Gog93]

- Entrevistas, medios "tradicionales" de sacar requisitos. Es importante entender las ventajas y limitaciones de las entrevistas y cómo deben ser conducidas.
- Escenarios, valiosos medios para proporcionar contexto a las exigencias del consumidor. Proporcionan un marco para preguntas sobre tareas del usuario permitiendo preguntas "y si" y "cómo se hace esto". El tipo más común de escenario es el caso del uso.
- Prototipos, una herramienta valiosa para clarificar requisitos confusos. Pueden actuar de una manera similar a los escenarios, proveyendo a los usuarios un contexto dentro del cual pueden entender mejor qué información necesitan proporcionar. Hay una amplia gama de técnicas de prototipado, maquetas de versiones de pruebas beta de los diseños de la pantalla del software, y un traslado fuerte de su uso para captura de los requisitos y el uso de prototipos para la validación de los requisitos (véase el asunto 6.2 Prototipado).
- Reuniones. El propósito de éstas es intentar alcanzar un efecto aditivo por el que un grupo de gente puede obtener más penetración en los requisitos del software que trabajando individualmente. Ellos pueden inspirarse y refinar las ideas que pueden ser difíciles de traer a la superficie usando entrevistas. Otra ventaja es que dejan a los stakeholders reconocer donde hay requisitos en conflicto. Cuando se aplica bien, esta técnica puede resultar en un rico y constante sistema de requisitos que difícilmente sería realizable de otro modo. Sin embargo, las reuniones necesitan ser dirigidas cuidadosamente (por lo tanto es necesario un facilitador) para evitar que una situación ocurra donde las capacidades críticas del equipo son erosionadas por lealtad del grupo, o los requisitos que reflejan las

preocupaciones de algunos favorecen la gente abierta (y quizás mayor) en detrimento de otros.

- Observación. La importancia del contexto del software dentro del ambiente de organización ha conducido a la adaptación de las técnicas de observación para captura de los requisitos. Los ingenieros de software aprenden sobre las tareas del usuario sumergiéndose en la observación de cómo los usuarios obran recíprocamente con su software. Estas técnicas son relativamente costosas, pero son instructivas porque ilustran que muchas tareas del usuario y procesos del negocio son demasiado sutiles y complejos para que sus agentes los describan fácilmente.

4. Análisis de requisitos
[Som05]

Este asunto se refiere al proceso de analizar requisitos para

- Detectar y resolver los conflictos entre los requisitos
- Descubrir los límites del software y cómo debe obrar recíprocamente con su ambiente
- Elaborar los requisitos del sistema para derivar requisitos software

La vista tradicional del análisis de requisitos ha sido que esté reducida a modelado conceptual utilizando uno de varios métodos de análisis tales como Análisis Estructurados y Técnicas de Diseño (SADT). Mientras que el modelado conceptual es importante, nosotros incluimos la clasificación de requisitos para ayudar a informar a compensaciones entre los requisitos (clasificación de los requisitos) y el proceso de establecer estas compensaciones (negociación de los requisitos). [Dav93]

El cuidado se debe tomar para describir requisitos con exactitud, suficientemente como para permitir que los requisitos sean validados, su implementación sea verificada, y sus costes estimados.

4.1 Clasificación de los requisitos
[Dav93; Kot00; Som05]

Los requisitos se pueden clasificar en un número de dimensiones. Los ejemplos incluyen:

- Si el requisito es funcional o no funcional (véase el asunto 1.3 funcional y requisitos no funcionales).
- Si el requisito está derivado de uno o más requisitos de alto nivel o una propiedad emergente (véase las características inesperadas del asunto 1.4) o está impuesto directamente ante el software por un tenedor de apuestas u otra fuente.
- Si el requisito está en el producto o proceso. Los requisitos en el proceso pueden obligarnos a la opción del contratista, a adaptar el proceso de la ingeniería del software o los estándares que se adherirán.
- La prioridad del requisito. Generalmente cuanta más alta es la prioridad, más esencial es el requisito para satisfacer las metas finales del software. A menudo clasificado en una escala de punto fijo tal como obligatorio, altamente deseable, deseable u opcional, la prioridad a menudo tiene que ser equilibrada con el coste de desarrollo e implementación.
- El alcance del requisito. El alcance se refiere al grado al cual un requisito afecta el software y componentes software. Algunos requisitos, particularmente los no funcionales, tienen un alcance global que no puede ser asignado a un componente discreto. Por lo tanto, el requisito con alcance global puede afectar fuertemente a la arquitectura del software y el diseño de muchos componentes, mientras que uno puede con un alcance estrecho ofrecer un número de opciones del diseño y no afectar demasiado a la satisfacción de otros requisitos.
- Volatilidad/estabilidad. Algunos requisitos cambiarán durante el ciclo de vida del software, y se igualarán durante el proceso de desarrollo. Es útil hacer estimaciones de la probabilidad de que se puedan hacer cambios. Por ejemplo, en actividades bancarias, los requisitos para las funciones para calcular y para acreditar los intereses en las cuentas son probablemente más estables que un requisito para mantener un tipo particular de cuenta exenta de impuestos. Lo anterior refleja una característica fundamental del dominio de las actividades bancarias (esas cuentas pueden ganar intereses), mientras que el último se puede dejar obsoleto por un cambio de gobierno. Señalar requisitos potencialmente volátiles puede ayudar al ingeniero del software a establecer un diseño que sea más tolerante al cambio.

Otras clasificaciones pueden ser apropiadas, dependiendo de la práctica normal y del uso que se le dé.

Hay un desfase fuerte entre la clasificación de los requisitos y las cualidades de los requisitos (véase las cualidades de los requisitos del punto 7.3).

4.2. El modelo conceptual
[Dav93; Kot00; Som05]

El desarrollo de modelos de un problema del mundo real es clave para el análisis de requisitos del software. Su propósito es ayudar a entender el problema, más que iniciar el diseño de la solución. Por lo tanto, modelos conceptuales abarcan modelos de entidades del dominio del problema, configurados para reflejar sus relaciones y dependencias con el mundo real.

Varias clases de modelos pueden ser desarrollados. Éstos incluyen datos y controlan flujos, modelos de estado, rastros de evento, interacciones de usuario, modelos de

objeto, modelos de datos y muchos otros. Los factores que influencian la opción del modelo incluyen

- La naturaleza del problema. Algunos tipos de software exigen que ciertos aspectos estén analizados rigurosamente. Por ejemplo, controlar el flujo y los modelos de estado son probablemente más importantes para el software en tiempo real que para el software de gerencia de información, mientras que es generalmente lo contrario para los modelos de datos.
- La maestría del ingeniero del software. Es a menudo más productivo adoptar una notación que modele o método con el cual el ingeniero del software tiene una experiencia.
- Los requisitos de proceso del cliente. Los clientes pueden imponer su notación o método favorecido, o prohibir cualquiera que le sea desconocido. Este factor puede estar en conflicto con el factor anterior.
- La disponibilidad de métodos y de herramientas. Notaciones o métodos que son mal apoyados por el entrenamiento y las herramientas pueden no alcanzar la aceptación esperada aunque se satisfagan tipos particulares de problemas.

Observar que, en casi todos los casos, es útil comenzar construyendo un modelo del contexto del software. El contexto del software proporciona una conexión entre el software previsto y su ambiente externo. Esto es crucial para entender el contexto del software en su ambiente operacional e identificar sus interfaces con el ambiente.

La aplicación de modelar se junta firmemente con la de los métodos. Para los propósitos prácticos, un método es una notación (o sistema de notaciones) apoyado por un proceso que dirige el uso de las notaciones. Hay escasa evidencia empírica para apoyar las demandas de superioridad de una notación sobre otra. Sin embargo, la extensa aceptación de un método o de una notación particular puede conducir a nivel industrial al beneficio de habilidades y de conocimiento. Ésta es actualmente la situación con el UML (Lenguaje de Modelado Unificado). (UML04)

El modelado formal usando notaciones basadas en matemática discreta, y que son detectables al razonamiento lógico, han tenido impacto en algunos dominios especializados. Éstos puede ser impuesto por los clientes o los estándares y puede ofrecer ventajas que obligan al análisis de ciertas funciones o componentes críticos.

Este asunto no busca “enseñar” un estilo o una notación de modelado particular sino proporcionar algo a la dirección con el propósito de modelar.

Dos estándares proporcionan las notaciones que pueden ser útiles en desarrollo conceptual realizando modelado-IEEE conceptual Std 1320.1, IDEF0 para modelado funcional; e IEEE Std 1320.2, IDEF1X97 (IDEFObject) para modelado de la información.

4.3. Asignación arquitectónica del diseño y de los requisitos
[Dav93; Som05]

En un cierto punto, la arquitectura de la solución debe ser derivada. El diseño arquitectónico es el punto en el cual el proceso de los requisitos se junta con software o diseño de sistemas e ilustra cómo de imposible es desemparejar ambas tareas. [Som01] Este asunto está de cerca relacionado con el capítulo de la estructura y de la arquitectura del software en KA del diseño del software. En muchos casos, el ingeniero del software actúa como arquitecto del software porque el proceso de analizar y de elaborar los requisitos exige que los componentes que serán responsables de satisfacer los requisitos estén identificados. Esto es localización de requisitos-la asignación, a los componentes, de la responsabilidad de satisfacer requisitos.

La asignación es importante para permitir análisis detallado de requisitos. Por lo tanto, por ejemplo, una vez un sistema de requisitos se han asignado a un componente, los requisitos individuales se pueden analizar más a fondo para descubrir otros requisitos de cómo el componente necesita obrar recíprocamente con otros componentes para satisfacer los requisitos asignados. En proyectos grandes, la asignación estimula un nuevo análisis para cada subsistema. Como ejemplo, requisitos para el funcionamiento de los frenos de un coche (distancia que frena, seguridad en condiciones que conducen, suavidad del uso, la presión del pedal requerida, y así sucesivamente) se puede asignar un hardware que frena (montajes mecánicos e hidráulicos) y un sistema de frenos antibloqueo (ABS). Solamente cuando un requisito para un sistema de frenos antibloqueo ha sido identificado, y los requisitos asignados a él, pueden usarse las capacidades del ABS, el hardware de frenado identificado, y las características indefinidas (tales como el peso del coche) para identificar los requisitos detallados del software del ABS.

El diseño arquitectónico se identifica de cerca con el modelado conceptual. El mapeado de las entidades del dominio del mundo real para componentes de software no siempre tiene un diseño obvio, así que arquitectónicamente se identifica como a asunto separado. Los requisitos de notaciones y los métodos son ampliamente iguales para modelado conceptual y diseño arquitectónico.

IEEE Std 1471-2000, práctica recomendada para la descripción arquitectónica de sistemas orientados al software, sugiere un acercamiento del múltiple punto de vista para describir la arquitectura de sistemas y de sus artículos del software.
(IEEE1471-00)

4.4. Negociación de los requisitos

Otro término comúnmente utilizado para este tema es "resolución del conflicto." Esto se refiere a problemas de resolución de los requisitos donde los conflictos ocurren entre dos stakeholders que requieren características mutuamente incompatibles, entre los requisitos y los recursos, o en entre requisitos funcionales y no funcionales, por ejemplo.

[Kot00, Som97] en la mayoría de los casos, es imprudente para el ingeniero del software tomar una decisión unilateral, y hace necesario consultar con los stakeholders para alcanzar un consenso en una compensación apropiada. Es a menudo importante por esas razones contractuales que tales decisiones sean detectables de nuevo al cliente. Hemos clasificado esto como asunto del análisis de requisitos del software porque los problemas emergen como resultado el análisis. Sin embargo, un caso fuerte se puede también hacer para considerar los requisitos como asunto de la validación.

5. Especificación de requisitos

Para la mayoría de las profesiones de la ingeniería, el término "especificación" se refiere a la asignación de valores o límites numéricos para metas del diseño del producto. (Vin90) Los sistemas físicos típicos tienen un número relativamente pequeño de tales valores. Típicamente el software tiene una gran cantidad de requisitos, y el énfasis se comparte entre la ejecución de la cuantificación numérica y el manejo de la complejidad de la interacción entre el gran número de requisitos. Así pues, en software el término, "especificación de requisitos del software se refiere típicamente a la producción de un documento, o a su equivalente electrónico, que puede estar sistemáticamente repasado, evaluado, y aprobado. Para los sistemas complejos, particularmente ésos que implican componentes no-software, se elaboran tres tipos de documentos: definición de sistema, sistema requisitos, y requisitos del software. Para sistemas simples, solamente el tercero de éstos es requerido. Los tres documentos se describen aquí, entendiendo que combinados pueden ser apropiados. Una descripción de la ingeniería de sistemas se puede encontrar en el Capítulo 12, disciplinas relacionadas de la tecnología de dotación lógica.

5.1. El documento de la definición de sistema

Este documento (conocido a veces como documento de exigencias del o concepto de operaciones) registra el sistema requisitos. Define los requisitos del sistema de alto nivel desde la perspectiva del dominio. Su número total de lectores incluye los representantes de los usuarios del sistema/de los clientes (la comercialización puede desempeñar estos papeles del mercado software), así que su contenido debe estar en términos de dominio. El documento enumera los requisitos del sistema junto con información de fondo sobre los objetivos totales para el sistema, su ambiente de misión y una declaración de apremios, asunciones, y requisitos no funcionales. Puede incluir los modelos conceptuales diseñados para ilustrar el contexto del sistema, panoramas del uso y las entidades principales del dominio, así como datos, la información, y flujos de trabajo. IEEE Std 1362, concepto del documento de las operaciones, proporciona consejo sobre la preparación y contenido de tal documento.
(IEEE1362-98)

5.2. Especificación de requisitos del sistema
[Dav93; Kot00; Rob99; Tha97]

Desarrolladores de sistemas con los componentes software y no software, un avión de pasajeros moderno, por ejemplo, separa a menudo la descripción de los requisitos del sistema de la descripción de los requisitos del software. En esto se especifica la visión, requisitos del sistema, los requisitos software se derivan de los requisitos del sistema, y entonces los requisitos para los componentes de software se especifican. En sentido estricto, la especificación de requisitos del sistema es una actividad de la ingeniería de sistemas y esta fuera del alcance de esta guía. IEEE Std 1233 es una guía para los requisitos del sistema que se convierten. (IEEE1233-98)

5.3. Especificación de requisitos del software
[Kot00; Rob99]

La especificación de requisitos del software establece la base para el acuerdo entre los clientes y los contratistas o los proveedores (en proyectos del mercado, estos papeles se pueden desempeñar por las divisiones de comercialización y desarrollo) en los que hay que hacer el producto de software, así como lo que no se espera que haga. Para los lectores no técnicos, el documento de la especificación de los requisitos es acompañado a menudo por un documento de la definición de los requisitos del software.

La especificación de requisitos del software permite un riguroso gravamen de requisitos antes de que el diseño pueda comenzar y reducir un reajuste final. Debe también proporcionar una base realista para estimar costes, riesgos, y horario del producto.

Las organizaciones pueden también utilizar un documento de especificación de requisitos software para desarrollar su propia validación y que la verificación sea más productiva.

La especificación de requisitos software proporciona una base informada para transferir un producto de software a los nuevos usuarios o a las máquinas nuevas. Finalmente, puede proporcionar una base para el realce de software.

Los requisitos del software se escriben a menudo en lenguaje natural, pero, en la especificación de requisitos del software, ésta se puede suplir por formal o semi-

formal. La selección de notaciones apropiadas permite requisitos y los aspectos particulares de la arquitectura del software que se describirá más ajustadamente que en lengua natural. La regla general es que las notaciones deben ser utilizadas para que permitan a los requisitos ser descritos tan exactamente como sea posible. Esto es particularmente crucial para otros tipos seguridad-crítica y casos de software confiable. Sin embargo, la opción de la notación es obligada a menudo por el entrenamiento, las habilidades y las preferencias de los autores y de los lectores del documento.

Se han desarrollado un número de indicadores de la calidad para relacionar la calidad de la especificación de requisitos del software a otras variables del proyecto por ejemplo coste, aceptación, funcionamiento, horario, reproducibilidad, indicadores de la calidad etc. para el individuo las declaraciones de la especificación de requisitos del software incluyen imperativos, directorios, frases débiles, opciones, y continuaciones. Indicadores para el documento de especificación de requisitos software incluye tamaño, legibilidad, especificación, profundidad, y estructura del texto.
[Dav93; Tha97] (Ros98)

IEEE tiene un estándar, IEEE Std 830 [IEEE830-98], para producción y contenido de la especificación de los requisitos del software. También, IEEE 1465 (similar a ISO/IEC 12119) es un estándar que trata requisitos de calidad en paquetes de software. (IEEE1465-98)

6. Validación de los requisitos
[Dav93]

Los documentos de los requisitos pueden estar conformes a la validación y procedimientos de verificación. Los requisitos pueden ser validados para asegurarse de que el ingeniero del software entiende los requisitos, y es también importante para verificar que un documento de requisitos se conforma con la compañía de los estándares, y éste es comprensible, constante, y finito. Las notaciones formales ofrecen la ventaja importante de permitir que las dos características pasadas sean probadas (en un sentido estricto, por lo menos). Diversos stakeholders, incluyendo los representantes del cliente y del revelador, deben también repasar los documentos. Los documentos de los requisitos son conformes a las mismas prácticas de gerencia de la configuración del software como los otros puntos relevantes de los procesos del ciclo de vida del software. (Bry94, Ros98)

Es normal programar explícitamente unos o más puntos en el proceso de los requisitos donde están los requisitos validados. La puntería es tomar cualquier problema antes de que los recursos se destinen a tratar los requisitos. La validación de los requisitos se refiere al proceso de examinar el documento de los requisitos para asegurarse de que este define el software correctamente (es decir, el software que los usuarios esperan). [Kot00]

6.1 Revisiones de los requisitos
[Kot00; Som05; Tha97]

Quizás los medios más comunes de la validación están cerca de inspección o revisiones de los documentos de los requisitos. Asignan un grupo de revisores en un escrito para buscar errores, asunciones confundidas, la carencia de la claridad, y la desviación de la costumbre. La composición del grupo que conduce la revisión es importante (por lo menos un representante del cliente debe ser incluido para un proyecto cliente-conducido, por ejemplo), y puede ayudar a proporcionar la dirección en qué buscar bajo listas de comprobación.

Las revisiones se pueden constituir en el final del documento de definición del sistema, el documento de la especificación de sistema, el documento de la especificación de requisitos del software, especificación de la línea de fondo para un nuevo lanzamiento, o en cualquier otro paso en el proceso. IEEE Std 1028 proporciona la dirección para conducir tales revisiones. (IEEE1028-97) Las revisiones son también cubiertas en KA de la calidad del software, punto 2.3 Revisiones e intervenciones.

6.2. Prototipado
[Dav93; Kot00; Som05; Tha97]

Prototipar es comúnmente el medio para validar la interpretación del ingeniero del software de los requisitos del software, así como para sacar nuevos requisitos. Hay una gama de técnicas de prototipado y un número de puntos en el proceso donde la validación del prototipo puede ser apropiada. La ventaja de usar prototipos es que pueden hacer más fácil la interpretación de las asunciones del ingeniero del software y, donde lo necesite, dan la explicación útil de porqué son incorrectas. Por ejemplo, el comportamiento dinámico de un interfaz utilizador se puede entender mejor a través de un prototipo animado que a través de la descripción textual o de modelos gráficos. Hay también desventajas, sin embargo. Éstos incluyen el peligro de la atención de los usuarios que es distraída de la base de la funcionalidad subyacente por las ediciones o los problemas cosméticos de la calidad con el prototipo. Por esta razón, algunas personas dicen que los prototipos que evitan. Los prototipos pueden ser costosos. Sin embargo, si evitan el despilfarro de los recursos causados intentando satisfacer requisitos erróneos, su coste puede ser más fácilmente justificado.

6.3. Validación modelo
[Dav93; Kot00; Tha97]

Es típicamente necesario validar la calidad de los modelos desarrollados durante el análisis. Por ejemplo, en modelos de objeto, es útil para realizar un análisis

estático para verificar que las trayectorias de comunicación existen entre los objetos que, en dominio de los stakeholders, intercambian datos. Si las notaciones de especificación formal se utilizan, es posible utilizar el razonamiento formal para probar características de la especificación.

6.4. Pruebas de aceptación

[Dav93]

Una característica esencial de un requisito del software es que debe ser posible validar que el producto final lo satisface. Los requisitos que no pueden ser validados son "deseos realmente justos." Una tarea importante es por lo tanto planear cómo verificar cada requisito. En la mayoría de los casos, el diseño de pruebas de aceptación hace esto. Identificar y diseñar pruebas de aceptación pueden ser difícil para los requisitos no funcionales (véase el asunto los requisitos funcionales y no funcionales de 1.3). Para ser validados, deben primero ser analizados al punto donde pueden ser expresados cuantitativamente.
La información adicional se puede encontrar en el software KA de prueba, conformidad de la prueba en el punto 2.2.4.

7. Consideraciones prácticas

El primer nivel de la descomposición de los puntos presentados en este KA puede parecer que describe una secuencia lineal de actividades. Ésta es una vista simplificada del proceso.
[Dav93]

Los requisitos procesan palmos de todo el ciclo de vida del software. Cambiar la gerencia y el mantenimiento de los requisitos en un estado que refleja exactamente el software que se construirá, o se ha construido, es clave para el éxito del proceso ingeniería del software
[Kot00; Lou95]

No toda organización tiene una cultura de documentación y manejo de requisitos. Es frecuente en las compañías de start-up dinámicas, conducidas por una "visión fuerte del producto" y recursos limitados, para ver la documentación de los requisitos como gastos indirectos innecesarios. Más a menudo, sin embargo, según estas compañías crecen, mientras que su base de cliente crece, y como su producto comienza a desarrollarse, estas descubren que necesitan recuperar los requisitos que las características de producto motivadas para determinar el impacto de cambios propuestos. Por lo tanto, la documentación de los requisitos y la gerencia del cambio son llave al éxito de cualquier proceso de los requisitos.

7.1. Naturaleza iterativa del proceso de los requisitos

[Kot00; You01]

Hay presión general en la industria del software para ciclos de desarrollo más cortos, y esto está particularmente pronunciado en sectores del mercado altamente competitivos. Por otra parte, la mayoría de los proyectos son obligados de cierta manera por su ambiente, y muchas son mejoras, o revisiones, del software existente donde está la arquitectura dada. En la práctica, por lo tanto, es casi siempre impráctico poner el proceso de los requisitos en ejecución como proceso lineal, determinista en el cual los requisitos del software se saquen de los stakeholders, asignado, y entregado al equipo de desarrollo del software. Es ciertamente un mito que los requisitos para los proyectos grandes del software siempre están entendidos perfectamente o especificados perfectamente. [Som97]

En su lugar, los requisitos iteran típicamente hacia un nivel de calidad y detallan que es suficiente permitir las decisiones del diseño y de la consecución que se harán. En algunos proyectos, esto puede dar lugar a requisitos antes de que todas sus características se entiendan completamente. Esto arriesga a una reanudación costosa si los problemas emergen tarde en el proceso de la ingeniería del software. Sin embargo, los ingenieros del software son obligados necesariamente por planes de la gerencia de proyecto y deben por lo tanto tomar medidas para asegurarse de que la "calidad" de los requisitos es tan alta como sea posible dado los recursos disponibles. Deben, por ejemplo, hacer explícita cualquier asunción que sostengan los requisitos, así como cualesquiera problemas sabidos.

En casi todos los casos, el entendimiento de los requisitos continúa desarrollándose mientras que procede el diseño y el desarrollo. Esto conduce a menudo a la revisión de requisitos tarde en el ciclo de vida. Quizás el punto más crucial de entender la ingeniería de requisitos es que una proporción significativa de los requisitos cambiará. Esto es a veces debido a los errores en el análisis, pero es con frecuencia una consecuencia inevitable del cambio en el "ambiente": por ejemplo, la funcionalidad del cliente o el ambiente del negocio, o el mercado en el cual software se va a vender. Cualquiera que sea la causa, es importante reconocer la inevitabilidad del cambio para atenuar sus efectos. El cambio tiene que ser manejado asegurando esos cambios propuestos mediante una revisión definida y un análisis del proceso de aprobación, y, aplicando los requisitos cuidadosos que remontan, del impacto, y la configuración del software (véase la configuración del software KA de la gerencia). Por lo tanto, el proceso de los requisitos no es simplemente una tarea anticipada en el desarrollo del software, pero atraviesa por completo el ciclo de vida del software. En un proyecto típico, las actividades de los requisitos del software se desarrollan en un cierto.

7.2. Cambiar a gestión

[Kot00]

La gestión del cambio es central en el análisis de requisitos. Este asunto describe el papel de la gestión del cambio, los procedimientos que necesitan estar preparados, y el análisis que se debe aplicar a los cambios propuestos. Tiene acoplamientos fuertes a la configuración del software KA de la gestión.

7.3. Cualidades de los requisitos [Kot00]

Los requisitos deben consistir no sólo una especificación de qué se requiere, sino también de la información ancilar que las ayudas manejan e interpretan los requisitos. Esto debe incluir varias dimensiones de la clasificación del requisito (véase la clasificación de los requisitos del asunto 4.1) y el método de la verificación o el plan de prueba de aceptación. Puede también incluir la información adicional tal como un resumen-análisis razonado para cada requisito, la fuente de cada requisito, y una historia del cambio. Los requisitos más importantes atribuyen, sin embargo, un identificador que permite requisitos identificados inequívocamente.

7.4 El remontar de los requisitos [Kot00]

El remontar de los requisitos se refiere a la fuente recuperación de requisitos y de predecir los efectos de esos requisitos. El trazado es fundamental para el análisis de la ejecución del impacto cuando los requisitos cambian. Un requisito debe ser detectable al revés a requisitos y los stakeholders que lo motivaron (de un requisito del software de nuevo a los requisitos del sistema que ayudan a satisfacer, por ejemplo). Inversamente, un requisito debe ser detectable entidades del diseño que lo satisfacen (por ejemplo, de un requisito del sistema en el software requisitos que se han elaborado a partir de él, y encendido en los módulos del código que lo ponen en ejecución).

Los requisitos que remontan para un proyecto típico formarán un gráfico acíclico dirigido complejo (DAG) de requisitos.

7.5 Requisitos que miden

Como cuestión práctica, es típicamente útil tener cierto concepto del "volumen" de los requisitos para un producto de software particular. Este punto es útil evaluando el "tamaño" de un cambio en requisitos, en estimar el coste de una tarea del desarrollo o del mantenimiento, o simplemente para el uso como el denominador en otra medida. La medida funcional del tamaño (FSM) es una técnica para evaluar el tamaño de un cuerpo de requisitos funcionales. IEEE Std 14143.1 define el concepto de FSM. [IEEE14143.1-00] Estándares de ISO/IEC y otras fuentes describen métodos particulares del FSM.

Información adicional sobre la medida del tamaño y los estándares se encuentran en el proceso de la ingeniería del software.

MATRIX OF TOPICS VS. REFERENCE MATERIAL

	[Dav93]	[Gog93]	[IEEE830-98]	[IEEE1413.1-00]	[Kot00]	[Lou95]	[Pfl01]	[Rob99]	[Som97]	[Som05]	[Tha97]	[You01]
1. Software Requirements Fundamentals												
1.1 Definition of a Software Requirement					*		*			c5	c1	
1.2 Product and Process Requirements					*				c1			
1.3 Functional and Non-functional Requirements					*				c1			
1.4 Emergent Properties										c2		
1.5 Quantifiable Requirements	c3s4									c6		
1.6 System Requirements and Software Requirements												
2. Requirements Process	*									c5		
2.1 Process Models					c2s1			*	c2	c3		
2.2 Process Actors	c2	*			c2s2			c3	c2			c3
2.3 Process Support and Management								c3	c2			c2,c7
2.4 Process Quality and Improvement					c2s4				c2			c5
3. Requirements Elicitation	*	*				*	*					
3.1 Requirements Sources	c2	*			c3s1	*	*				c1	
3.2 Elicitation Techniques	c2	*			c3s2	*	*					
4. Requirements Analysis	*									c6		
4.1 Requirements Classification	*				c8s1					c6		
4.2 Conceptual Modeling	*				*					c7		
4.3 Architectural Design and Requirements Allocation	*									c10		
4.4 Requirements Negotiation					c3s4				*			
5. Requirements Specification												
5.1 The System Definition Document												
5.2 The System Requirements Specification	*				*			c9			c3	
5.3 The Software Requirements Specification	*		*		*			c9			c3	
6. Requirements Validation	*				*							
6.1 Requirements Reviews					c4s1					c6	c5	
6.2 Prototyping	c6				c4s2					c8	c6	
6.3 Model Validation	*				c4s3						c5	
6.4 Acceptance Tests	*											
7. Practical Considerations	*				*	*						
7.1 Iterative Nature of the Requirements Process					c5s1				c2			c6
7.2 Change Management					c5s3							
7.3 Requirement Attributes					c5s2							
7.4 Requirements Tracing					c5s4							
7.5 Measuring Requirements				*								

REFERENCIAS RECOMENDADAS PARA REQUSITOS SW

[Dav93] A.M. Davis, *Software Requirements: Objects, Functions and States*, Prentice Hall, 1993.
[Gog93] J. Goguen and C. Linde, "Techniques for Requirements Elicitation," presented at International Symposium on Requirements Engineering, 1993.
[IEEE830-98] IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications*, IEEE, 1998.
(IEEE14143.1-00) IEEE Std 14143.1-2000//ISO/ IEC14143-1:1998, *Information Technology—Software Measurement—Functional Size Measurement—Part 1: Definitions of Concepts*, IEEE, 2000.
[Kot00] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*, John Wiley & Sons, 2000.
[Lou95] P. Loucopulos and V. Karakostas, *Systems Requirements Engineering*, McGraw-Hill, 1995. [Pfl01] S.L. Pfleeger, "Software Engineering: Theory and Practice," second ed., Prentice Hall, 2001, Chap. 4.
[Rob99] S. Robertson and J. Robertson, *Mastering the Requirements Process*, Addison-Wesley, 1999.
[Som97] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, 1997, Chap. 1-2.
[Som05] I. Sommerville, *Software Engineering*, seventhed., Addison-Wesley, 2005.
[Tha97] R.H. Thayer and M. Dorfman, eds., *Software Requirements Engineering*, IEEE Computer Society Press, 1997, pp. 176-205, 389-404.
[You01] R.R. You, *Effective Requirements Practices*, Addison-Wesley, 2001.

APÉNDICE A. LISTA DE LECTURAS COMPLEMENTARIAS

(Ale02) I. Alexander and R. Stevens, *Writing Better Requirements*, Addison-Wesley, 2002.

(Ard97) M. Ardis, "Formal Methods for Telecommunication System Requirements: A Survey of Standardized Languages," *Annals of Software Engineering*, vol. 3, 1997.

(Ber97) V. Berzins et al., "A Requirements Evolution Model for Computer Aided Prototyping," presented at Ninth IEEE International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 1997.

(Bey95) H. Beyer and K. Holtzblatt, "Apprenticing with the Customer," *Communications of the ACM*, vol. 38, iss. 5, May 1995, pp. 45-52.

(Bru95) G. Bruno and R. Agarwal, "Validating Software Requirements Using Operational Models," presented at Second Symposium on Software Quality Techniques and Acquisition Criteria, 1995.

(Bry94) E. Bryne, "IEEE Standard 830: Recommended Practice for Software Requirements Specification," presented at IEEE International Conference on Requirements Engineering, 1994.

(Buc94) G. Bucci et al., "An Object-Oriented Dual Language for Specifying Reactive Systems," presented at IEEE International Conference on Requirements Engineering, 1994.

(Bus95) D. Bustard and P. Lundy, "Enhancing Soft Systems Analysis with Formal Modeling," presented at Second International Symposium on Requirements Engineering, 1995.

(Che94) M. Chechik and J. Gannon, "Automated Verification of Requirements Implementation," presented at Proceedings of the International Symposium on Software Testing and Analysis, special issue, 1994.

(Chu95) L. Chung and B. Nixon, "Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach," presented at Seventeenth IEEE International Conference on Software Engineering, 1995.

(Cia97) P. Ciancarini et al., "Engineering Formal Requirements: An Analysis and Testing Method for Z Documents," *Annals of Software Engineering*, vol. 3, 1997.

(Cre94) R. Crespo, "We Need to Identify the Requirements of the Statements of Non-Functional Requirements," presented at International Workshop on Requirements Engineering: Foundations of Software Quality, 1994.

(Cur94) P. Curran et al., "BORIS-R Specification of the Requirements of a Large-Scale Software Intensive System," presented at Requirements Elicitation for Software-Based Systems, 1994.

(Dar97) R. Darimont and J. Souquieres, "Reusing Operational Requirements: A Process-Oriented Approach," presented at IEEE International Symposium on Requirements Engineering, 1997.

(Dav94) A. Davis and P. Hsia, "Giving Voice to Requirements Engineering: Guest Editors' Introduction," *IEEE Software*, vol. 11, iss. 2, March 1994, pp. 12-16.

(Def94) J. DeFoe, "Requirements Engineering Technology in Industrial Education," presented at IEEE International Conference on Requirements Engineering,1994.

(Dem97) E. Demirors, "A Blackboard Framework for Supporting Teams in Software Development," presented at Ninth IEEE International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 1997.

(Die95) M. Diepstraten, "Command and Control System Requirements Analysis and System Requirements Specification for a Tactical System," presented at First IEEE International Conference on Engineering of complex Computer Systems, 1995.

(Dob94) J. Dobson and R. Strens, "Organizational Requirements Definition for Information Technology," presented at IEEE International Conference on Requirements Engineering, 1994.

(Duf95) D. Duffy et al., "A Framework for Requirements Analysis Using Automated Reasoning," presented at Seventh International Conference on Advanced Information Systems Engineering, 1995.

(Eas95) S. Easterbrook and B. Nuseibeh, "Managing Inconsistencies in an Evolving Specification," presented at Second International Symposium on Requirements Engineering, 1995.

(Edw95) M. Edwards et al., "RECAP: A Requirements Elicitation, Capture, and Analysis Process Prototype Tool for Large Complex Systems," presented at First IEEE International Conference on Engineering of Complex Computer Systems, 1995.

(ElE95) K. El-Emam and N. Madhavji, "Requirements Engineering Practices in Information Systems Development: A Multiple Case Study," presented at Second International Symposium on Requirements Engineering, 1995.

(Fai97) R. Fairley and R. Thayer, "The Concept of Operations: The Bridge from Operational Requirements to Technical Specifications," *Annals of Software Engineering*, vol. 3, 1997.

(Fic95) S. Fickas and M. Feather, "Requirements Monitoring in Dynamic Environments," presented at Second International Symposium on Requirements Engineering, 1995.

(Fie95) R. Fields et al., "A Task-Centered Approach to Analyzing Human Error Tolerance Requirements," resented at Second International Symposium on Requirements Engineering, 1995.

(Gha94) J. Ghajar-Dowlatshahi and A. Varnekar, "Rapid Prototyping in Requirements Specification Phase of Software Systems," presented at Fourth International Symposium on Systems Engineering, National Council on Systems Engineering, 1994.

(Gib95) M. Gibson, "Domain Knowledge Reuse During Requirements Engineering," presented at Seventh International Conference on Advanced Information Systems Engineering (CAiSE '95), 1995.

(Gol94) L. Goldin and D. Berry, "AbstFinder: A Prototype Abstraction Finder for Natural Language Text for Use in Requirements Elicitation: Design, Methodology and Evaluation," presented at IEEE International Conference on Requirements Engineering, 1994.

(Got97) O. Gotel and A. Finkelstein, "Extending Requirements Traceability: Lessons Learned from an Industrial Case Study," presented at IEEE International Symposium on Requirements Engineering, 1997.
(Hei96) M. Heimdahl, "Errors Introduced during the TACS II Requirements Specification Effort: A Retrospective Case Study," presented at Eighteenth IEEE International Conference on Software Engineering, 1996.
(Hei96a) C. Heitmeyer et al., "Automated Consistency Checking Requirements Specifications," *ACM Transactions on Software Engineering and Methodology*, vol. 5, iss. 3, July 1996, pp. 231-261.
(Hol95) K. Holtzblatt and H. Beyer, "Requirements Gathering: The Human Factor," *Communications of the ACM*, vol. 38, iss. 5, May 1995, pp. 31-32.
(Hud96) E. Hudlicka, "Requirements Elicitation with Indirect Knowledge Elicitation Techniques: Comparison of Three Methods," presented at Second IEEE International Conference on Requirements Engineering, 1996.
(Hug94) K. Hughes et al., "A Taxonomy for Requirements Analysis Techniques," presented at IEEE International Conference on Requirements Engineering, 1994.
(Hug95) J. Hughes et al., "Presenting Ethnography in the Requirements Process," presented at Second IEEE International Symposium on Requirements Engineering, 1995.
(Hut94) A.T.F. Hutt, ed., *Object Analysis and Design - Comparison of Methods. Object Analysis and Design - Description of Methods*, John Wiley & Sons, 1994.(INCOSE00) INCOSE, *How To: Guide for all Engineers, Version 2*, International Council on Systems Engineering, 2000.
(Jac95) M. Jackson, *Software Requirements and Specifications*, Addison-Wesley, 1995.
(Jac97) M. Jackson, "The Meaning of Requirements," *Annals of Software Engineering*, vol. 3, 1997.
(Jon96) S. Jones and C. Britton, "Early Elicitation and Definition of Requirements for an Interactive Multimedia Information System," presented at Second IEEE International Conference on Requirements Engineering, 1996.
(Kir96) T. Kirner and A. Davis, "Nonfunctional Requirements for Real-Time Systems," *Advances in Computers*, 1996.
(Kle97) M. Klein, "Handling Exceptions in Collaborative Requirements Acquisition," presented at IEEE International Symposium on Requirements Engineering,
1997.
(Kos97) R. Kosman, "A Two-Step Methodology to Reduce Requirements Defects," *Annals of Software Engineering*, vol. 3, 1997.
(Kro95) J. Krogstie et al., "Towards a Deeper Understanding of Quality in Requirements Engineering," presented at Seventh International Conference on Advanced Information Systems Engineering (CAiSE '95), 1995.
(Lal95) V. Lalioti and B. Theodoulidis, "Visual Scenarios for Validation of Requirements Specification," presented at Seventh International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 1995.
(Lam95) A. v. Lamsweerde et al., "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt," presented at Second International Symposium on Requirements Engineering, 1995.
(Lei97) J. Leite et al., "Enhancing a Requirements Baseline with Scenarios," presented at IEEE International Symposium on Requirements Engineering, 1997.
(Ler97) F. Lerch et al.., "Using Simulation-Based Experiments for Software Requirements Engineering," *Annals of Software Engineering*, vol. 3, 1997.
(Lev94) N. Leveson et al., "Requirements Specification or Process-Control Systems," *IEEE Transactions on Software Engineering*, vol. 20, iss. 9, September 1994, pp. 684-707.
(Lut96a) R. Lutz and R. Woodhouse, "Contributions of SFMEA to Requirements Analysis," presented at Second IEEE International Conference on Requirements Engineering, 1996.
(Lut97) R. Lutz and R. Woodhouse, "Requirements Analysis Using Forward and Backward Search," *Annals of Software Engineering*, vol. 3, 1997.
(Mac96) L. Macaulay, *Requirements Engineering*, Springer-Verlag, 1996.
(Mai95) N. Maiden et al., "Computational Mechanisms for Distributed Requirements Engineering," presented at Seventh International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 1995.
(Mar94) B. Mar, "Requirements for Development of Software Requirements," presented at Fourth International Symposium on Systems Engineering, 1994.
(Mas97) P. Massonet and A. v. Lamsweerde, "Analogical Reuse of Requirements Frameworks," presented at IEEE International Symposium on Requirements Engineering, 1997.
(McF95) I. McFarland and I. Reilly, "Requirements Traceability in an Integrated Development Environment," presented at Second International Symposium on Requirements Engineering, 1995.
(Mea94) N. Mead, "The Role of Software Architecture in Requirements Engineering," presented at IEEE International Conference on Requirements Engineering, 1994.
(Mos95) D. Mostert and S. v. Solms, "A Technique to Include Computer Security, Safety, and Resilience Requirements as Part of the Requirements Specification," *Journal of Systems and Software*, vol. 31, iss. 1, October 1995, pp. 45-53.
(Myl95) J. Mylopoulos et al., "Multiple Viewpoints Analysis of Software Specification Process," *IEEE Transactions on Software Engineering*, 1995.
(Nis92) K. Nishimura and S. Honiden, "Representing and Using Non-Functional Requirements: A Process-Oriented Approach," *IEEE Transactions on Software Engineering*, December 1992.
(Nis97) H. Nissen et al., "View-Directed Requirements Engineering: A Framework and Metamodel," presented at Ninth IEEE International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 1997.
(OBr96) L. O'Brien, "From Use Case to Database: Implementing a Requirements Tracking System," *Software Development*, vol. 4, iss. 2, February 1996, pp. 43-47.
(UML04) Object Management Group, *Unified Modeling Language*, www.uml.org, 2004. (Opd94) A. Opdahl, "Requirements Engineering for Software Performance,"

presented at International Workshop on Requirements Engineering: Foundations of Software Quality, 1994.
(Pin96) F. Pinheiro and J. Goguen, "An Object-Oriented Tool for Tracing Requirements," *IEEE Software*, vol. 13, iss. 2, March 1996, pp. 52-64.
(Pla96) G. Playle and C. Schroeder, "Software Requirements Elicitation: Problems, Tools, and Techniques," *Crosstalk: The Journal of Defense Software Engineering*, vol. 9, iss. 12, December 1996, pp. 19-24.
(Poh94) K. Pohl et al., "Applying AI Techniques to Requirements Engineering: The NATURE Prototype," presented at IEEE Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence, 1994.
(Por95) A. Porter et al., "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment," *IEEE Transactions on Software Engineering*, vol. 21, iss. 6, June 1995, pp. 563-575.
(Pot95) C. Potts et al., "An Evaluation of Inquiry-Based Requirements Analysis for an Internet Server," presented at Second International Symposium on Requirements Engineering, 1995.
(Pot97) C. Potts and I. Hsi, "Abstraction and Context in Requirements Engineering: Toward a Synthesis," *Annals of Software Engineering*, vol. 3, 1997.
(Pot97a) C. Potts and W. Newstetter, "Naturalistic Inquiry and Requirements Engineering: Reconciling Their Theoretical Foundations," presented at IEEE International Symposium on Requirements Engineering, 1997.
(Ram95) B. Ramesh et al., "Implementing Requirements Traceability: A Case Study," presented at Second International Symposium on Requirements Engineering, 1995.
(Reg95) B. Regnell et al., "Improving the Use Case Driven Approach to Requirements Engineering," presented at Second IEEE International Symposium on Requirements Engineering, 1995.
(Reu94) H. Reubenstein, "The Role of Software Architecture in Software Requirements Engineering," presented at IEEE International Conference on Requirements Engineering, 1994.
(Rob94) J. Robertson and S. Robertson, *Complete Systems Analysis*, Vol. 1 and 2, Prentice Hall, 1994.
(Rob94a) W. Robinson and S. Fickas, "Supporting Multi-Perspective Requirements Engineering," presented at IEEE International Conference on Requirements Engineering, 1994.
(Ros98) L. Rosenberg, T.F. Hammer, and L.L. Huffman, "Requirements, testing and metrics," presented at 15th Annual Pacific Northwest Software Quality Conference, 1998.
(Sch94) W. Schoening, "The Next Big Step in Systems Engineering Tools: Integrating Automated Requirements Tools with Computer Simulated Synthesis and Test," presented at Fourth International Symposium on Systems Engineering, 1994.
(She94) M. Shekaran, "The Role of Software Architecture in Requirements Engineering," presented at IEEE International Conference on Requirements Engineering, 1994.
(Sid97) J. Siddiqi et al., "Towards Quality Requirements Via Animated Formal Specifications," *Annals of Software Engineering*, vol. 3, 1997.
(Span97) G. Spanoudakis and A. Finkelstein, "Reconciling Requirements: A Method for Managing Interference, Inconsistency, and Conflict," *Annals of Software Engineering*, vol. 3, 1997.
(Ste94) R. Stevens, "Structured Requirements," presented at Fourth International Symposium on Systems Engineering, 1994.
(Vin90) W.G. Vincenti, *What Engineers Know and How They Know It - Analytical Studies form Aeronautical History*, John Hopkins University Press, 1990.
(Wei03) K. Weigers, *Software Requirements*, second ed., Microsoft Press, 2003.
(Whi95) S. White and M. Edwards, "A Requirements Taxonomy for Specifying Complex Systems," presented at First IEEE International Conference on Engineering of Complex Computer Systems, 1995.
(Wil99) B. Wiley, *Essential System Requirements: A Practical Guide to Event-Driven Methods*, Addison-Wesley, 1999.
(Wyd96) T. Wyder, "Capturing Requirements With Use Cases," *Software Development*, vol. 4, iss. 2, February 1996, pp. 36-40.
(Yen97) J. Yen and W. Tiao, "A Systematic Tradeoff Analysis for Conflicting Imprecise Requirements," presented at IEEE International Symposium on Requirements Engineering, 1997.
(Yu97) E. Yu, "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering," presented at IEEE International Symposium on Requirements Engineering, 1997.
(Zav96) P. Zave and M. Jackson, "Where Do Operations Come From? A Multiparadigm Specification Technique," *IEEE Transactions on Software Engineering,*, vol. 22, iss. 7, July 1996, pp. 508-528.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE830-98) IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications*, IEEE, 1998.

(IEEE1028-97) IEEE Std 1028-1997 (R2002), *IEEE Standard for Software Reviews*, IEEE, 1997.

(IEEE1233-98) IEEE Std 1233-1998, *IEEE Guide for Developing System Requirements Specifications*, 1998.

(IEEE1320.1-98) IEEE Std 1320.1-1998, *IEEE Standard for Functional Modeling Language-Syntax and Semantics for IDEF0*, IEEE, 1998.

(IEEE1320.2-98) IEEE Std 1320.2-1998, *IEEE Standard for Conceptual Modeling Language-Syntax and Semantics for IDEFIX97 (IDEFObjetct)*, IEEE, 1998.

(IEEE1362-98) IEEE Std 1362-1998, *IEEE Guide for Information Technology-System Definition-Concept of Operations (ConOps) Document*, IEEE, 1998.

(IEEE1465-98) IEEE Std 1465-1998//ISO/ IEC12119:1994, *IEEE Standard Adoption of International Standard ISO/IEC12119:1994(E), Information Technology-Software Packages-Quality requirements and testing*, IEEE, 1998.

(IEEEP1471-00) IEEE Std 1471-2000, *IEEE Recommended Practice for Architectural Descriptionos Software Intensive Systems*, Architecture Working Group of the Software Engineering Standards Committee, 2000.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/ IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

(IEEE14143.1-00) IEEE Std 14143.1-2000//ISO/ IEC14143-1:1998, *Information Technology-Software Measurement-Functional Size Measurement-Part 1: Definitions of Concepts*, IEEE, 2000.

CAPÍTULO 3
DISEÑO DE SOFTWARE

ACRÓNIMOS

ADL Lenguajes de Descripción de Arquitecturas.
CRC Tarjeta Clase-Responsabilidades-Colaboradores
ERD Diagrama Entidad-Relación
IDL Lenguaje de Descripción de Interfaz
DFD Diagrama de Flujo de DatosData Flow Diagram
PDL Pseudo-código y lenguaje de Diseño de Programa
CBD Diseño Basado en Componentes

INTRODUCCIÓN

El diseño se define en [IEEE610.12-90] como "el proceso para definir la arquitectura, los componentes, los interfaces, y otras características de un sistema o un componente" y "el resultado de este proceso." Visto como proceso, el diseño del software es la actividad del ciclo de vida de la cual los requisitos del software se analizan para producir una descripción de la estructura interna del software que servirá como la base para su construcción.

Más exacto, un diseño del software (el resultado) debe describir la arquitectura del software, en cómo la descomposición del software, la organización de los componentes, y los interfaces entre los mismos componentes. Debe también describir los componentes en un nivel de detalle que permita su construcción.

El diseño del software desempeña un papel importante en el desarrollo de software: permite que la Ingeniería del software produzca los diversos modelos para la solución que se pondrá en desarrollo. Podemos analizar y evaluar estos modelos para determinar si o no permitirán que se satisfaga los requisitos.

Podemos también examinar y evaluar varias soluciones alternativas y compensaciones. Finalmente, podemos utilizar los modelos que resultan para planear las actividades subsecuentes del desarrollo, además de usarlas como entrada o punto de partida de la construcción y prueba en un listado estándar de los procesos del ciclo de vida del software, tales como, procesos del ciclo de vida del software de IEEE/EIA 12207 [IEEE12207.0-96], diseño del software consiste en dos actividades que quepan entre el análisis de requisitos del software y la construcción del software:

- Diseño de la arquitectura del software (a veces llamado diseño de nivel superior): describiendo la estructura del software y organización e identificar a nivel superior los varios componentes
- Diseño detallado software del: describiendo cada componente suficientemente para tener en cuenta su construcción.

Referente al alcance del área del conocimiento del diseño del software (KA), la descripción actual de KA no discute todos los asuntos del nombre del cual contenga la palabra "diseño." En la terminología de Tom DeMarco (DeM99), el KA discutido en este capítulo trata principalmente del D-diseño (diseño de la descomposición, traza del software en partes de componentes). Sin embargo, debido a su importancia en el campo cada vez mayor de la arquitectura del software, también trataremos el diseño desde el punto de congelación (el diseño del patrón de familia, cual meta es establecer concordancias explotables en una familia del software). Por el contrario, el KA del diseño del software no trata el I-Diseño (el diseño de la Innovación, realizado generalmente durante el proceso de los requisitos del software con el objetivo de conceptuar y de especificar el software para satisfacer las necesidades y los requisitos), puesto que este asunto se debe considerar parte del análisis y la especificación de requisitos la descripción de KA del diseño del software se relaciona específicamente con los requisitos del software, la construcción del software, la gerencia de la ingeniería del software, la calidad del software, y las disciplinas relacionadas con la ingeniería del software

INTERRUPCIÓN DE LOS ASUNTOS PARA EL DISEÑO DEL SOFTWARE

1. Fundamentos del diseño del software

Los conceptos, las nociones, y la terminología introducida aquí forman una base subyacente para entender el papel y el alcance del diseño del software.

1.1. Conceptos generales de diseño

El software no es el único campo donde está implicado el diseño. En el sentido amplio, podemos ver diseño como forma de solucionar un problema. [Bud03: c1] Por ejemplo, el concepto de un problema travieso del problema-uno sin definitivo solución-es interesante en términos de entender los límites del diseño. [Bud04: c1] Un número de otras nociones y conceptos están también de interés en diseño el entender en su sentido general: metas, apremios, alternativas, representaciones, y soluciones. [Smi93]

1.2. Contexto del diseño del software

Para entender el papel del diseño del software, es importante entender el contexto, el ciclo de vida de la tecnología de dotación lógica. Así, es importante entender las características principales del análisis de requisitos del software contra diseño del software contra

la construcción del software contra la prueba del software. [IEEE12207.0-96]; Lis01: c11; 2 Mar; Pfl01: c2; Pre04: c2]

1.3. *Proceso del diseño del software*

El diseño del software generalmente se considera un proceso de dos etapas: [Bas03; Dor02: v1c4s2; Fre83: I; IEEE12207.0-96]; Lis01: c13; 2 Mar: D]

1.3.1. Diseño arquitectónico

El diseño arquitectónico describe cómo el software se descompone y se organiza en los componentes (la arquitectura) del software [IEEEP1471-00]

1.3.2. Diseño detallado

El diseño detallado describe el comportamiento específico de estos componentes.

La salida de este proceso es un sistema de modelos y los artefactos que registran las decisiones principales que se han tomado. [Bud04: c2; IEE1016-98; Lis01: c13; Pre04: c9]

1.4. Permitir técnicas

Según el diccionario del inglés de Oxford, un principio es "una verdad básica o una ley general… que se utiliza como una base del razonamiento o guía de la acción." Los principios del diseño del software, también llamados técnicas permisibles [Bus96], son nociones dominantes que consideran fundamental a los diversos acercamientos y conceptos del diseño del software. Las técnicas que lo permiten son las siguientes: [Bas98: c6; Bus96: c6; IEEE1016-98; Jal97: c5, c6; Lis01: c1, c3; Pfl01: c5; Pre04: c9]

1.4.1. Abstracción

La abstracción es "el proceso de olvidarse de la información para poder tratar las cosas que son diferentes como si fueran iguales." *Lis01+ En el contexto del diseño del software, dos mecanismos dominantes de la abstracción son parametrización y especificación. La abstracción por la especificación conduce a tres clases importantes de abstracción: abstracción procesal, abstracción de los datos, y abstracción del control (iteración). [Bas98: c6; Jal97: c5, c6; Lis01: c1, c2, c5, c6; Pre04: c1]

1.4.2. Acoplador y cohesión

El acoplador se define como la fuerza de las relaciones entre los módulos, mientras que la cohesión es definida por cómo los elementos que componen un módulo son relacionados. [Bas98: c6; Jal97: c5; Pfl01: c5; Pre04: c9]

1.4.3. Descomposición y modularización

Descomposición y modularización del software en partes más pequeñas e independientes, generalmente con la meta de poner diversas funcionalidades o responsabilidades en diversos componentes. [Bas98: c6; Bus96: c6; Jal97: c5; Pfl01: c5; Pre04: c9]

1.4.4. Encapsulación/el ocultar de la información

Medios que ocultan de la encapsulación/de la información que agrupan y que empaquetan los elementos y los detalles internos de una abstracción y que hacen esos detalles inaccesibles. [Bas98: c6; Bus96: c6; Jal97: c5; Pfl01: c5; Pre04: c9]

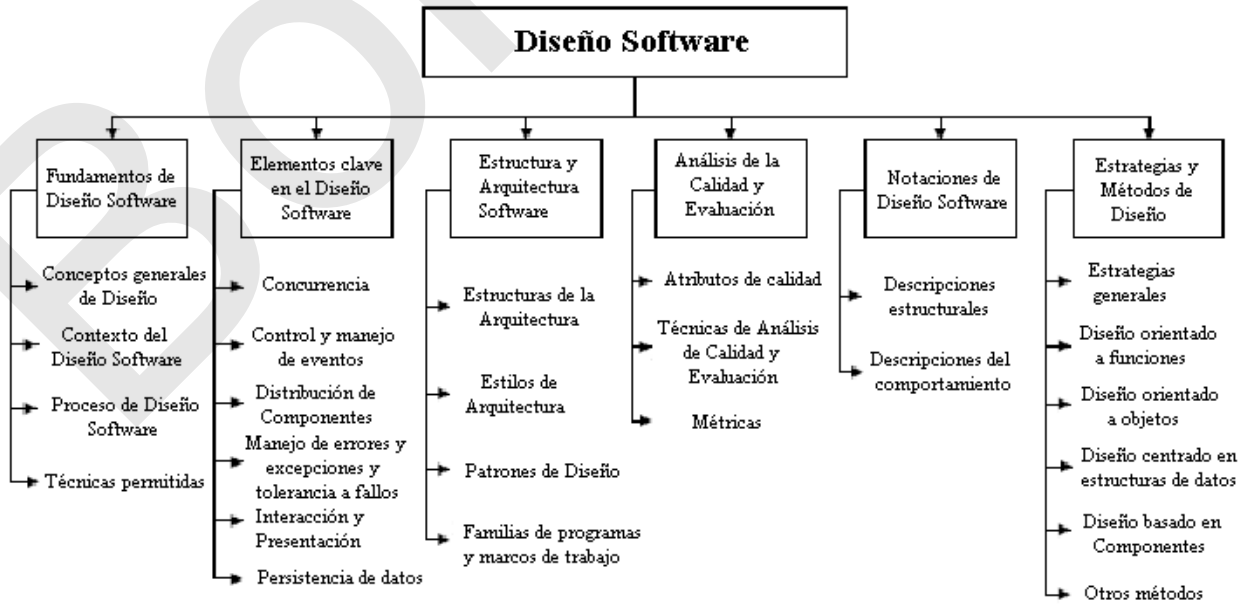


Figura1 Descomposición de los temas del KA de Diseño Software

1.4.5. Separación del interfaz y de la puesta en práctica

La separación del interfaz y de la puesta en práctica implica el definir de un componente especificando un interfaz público, a parte de los detalles de cómo se observa el componente. [Bas98: c6; Bos00: c10; Lis01: c1, c9]

1.4.6. Desahogo, lo completo y deshaciéndose de lo primitivo

Alcanzando desahogo, lo completo, y medios no primitivos se asegura que un componente de software captura todas las características importantes de una abstracción, y nada más. [Bus96: c6; Lis01: c5].

2. Cuestiones claves en diseño del software

Se tiene que tener en cuenta a la hora de diseñar software una serie de principios claves. Algunos son preocupaciones que todo el software debe tratar -por ejemplo, funcionamiento de la calidad. Otra edición importante es cómo se descomponer, se organiza, y constituyen los paquetes de software. Esto es tan fundamental que todos los acercamientos del diseño deben tratarlo de un modo u otro (véase las técnicas del asunto 1.4 y la subárea 6, los métodos que permiten el diseño del software). En cambio, otras ediciones "se ocupan de un cierto aspecto del comportamiento del software que no está en el dominio del uso, pero que trata algunos de los dominios de soporte." *Bos00+ Tales ediciones, que interseccionaron a menudo la funcionalidad del sistema, se han referido como aspectos: "*aspectos+ tender para no ser unidades de la descomposición funcional del software, sino algo para ser las características que afectan el funcionamiento o la semántica de los componentes de manera sistemáticas" (Kic97). Un número de estas claves, ediciones de la cruz-corte son los siguientes (presentado en orden alfabético):

Concurrencia

Cómo descomponer el software en procesos, tareas, e hilos y reparto con eficacia relacionada, atomicidad, la sincronización, y ediciones programar. [Bos00: c5; 2 Mar: CSD; Mey97: c30; Pre04: c9]

Control y dirección de acontecimientos

Cómo organizar datos y controlar flujo, cómo manejar acontecimientos reactivos y temporales a través de varios mecanismos tales como invocación y servicios repetidos implícitos. [Bas98: c5; Mey97: c32; Pfl01: c5]

2.3 Distribución de componentes

Cómo distribuir el software a través del hardware, cómo los componentes se comunican, cómo el middleware se puede utilizar para ocuparse de software heterogéneo. [Bas03: c16; Bos00: c5; Bus96: c el 2 Mar de 94: DD; Mey97: c30; Pre04: c30]

2.1. Dirección del error y de excepción y tolerancia de fallos

Cómo prevenir y tolerar averías y ocuparse de condiciones excepcionales. [Lis01: c4; Mey97: c12; Pfl01: c5]

2.1. Interacción y presentación

Cómo estructurar y organizar las interacciones con los usuarios y la presentación de la información (por ejemplo, separación de la presentación y de la lógica del negocio usando el acercamiento del Modelo-Vista-Regulador). [Bas98: c6; Bos00: c5; Bus96: c2; Lis01: c13; Mey97: c32] Debe ser observado que este asunto no está sobre especificar los detalles del interfaz utilizador, que es la tarea del diseño del interfaz utilizador (una parte de ergonómica del software); ver las disciplinas relacionadas de la tecnología de dotación lógica.

2.1. Persistencia de los datos

Cómo los datos duraderos deben ser dirigidos. [Bos00: c5; Mey97: c31];

3. Estructura y arquitectura del software

En su sentido terminante, una arquitectura del software es "una descripción de los subsistemas y de los componentes de un sistema de software y de las relaciones entre ellas." (Bus96: c6) La arquitectura procura así definir la estructura interna - según el diccionario del inglés de Oxford, "la manera de la cual se construye o se organiza algo" - del software que resulta. Durante los mid-1990s, sin embargo, la arquitectura del software comenzó a emerger como disciplina más amplia que implicaba el estudio de las estructuras y de las arquitecturas del software en una manera más genérica [Sha96]. Esto dio lugar a un número de ideas interesantes sobre diseño del software en diversos niveles de la abstracción. Algunos de estos conceptos pueden ser útiles durante el diseño arquitectónico (por ejemplo, estilo arquitectónico) del software específico, así como durante su diseño detallado (por ejemplo, patrones de nivel inferior del diseño). Pero pueden también ser útiles para diseñar sistemas genéricos, conduciendo al diseño de familias de los programas (también conocidos como líneas de productos). Interesante, la mayor parte de estos conceptos se pueden considerar como tentativas de describir, y de reutilizar así, conocimiento genérico del diseño.

3.1. Estructuras y puntos de vista arquitectónicos

Diversas facetas de alto nivel de una poder del diseño del software y deben ser descritas y ser documentadas. Estas facetas a menudo se llaman las opiniones: "Una visión representa un aspecto parcial de una arquitectura del software que demuestre características específicas de un sistema de software" *Bus96: c6]. Estas visiones distintas pertenecen a las ediciones distintas asociadas a diseño del software - por ejemplo, la visión lógica (que satisface los requisitos funcionales) contra la visión de proceso (ediciones de la concurrencia) contra la visión física (ediciones de la distribución) contra la opinión del desarrollo (cómo el diseño se analiza en unidades de la puesta en práctica). Otros autores utilizan diversas terminologías, como del comportamiento contra funcional contra estructural contra los datos que modelan opiniones. Resumiendo, un diseño del software es un artefacto múltiple producido por el proceso del diseño e integrado generalmente por visiones relativamente independientes y ortogonal. [Bas03: c2; Boo99: c31; Bud04: c5; Bus96: c6; IEEE1016-98; IEEE1471-00] Estilos arquitectónicos (patrones arquitectónicos macro)

Un estilo arquitectónico es "un sistema de apremios en una arquitectura *que+ define un sistema o una familia de arquitecturas que las satisfagan" *Bas03: c2+. Un estilo arquitectónico se puede considerar así mientras que un meta-modelo que pueda proporcionar la organización de alto nivel del software (su arquitectura macro). Los varios autores han identificado un número de estilos arquitectónicos importantes. [Bas03: c5; Boo99: c28; Bos00: c6; Bus96: c1, c6; Pfl01: c5]

- Estructura general (por ejemplo, capas, pipas, y filtros, pizarra)
- Sistemas distribuidos (por ejemplo, servidor de cliente, tres gradas, corredor)
- Sistemas interactivos (por ejemplo, regulador de la Modelo-Vista, Presentación-Abstracción-Control)
- Sistemas adaptables (por ejemplo, micro-núcleo, reflexión)
- Oros (por ejemplo, hornada, intérpretes, control, basados en las reglas de proceso).

3.2. Patrones del diseño (patrones arquitectónicos micro).

Resumido brevemente, un patrón es "una solución común a un problema común en un contexto dado." (Jac99) Mientras que los estilos arquitectónicos se pueden ver como patrones que describen la organización de un nivel alto del software (su arquitectura macro); otros patrones del diseño se pueden utilizar para describir los detalles en un nivel más bajo, más local (su arquitectura micro). [Bas98: c13; Boo99: c28; Bus96: c1; 2 Mar: DP]

- Patrones de creación (por ejemplo, builder, factory, prototipo, y singleton)
- Patrones estructurales (por ejemplo, adapter, bridge, composite, decorator, façade, flyweight, and proxy)
- Patrones del comportamiento (por ejemplo, command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor)

3.3 Familias de programas y de marcos.

Una posible opción para permitir la reutilización de los diseños y de los componentes del software es diseñar las familias del software, también conocidas como líneas del producto de software. Estas pueden ser hechas identificando las concordancias entre los miembros de tales familias y por los componentes reutilizables y adaptables entre miembros de la familia. [Bos00: c7, c10; Bas98: c15; Pre04: c30] En programación orientada a objetos, una clave relacionada es la del marco: un subsistema parcialmente completo del software que puede ser ampliado apropiadamente instalando los plug-ins específicos (también conocidos como puntos calientes). [Bos00: c11; Boo99: c28; Bus96: c6]

4. Análisis y evaluación de la calidad del diseño del software

Esta sección incluye generalidades de la calidad y evaluación que se relacionen específicamente con el diseño del software. La mayoría se cubren de una manera general en Software Quality KA

4.1. Cualidades de los atributos

Varias atributos son generalmente importantes para obtener un diseño del software de buenos calidad - varios "ilities" (capacidad de mantenimiento, portabilidad, testeo, trazabilidad), los varios "nesses" (corrección, robustez), incluyendo la "aptitud del propósito." *Bos00: c5; Bud04: c4; Bus96: c6; ISO9126.1-01; ISO15026-98; Mar de 94: D; Mey97: c3; Pfl01: c5] Una distinción interesante es la que está entre las cualidades de la calidad discernible en el tiempo de ejecución (funcionamiento, seguridad, disponibilidad, funcionalidad, utilidad), ésas no discernibles en el tiempo de ejecución (modificabilidad, portabilidad, reutilidad, integridad, y testeabilidad), y ésas relacionadas con las calidades intrínsecas de la arquitectura (integridad, corrección, y lo completo, capacidad conceptuales de la estructura). [Bas03: c4]

4.2 Técnicas de evaluación y calidad del análisis.

Varias técnicas pueden ayudar a asegurar la calidad de un diseño del software:

- Revisiones de diseño del software: informal o semiformal, a menudo basado en grupo, las técnicas para verificar y para asegurar la calidad de los artefactos del diseño (por ejemplo, revisiones de la arquitectura [Bas03: c11], revisiones de diseño, e inspecciones [Bud04: c4; Fre83: VIII; IEEE1028-97; Jal97: c5, c7; Lis01: c14; Pfl01: c5], técnicas

basadas en panorama [Bas98: c9; Bos00: c5], y la toma de los requisitos [Dor02: v1c4s2; Pfl01:])

- Análisis estático: análisis estático formal o semiformal (ningún ejecutable) que se puede utilizar para evaluar un diseño (por ejemplo, el análisis o el cross-checking automatizado) [Jal97 del fault-tree: c5; Pfl01:]
- Simulación y prototipado: técnicas dinámicas para evaluar un diseño (por ejemplo, simulación o prototipo de la viabilidad [Bas98 del funcionamiento: c10; Bos00: c5; Bud04: c4; Pfl01: c5])

4.3 Medidas.

Las medidas se pueden utilizar para determinar o para estimar cuantitativamente varios aspectos del tamaño, de la estructura, o de la calidad de un diseño del software. La mayoría de las medidas se han propuesto que dependen generalmente del acercamiento usado para producir el diseño. Estas medidas se clasifican en dos amplias categorías:

- Diseño de medidas orientada a función (estructuradas): Estructura del diseño, obtenida sobre todo con la descomposición funcional; representado generalmente como una carta de estructura (a veces llamada un diagrama jerárquico) en la cual varias medidas pueden ser computadas [Jal97: c5, c7, Pre04:]
- Diseño de medidas orientada a objetos: La estructura total del diseño se representa a menudo como diagrama de la clase, en el cual varias medidas pueden ser computadas. Las medidas en las características del contenido interno de cada clase pueden también ser computadas [Jal97: c6, c7; Pre04: c15]

5 Notaciones del diseño del software

Muchas notaciones e idiomas existen para representar los artefactos del diseño del software. Algunos se utilizan principalmente para describir la organización estructural de un diseño, otras para representar comportamiento del software. Ciertas notaciones se utilizan sobre todo durante el diseño arquitectónico y otros principalmente durante el diseño detallado, aunque algunas notaciones se pueden utilizar en ambos pasos. Además, algunas notaciones se utilizan sobre todo en el contexto de métodos específicos (véase el *Software Design Strategies and Methods* subárea). Aquí, se categorizan en las notaciones para describir la opinión (estática) estructural contra la visión (dinámica) del comportamiento.

5.1 Descripción estructural (vista estática):

Las siguientes notaciones, sobre todo (pero no siempre) gráficas, describen y representan los aspectos estructurales del diseño de software – las cuales, describen los componentes principales y cómo se interconectan (visión estática):

- Lenguajes descriptivos de la arquitectura: textuales, a menudo formal, los lenguajes describían una arquitectura del software en términos de componentes y conectadores [Bas03: c12]
- Diagramas de la clase y objeto: usados para representar un sistema de clases (y de objetos) y de sus correlaciones [Boo99: c8, c14; Jal97:]
- Diagramas de componentes: usados para representar un sistema de componentes ("parte física y reemplazable de un sistema al cual conforma y proporciona la realización de un sistema de interfaces" *Boo99+) y de sus correlaciones *Boo99: +
- Tarjetas del colaborador de la responsabilidad de la clase (CRCs): denotan los nombres de los componentes (clases), de sus responsabilidades, y nombres de sus componentes de colaboración' [Boo99: c4;]
- Diagramas de despliegue: representar un sistema de nodos (físico) y de sus correlaciones, y, así, modelaban los aspectos físicos de un sistema [Boo99:]
- Diagramas de la Entidad-relación (ERDs): representan modelos conceptuales de los datos almacenados en los sistemas de información [Bud04: c6; Dor02: v1c5; 2]
- Lenguaje descriptivo de la interfaz (IDLS): programación como lenguajes usados para definir los interfaces (nombres y tipos de operaciones exportadas) de los componentes de software [Bas98: c8; Boo99:]
- Diagramas de la estructura de Jackson: Usados para describir las estructuras de datos en términos de secuencia, selección, e iteración [Bud04: c6; 2 Mar:
- Estructura de cartas: Usados para describir la estructura que llamaba de los programas (el módulo llama, y es llamado por otro módulo) [Bud04: c6; Jal97: c5; 2 Mar: Dr; Pre04: c10]

5.2 Descripciones del comportamiento (visión dinámica):

Las siguientes notaciones y lenguajes, algunos gráficos y otros textuales, se utilizan para describir el comportamiento dinámico del software y de los componentes. Muchas de estas notaciones son útiles sobre todo, pero no exclusivamente, durante el diseño detallado.

- Diagramas de actividad: Muestran el flujo del control de la actividad ("ejecución no-atómica en curso dentro de una máquina del estado") a la actividad *Boo99: +
- Diagramas de colaboración: Muestran las interacciones que ocurren entre un grupo de objetos, donde está el énfasis en los objetos, sus acoplamientos, y los mensajes que intercambian en estos acoplamientos [Boo99:]

- Organigramas de datos: Muestran los flujos de datos entre un sistema y los procesos [Bud04: c6; 2 Mar: Dr; Pre04:]
- Tablas y diagramas de decisión: representan combinaciones complejas de las condiciones y de las acciones [Pre04:]
- Organigramas y organigramas estructurados: Representan el control de flujo y de las acciones asociadas que se realizarán [Fre83: VII; 2 Mar: Dr; Pre04:]
- Diagramas de secuencia: Muestran las interacciones entre un grupo de objetos, con énfasis sobre el tiempo de ordenación de mensajes [Boo99:]
- Transición de estado y diagramas de carta de estado: demostraban el control de flujo de estado a estado en una máquina de estados[Boo99: c24; Bud04: c6; 2 Mar: Dr; Jal97:]
- Lenguajes formales de especificación: Lenguajes textuales que utilizan nociones básicas de matemáticas (por ejemplo, lógica, sistema, secuencia), para obtener de forma rigurosa y abstracta, definir interfaces y comportamientos del componente de software, a menudo en términos de pre y post-condiciones [Bud04: c18; Dor02: v1c6s5; Mey97:]
- Lenguajes del diseño de pseudo código del programa (PDLs): Programa estructurado como los lenguajes usados para describir, generalmente en la etapa detallada del diseño, el comportamiento de un procedimiento o el método [Bud04: c6; Fre83: VII; Jal97: c7; Pre04: c8, c11]

6 Estrategias y métodos del diseño de software

Existen varias estrategias generales para ayudar a dirigir el proceso de diseño. [Bud04: c9, 2 Mar: D] Al contrario que en las estrategias generales, los métodos son más específicos, sugieren y proporcionan generalmente un sistema de notaciones que se utilizarán con el método, una descripción del proceso que se utilizará después del método y un sistema de pautas al usar el método. [Bud04: c8] Tales métodos son útiles como medios de transferir conocimiento y como marco común para los equipos de los ingenieros de software. [Bud03: c8] Ver también Herramientas y métodos KA de Ingeniería de software.

6.1 Estrategias generales

Algunos de los ejemplos citados de las estrategias generales útiles en el proceso del diseño son dividir-y-conquistar y el refinamiento[Bud04: c12; Fre83: V], de arriba hacia abajo contra las estrategias bottom-up [Jal97: c5; Lis01: c13], abstracción de los datos y el ocultar de la información [Fre83: V], uso de la heurística [Bud04: c8], uso de patrones y lenguajes de patrones [Bud04: c10; Bus96: c5], uso de un acercamiento iterativo e incremental. [Pfl01: c2]

6.2 Diseño (estructurado) orientado a función:

[Bud04: c14; Dor02: v1c6s4; Fre83: V; Jal97: c5; Pre04: está uno c9, c10]
Esto es uno de los métodos clásicos del diseño de software, donde los centros de descomposición identifican las funciones del software y después elaboran y refinan de una manera de arriba hacia abajo. El diseño estructurado se utiliza generalmente después de análisis estructurado, produciendo así, entre otras cosas, organigramas de datos y de descripciones de proceso asociados. Los investigadores han propuesto varias estrategias (por ejemplo, análisis de la transformación, análisis de la transacción) y la heurística (por ejemplo, fan-in/fan-out, alcance del efecto contra el alcance del control) para transformar un DFD en una arquitectura del software representada generalmente como carta de estructura.

6.3 Diseño orientado a objeto

[Bud0: c16; Dor02: v1: c6s2, s3; Fre83: VI; Jal97: c6; 2 Mar: D; Pre04: c9]
Numerosos métodos de diseño de software basados en objetos han sido propuestos. El campo se ha desarrollado basado en el diseño objeto de los mediados de los años ochenta (sustantivo = objeto; verbo = método; adjetivo = cualidad) con el diseño orientado a objetos, donde la herencia y el polimorfismo desempeñan un papel importante, el campo del diseño del componente-basado, donde la meta información puede ser definida y ser alcanzada (con la reflexión, por ejemplo). Aunque las raíces del diseño Orientado a Objetos provienen del concepto de la abstracción de los datos, el diseño responsabilidad-conducido también se ha propuesto como alternativo al diseño Orientado a Objetos.

6.4 Diseño Dato-Estructura-Centrado

[Bud04: c15; Fre83: III, VII; 2 Mar02:D]
El diseño Dato-estructura-centrado (por ejemplo, Jackson, Warnier-Orr) comienza desde las estructuras de datos que un programa manipula más bien que desde las funciones que realiza. La Ingeniería de software primero describe las estructuras de datos de entrada y de salida (que usan los diagramas de la estructura de Jackson, por ejemplo) y en seguida desarrolla la estructura de control del programa basada en estos diagramas de estructura de datos. La varia heurística se ha propuesto para tratar como caso especial, cuando hay una unión mal hecha entre la entrada y las estructuras de la salida.

6.5 Diseño basado en componente (CBD):

Un componente de software es una unidad independiente, teniendo bien definidos los interfaces y dependencias que se pueden componer y desplegar independientemente. El diseño basado en componente trata las ediciones relacionadas con el abastecimiento,

desarrollo, e integración de tales componentes para mejorar la reutilización. [Bud04: c11]

6.6 Otros métodos

Otros interesantes pero menos aprovechados también existen: métodos formales y rigurosos [Bud04: c18; Dor02: c5; Fre83; Mey97: c11; Pre04: c29] y métodos transformacionales. [Pfl98: c2]

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Bas03] {Bas98}	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[iEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal 97]	[Li s01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
1.Fundamentos del Diseño de Software																c11s1				
1.1Conceptos Generales de Diseño				c1												c13s1, c13s2				
1.2El contexto de Diseño Software											*					c1s1, c13s2, c3s1-c3s3,c125-128, c9s1-c9s3	D		c2s2	c2
1.3 El Proceso de Diseño Software	c2s1			c2		v1c4s2	2-22	*		*	*						D			c9
1.4 Técnicas Permitidas	{c6s1}		c10s3		c6s3			*						c5s1, c5s2, c6s2					c5s2, c5s5	c9
2.Elementos clave en el Diseño Software																				
2.1 Concurrencia			c5s4.1														CSD	c30		c9
2.2 Control y manejo de eventos	{c5s2}																{DD}	c32s4, c32s5	c5s3	
2.3 Distribución de componentes	c16s3, c16s4		c5s4.1		c2s3													c30		c30
2.4 Manejo de errores y excepciones y tolerancia a fallos																c4s3-c4s5		c12	c5s5	
2.5 Interacción y presentación	{c6s2}		c5s4.1		c2s4											c13s3		c32s2		
2.6 Persistencia de Datos			c5s4.1															c31		

* ver siguiente sección

	[Bas03] {Bas98}	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[iEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal 97]	[Li s01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
3. Estructura y Arquitectura Software		c31																		
3.1 Estructuras de la Arquitectura	c2s5	c28		c5	c6s1			*		*										
3.2 Estilos de Arquitectura	c5s9	c28	c6s3.1		c1s1-c1s3, c6s2													c2s3		
3.3 Patrones de Diseño	{c13s3}	c28			c1s1-c1s3											DP				
3.4 Familias de programas y Marcos de Trabajo	{c15s1, c15s3}		c7s1, c7s2, c10s2-c10s4, c11s2 –c11s4		c6s2														c30	
4. Análisis de la Calidad y Evaluación del Diseño de Software																				
4.1 Atributos de Calidad	c4s2		c5s2.3	c4	c6s4							*	*			{D}	c3	c5s5		
4.2 Técnicas de Análisis de Calidad y Evaluación	c11s3, [c9s1, c9s2, c10s2, c10s3}		c5s2.1, c5s2.2, c5s3, c5s4	c4		v1c4s2	542-576		*					c5s5, c7s3	c14s1			c5s6, c5s7, c11s5		
4.3 Métricas														c5s6, c6s5, c7s4					c15	

	[Bas03] {Bas98}	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[iEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal 97]	[Li s01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
5. Notaciones de Diseño de Software																				
5.1 Descripciones estructurales (Vista Estática	{c8s4} c12s1, c12s2	c4, c8 c11, c12, c14, c30, c31		c6	429									c5s3, c6s3		DR				
5.2 Descripciones del Comportamiento (Vista Dinámica)				c6, c18		v1c5	485-490, 506-513							c7s2		DR			c10	
6. Métodos y Estrategias en Diseño de Software																	c11		c8,c11	
6.1 Estrategias generales				c8, c10, c12	c5s1-c5s4		304-320, 533-539							c5s1.4	c13s13					
6.2 Diseño Orientado a Funciones (Estructurado)							328-352							c5s4				c2s2		
6.3 Diseño Orientado a Objetos							420-436							c6s4		D			c9,c10	
6.4 Diseño centrado en Estructuras de Datos							201-120, 514-532									D			c9	
6.5 Diseño Basado en Componentes (CBD)																				
6.6 Otros						181-192											c11	c2s2	c29	

REFERENCIAS RECOMENDADAS PARA EL DISEÑO DE SOFTWARE

[Bas98] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.[Bas03] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, second ed., Addison-Wesley, 2003.

[Boo99] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[Bos00] J. Bosch, Design & Use of Software Architectures: Adopting and Evolving a Product-Line Approach, first ed., ACM Press, 2000.

[Bud04] D. Budgen, *Software Design*, second ed., Addison-Wesley, 2004.

[Bus96] F. Buschmann et al., Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1996.

[Dor02] M. Dorfman and R.H. Thayer, eds., *Software Engineering* (Vol. 1 & Vol. 2), IEEE Computer Society Press, 2002.

[Fre83] P. Freeman and A.I. Wasserman, *Tutorial on Software Design Techniques*, fourth ed., IEEE Computer Society Press, 1983.

[IEEE610.12-90] IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

[IEEE1016-98] IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions, IEEE, 1998.

[IEEE1028-97] IEEE Std 1028-1997 (R2002), *IEEE Standard for Software Reviews*, IEEE, 1997.

[IEEE1471-00] IEEE Std 1471-2000, *IEEE Recommended Practice for Architectural Description of Software Intensive Systems*, Architecture Working Group of the Software Engineering Standards Committee, 2000.

[IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes, IEEE, 1996.

[ISO9126-01] ISO/IEC 9126-1:2001, Software Engineering Product Quality—Part 1: Quality Model, ISO and IEC, 2001.

[ISO15026-98] ISO/IEC 15026-1998, Information Technology — System and Software Integrity Levels, ISO and IEC, 1998.

[Jal97] P. Jalote, An Integrated Approach to Software Engineering, second ed., Springer-Verlag, 1997.

[Lis01] B. Liskov and J. Guttag, Program Development in Java: Abstraction, Specification, and Object-Oriented Design, Addison-Wesley, 2001.

[Mar94] J.J. Marciniak, Encyclopedia of Software Engineering, J. Wiley & Sons, 1994. The references to the Encyclopedia are as follows:

CBD = Component-Based Design
D = Design
DD = Design of the Distributed System
DR = Design Representation

[Mar02] J.J. Marciniak, Encyclopedia of Software Engineering, second ed., J. Wiley & Sons, 2002.

[Mey97] B. Meyer, Object-Oriented Software Construction, second ed., Prentice-Hall, 1997.

[Pfl01] S.L. Pfleeger, Software Engineering: Theory and Practice, second ed., Prentice-Hall, 2001.

[Pre04] R.S. Pressman, Software Engineering: A Practitioner's Approach, sixth ed., McGraw-Hill, 2004.

[Smi93] G. Smith and G. Browne, "Conceptual Foundations of Design Problem-Solving," IEEE Transactions on Systems, Man and Cybernetics, vol. 23, iss. 5, 1209-1219, Sep.-Oct. 1993.

APÉNDICE A. LISTA DE LECTURAS COMPLEMENTARIAS

(Boo94a) G. Booch, Object Oriented Analysis and Design with Applications, second ed., The Benjamin/Cummings Publishing Company, 1994.

(Coa91) P. Coad and E. Yourdon, Object-Oriented Design, Yourdon Press, 1991.

(Cro84) N. Cross, Developments in Design Methodology, John Wiley & Sons, 1984.

(DSo99) D.F. D'Souza and A.C. Wills, Objects, Components, and Frameworks with UML — The Catálisis Approach, Addison-Wesley, 1999.

(Dem99) T. DeMarco, "The Paradox of Software Architecture and Design," Stevens Prize Lecture, Aug. 1999.

(Fen98) N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, second ed., Internacional Thomson Computer Press, 1998.

(Fow99) M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

(Fow03) M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2003.

(Gam95) E. Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

(Hut94) A.T.F. Hutt, Object Analysis and Design — Comparison of Methods. Object Analysis and Design — Description of Methods, John Wiley & Sons, 1994.

(Jac99) I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.

(Kic97) G. Kiczales et al., "Aspect-Oriented Programming," presented at ECOOP '97 — Object-Oriented Programming, 1997.

(Kru95) P. B. Kruchten, "The 4+1 View Model of Architecture," IEEE Software, vol. 12, iss. 6, 42-50, 1995

(Lar98) C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice-Hall, 1998.

(McC93) S. McConnell, Code Complete: A Practical Handbook of Software Construction, Microsoft Press, 1993.

(Pag00) M. Page-Jones, Fundamentals of Object-Oriented Design in UML, Addison-Wesley, 2000.

(Pet92) H. Petroski, To Engineer Is Human: The Role of Failure in Successful Design, Vintage Books, 1992.

(Pre95) W. Pree, Design Patterns for Object-Oriented Software Development, Addison-Wesley and ACM Press, 1995.

(Rie96) A.J. Riel, Object-Oriented Design Heuristics, Addison-Wesley, 1996.

(Rum91) J. Rumbaugh et al., Object-Oriented Modeling and Design, Prentice-Hall, 1991.

(Sha96) M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, 1996.

(Som05) I. Sommerville, Software Engineering, seventh ed., Addison-Wesley, 2005.

(Wie98) R. Wieringa, "A Survey of Structured and Object-Oriented Software Specification Methods and Techniques," ACM Computing Surveys, vol. 30, iss. 4, 1998, pp. 459-527.

(Wir90) R. Wirfs-Brock, B. Wilkerson, and L. Wiener, Designing Object-Oriented Software, Prentice-Hall, 1990.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990.

(IEEE1016-98) IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions, IEEE, 1998.

(IEEE1028-97) IEEE Std 1028-1997 (R2002), IEEE Standard for Software Reviews, IEEE, 1997.

(IEEE1471-00) IEEE Std 1471-2000, IEEE Recommended Practice for Architectural Descriptions of Software-Intensive Systems, Architecture Working Group of the Software Engineering Standards Committee, 2000.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes, vol. IEEE, 1996.

(ISO9126-01) ISO/IEC 9126-1:2001, Software Engineering-Product Quality-Part 1: Quality Model, ISO and IEC, 2001.

(ISO15026-98) ISO/IEC 15026-1998 Information Technology — System and Software Integrity Levels, ISO and IEC, 1998.

CAPÍTULO 4
CONSTRUCCIÓN DEL SOFTWARE

ACRÓNIMOS

OMG	Grupo de Gestión de Objetos
UML	Lenguaje Unificado de Modelado

INTRODUCCIÓN

El término construcción del software hace referencia a la creación detallada de software operativo y significativo, por medio de una combinación de codificación, verificación, pruebas unitarias, pruebas de integración y depuración.

El Área de Conocimiento de la Construcción del Software está vinculada a todas las otras KAs (Áreas de Conocimiento), más fuertemente al Diseño del Software y a las Pruebas del Software. Esto se debe a que el proceso mismo de construcción del software cubre tanto el diseño significativo de software como las actividades de pruebas. También utiliza las salidas del diseño y proporciona una de las entradas para las pruebas, consistiendo estas actividades en el diseño y en las pruebas, y en este caso no en las KAs. Las fronteras detalladas entre el diseño, la construcción y las pruebas (si es que existen) varían dependiendo de los procesos de ciclo de vida del software utilizados en un proyecto.

A pesar de que se pueda realizar parte del diseño detallado antes de la construcción, mucho del trabajo del diseño se lleva a cabo durante la actividad misma de la construcción. Por lo que el KA de Construcción del Software está vinculado muy de cerca al KA de Diseño del Software.

Por medio de la construcción los ingenieros del software realizan tanto pruebas unitarias, como pruebas de integración de su trabajo. De tal manera que el KA de Construcción del Software está también vinculada de cerca al KA de Pruebas del Software.

La construcción del software, por lo general, produce el mayor número de elementos de configuración que se necesitan gestionar en un proyecto de software (archivos de código fuente, contenido, casos de pruebas, etc.). De este modo, el KA de Construcción del Software también está vinculado de cerca al KA de Gestión de la Configuración del Software.

Dado que la construcción del software tiene una gran dependencia de las herramientas y de los métodos, y de que se trata probablemente del KA que más herramientas tienen y utiliza, está vinculada al KA de Herramientas y Métodos de la Ingeniería del Software.

Aunque la calidad del software es importante en todas las KAs, el código es el último entregable de un proyecto de software y, por tanto, la Calidad del Software está vinculada de cerca a la Construcción del Software.

Entre las Disciplinas Descritas de la Ingeniería del Software el KA de Construcción del Software es lo más parecido a la ciencia informática en su uso del conocimiento de algoritmos y de las prácticas detalladas de codificación, ambas son consideradas, con frecuencia, como pertenecientes al dominio de la ciencia informática. También está relacionada con la gestión del proyecto en la medida en que la gestión de la construcción pueda presentar retos considerables.

DESCOMPOSICIÓN DE LOS TEMAS DE CONSTRUCCIÓN DEL SOFTWARE

A continuación se presenta la descomposición del KA de la Construcción del Software junto con breves descripciones de los temas más importantes asociados a este. La figura 1 ofrece una representación gráfica de la descomposición de alto nivel de las divisiones de este KA.

1. Fundamentos de la Construcción del Software

Los fundamentos de la construcción del software incluyen:

- Minimizar la complejidad
- Anticiparse a los cambios
- Construir para verificar
- Estándares en la construcción

Los tres primeros conceptos se aplican tanto al diseño como a la construcción. Las siguientes secciones definen estos conceptos y describen cómo se aplican a la construcción.

1.1 Minimizar la complejidad

[Bec99; Ben00; Hun00; Ker99; Mag93; McC04]

El principal factor que hace que la gente utilice ordenadores consiste en la limitadísima capacidad que tiene para retener estructuras complejas e información en su memoria operativa, especialmente durante largos períodos de tiempo. Esto lleva a uno de los más fuertes impulsores de la construcción del software: minimizar la complejidad. La necesidad de reducir la complejidad se aplica esencialmente a todo aspecto de la construcción del software, y es de crítica importancia para el proceso de verificación y pruebas de las construcciones del software.

En la construcción del software sólo se alcanza una reducida complejidad por medio del énfasis en la

creación de código que sea simple y legible, y no tanto inteligente.
Se logra minimizar la complejidad mediante el uso de estándares, como se ve en el apartado 1.4 *Estándares de Construcción*, y mediante numerosas técnicas específicas que están resumidas en el apartado 3.3 *Codificación.* También se apoya en las técnicas de calidad enfocadas a la construcción resumidas en el apartado 3.5 Calidad de la Construcción.

1.2 Anticiparse a los cambios

[Ben00; Ker99; McC04]
La mayoría del software cambiará a lo largo del tiempo, y el anticiparse a los cambios dirige muchos aspectos de la construcción del software. El software es inevitablemente parte de los ambientes externos que cambian continuamente, y los cambios en esos ambientes externos afectan al software de diversos modos.
El anticiparse a los cambios se apoya en muchas técnicas específicas resumidas en el apartado 3.3 *Codificación.*

1.3 Construir para verificar

[Ben00; Hun00; Ker99; Mag93; McC04]
Construir para verificar significa construir software de tal manera que los ingenieros del software puedan sacar a relucir los fallos con facilidad al estar escribiendo el código, además de cuando realizan pruebas independientes y actividades operacionales. Las técnicas específicas que sirven de base para construir con vistas a verificar incluyen el seguimiento de estándares de codificación que permitan las revisiones del código, las pruebas unitarias, la organización del código que permita pruebas automáticas, y el uso restringido de estructuras de lenguaje que sean complejas o difíciles de entender, entre otras.

1.4 Estándares en la construcción

[IEEE12207-95; McC04]
Los estándares que afectan directamente a elementos de la construcción incluyen:

- Métodos de comunicación (por ejemplo, estándares para los formatos de los documentos y de los contenidos)
- Programación de lenguajes (por ejemplo, estándares de lenguaje para lenguajes como Java y C++)
- Plataformas (por ejemplo, estándares de interfaces del programador para llamadas al sistema operativo)
- Herramientas (por ejemplo, estándares diagramáticos para notaciones como UML (Lenguaje Unificado de Modelado))

Uso de estándares externos. Construir depende del uso de estándares externos para los lenguajes de construcción, las herramientas de construcción, las interfaces técnicas, y las interacciones entre la Construcción del Software y las otras KAs. Los estándares provienen de numerosas fuentes, incluyendo las especificaciones de interfaz del hardware y del software, tales como el Grupo de Gestión de Objetos (OMG) y las organizaciones internacionales tales como la IEEE o ISO.
Uso de estándares internos. Los estándares también pueden crearse partiendo de una base organizacional a un nivel corporativo o para su uso en proyectos específicos. Estos estándares permiten la coordinación de actividades de grupo, el minimizar la complejidad, el anticipar los cambios y el construir para verificar.

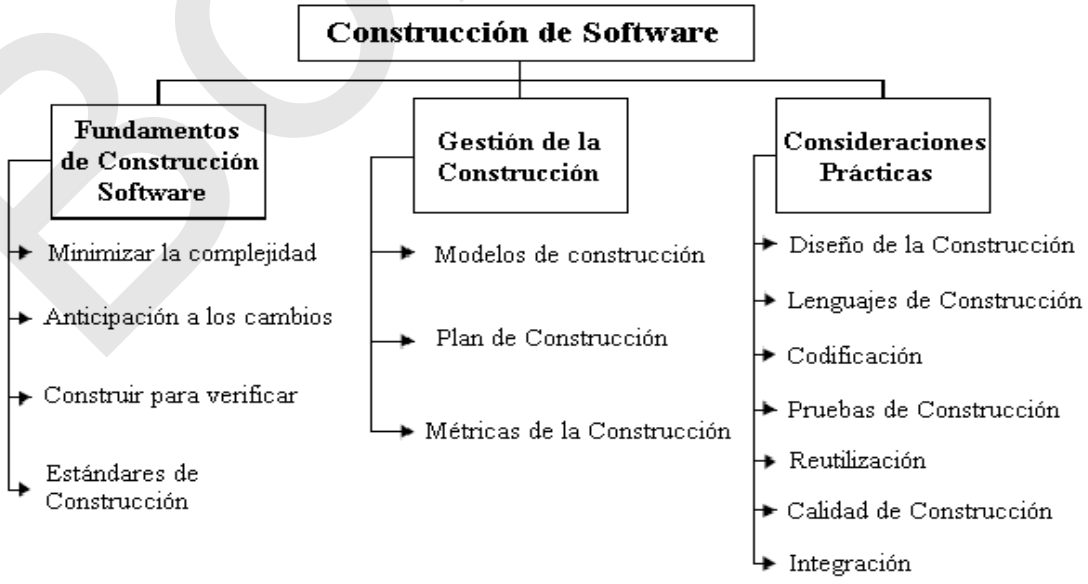


Figura 1 Descomposición de los temas del KA de Construcción de Software

2. Gestión de la Construcción

2.1 Modelos de Construcción [Bec99; McC04]

Se han creado numerosos modelos para el desarrollo del software, algunos de los cuales ponen más énfasis en la construcción que otros.

Algunos modelos son más lineales que otros desde el punto de vista de la construcción tales como los modelos en cascada y los del ciclo de vida de entregas por etapas. Estos modelos tratan la construcción como una actividad que sucede sólo después de que se haya completado un significativo trabajo con los prerrequisitos –incluyendo un trabajo detallado sobre los requisitos, un extensivo trabajo sobre el diseño y una planificación detallada. Los enfoques más lineales tienden a poner el énfasis en las actividades que preceden a la construcción (requisitos y diseño), y tienden a crear separaciones más marcadas entre las actividades. En estos modelos, la codificación sería el punto de mayor énfasis de la construcción.

Otros modelos son más iterativos, tales como el prototipado evolucionista, Programación Extrema y "Scrum". Estos enfoques tienden a tratar la construcción como una actividad que ocurre en estos momentos con otras actividades de desarrollo del software incluyendo los requisitos, el diseño y la planificación, o que se traslapa con ellas. Estos enfoques tienden a mezclar el diseño, la codificación y las actividades de pruebas, y con frecuencia tratan la combinación de actividades como una construcción.

Por consiguiente, lo que está considerado como "construcción" depende hasta cierto grado del modelo de ciclo de vida utilizado.

2.2 Planificación de la Construcción

[Bec99; McC04]

La elección de un método de construcción es un aspecto clave de la planificación de la actividad de construcción. La elección de un método de construcción afecta hasta dónde se realizan los prerrequisitos de construcción, el orden en el que se realizan, y el grado hasta el que se espera que se completen antes de que comience el trabajo de construcción.

El modo como se afronta la construcción afecta a la habilidad del proyecto para reducir la complejidad, anticipar cambios y construir para verificar. Cada uno de estos objetivos puede también afrontarse en los niveles de proceso, requisitos y diseño –pero también estarán influenciados por la elección de un método de construcción.

La planificación de la construcción también define el orden en el que se crean e integran, según el método elegido, los componentes, los procesos de gestión de la calidad del software, la asignación de tareas a ingenieros del software específicos y el resto de las tareas.

2.3 Medición de la Construcción

[McC04]

Se pueden medir numerosas actividades de construcción y artefactos, incluidos el código desarrollado, el código modificado, el código reutilizado, el código destruido, la complejidad del código, las estadísticas de la inspección del código, las tasas de rectificación de errores y de identificación de errores, y los horarios. Estas mediciones pueden ser útiles para propósitos de gestión de la construcción, asegurando la calidad durante la construcción, mejorando los procesos de construcción, amén de otras razones.

3. Consideraciones Prácticas

La construcción es una actividad en la cual el software se las tiene que ver con restricciones arbitrarias y caóticas del mundo real, y hacer exactamente lo que piden. Gracias a su proximidad a las restricciones del mundo real, la construcción está guiada por consideraciones prácticas más que otras KAs, y la ingeniería del software es quizás el área de construcción más artesanal.

3.1 Diseño de la Construcción

[Bec99; Ben00; Hun00; IEEE12207-95; Mag93; McC04]

Algunos proyectos asignan una mayor actividad de diseño a la construcción; otros a una fase que se centra explícitamente en el diseño. Independientemente de su asignación exacta, en el nivel de construcción también se trabaja algo el diseño detallado y ese trabajo de diseño tiende a estar dictaminado por restricciones inamovibles impuestas por un problema del mundo real que está siendo afrontado por el software.

Así como los obreros de una construcción que construyen una estructura física tienen que realizar modificaciones a pequeña escala para cubrir huecos no previstos en los planes del constructor, los obreros de la construcción del software tendrán que hacer modificaciones en una mayor o menor escala para revelar los detalles del diseño de software durante la construcción.

Los detalles de la actividad de diseño a nivel de la construcción son esencialmente los mismos que se describen en el KA del Diseño del Software, pero se aplican en una escala inferior.

3.2 Lenguajes de Construcción

[Hun00; McC04]

Los lenguajes de construcción incluyen todos los tipos de comunicación mediante los cuales un humano puede especificar una solución ejecutable para un problema de un ordenador.

El tipo más simple de lenguaje de construcción es un *lenguaje de configuración* en el que los ingenieros del software eligen de entre un conjunto limitado de opciones predefinidas para crear nuevas o típicas

instalaciones del software. Los archivos de configuración basados en texto utilizados tanto en los sistemas operativos de Windows como de Unix son ejemplos de esto, y otro ejemplo son las listas de selección en forma de menú de algunos generadores de programas.

Los *lenguajes de herramientas* se utilizan para construir aplicaciones partiendo de las herramientas (conjuntos integrados de partes reutilizables específicas de las aplicaciones), y son más complejos que los lenguajes de configuración. Los lenguajes de herramientas pueden definirse explícitamente como lenguajes de programación de aplicaciones (por ejemplo, scripts), o pueden simplemente estar implícitos en el conjunto de interfaces de las herramientas.

Los *lenguajes de programación* son el tipo más flexible de lenguaje de construcción. También son los que menos información contienen acerca de las áreas específicas de la aplicación y los procesos de desarrollo, y por tanto requieren el mayor entrenamiento y destreza posibles para utilizarlo con eficacia.

Existen tres tipos generales de notación utilizados para los lenguajes de programación, a saber:

- Lingüísticos
- Formales
- Visuales

Las notaciones lingüísticas se distinguen en particular por la utilización de cadenas de texto del tipo palabra para representar construcciones complejas de software, y por la combinación en patrones de tales cadenas del tipo palabra que tienen una sintaxis del tipo sentencia. Utilizadas adecuadamente, cada una de estas cadenas debería tener una fuerte connotación ofreciendo un entendimiento intuitivo inmediato de lo que sucedería cuando se ejecutara la construcción del software subyacente.

Las notaciones formales se apoyan menos en los significados de las palabras y cadenas de texto intuitivos y de todos los días, y más en las definiciones respaldadas por definiciones precisas, sin ambigüedad, y formales (o matemáticas). Las notaciones de construcción formal y los métodos formales están en el corazón de la mayoría de las formas de programación de sistemas, donde la precisión, el comportamiento del tiempo, y la capacidad de realizar pruebas son más importantes que la facilidad de mapeo a un lenguaje natural. Las construcciones formales también utilizan modos de combinar símbolos definidos con precisión que evitan la ambigüedad de muchas construcciones del lenguaje natural.

Las notaciones visuales se apoyan bastante poco en las notaciones orientadas al texto tanto de la construcción lingüística como de la formal, y en cambio sí se apoyan en una interpretación visual directa y en la colocación de las entidades visuales que representan al software subyacente. La construcción visual tiende a estar un tanto limitada por la dificultad de hacer declaraciones "complejas" utilizando sólo el movimiento de entidades visuales en un despliegue. Sin embargo, también puede convertirse en un arma poderosa en los casos en donde la principal tarea de programación es simplemente construir y "ajustar" una interfaz visual a un programa, cuyo comportamiento detallado ha sido definido anteriormente.

3.3 Codificación

[Ben00; IEEE12207-95; McC04]

Las consideraciones siguientes se aplican a la actividad de construcción del código del software:

- Técnicas para crear código fuente comprensible, que incluye la asignación de nombres y el esquema del código fuente
- Utilización de clases, tipos enumerados, variables, constantes predefinidas, y otras entidades similares
- Utilización de estructuras de control
- Tratamiento de las condiciones de error –tanto lo errores planeados como las excepciones (la entrada de datos malos, por ejemplo)
- Prevención de brechas en la seguridad a nivel de código (el búfer o el índice de la matriz se desborda, por ejemplo)
- Utilización de recursos por medio del uso de mecanismos de exclusión y disciplina en el acceso serial a recursos reutilizables (incluyendo threads o bloqueos de bases de datos)
- Organización del código fuente (en declaraciones, rutinas, clases, paquetes u otras estructuras)
- Documentación del código
- Puesta a punto del código

3.4 Pruebas de Construcción

[Bec99; Hun00; Mag93; McC04]

Construir implica dos tipos de pruebas, que por lo general las realiza el mismo ingeniero del software que escribió el código:

- Pruebas unitarias
- Pruebas de integración

El propósito de las pruebas de construcción es reducir la brecha entre el tiempo en el que se introducen fallos en el código y el tiempo en el que se detectan esos fallos. En algunos casos, las pruebas de construcción se llevan a cabo después de la escritura del código. En otros casos, se pueden elaborar casos de pruebas antes de que se escriba el código.

Es típico de las pruebas de construcción el incluir un subconjunto de tipos de pruebas, que se describen en el KA de Pruebas del Software. Por ejemplo, no es típico de las pruebas de construcción el incluir las pruebas del sistema, las pruebas alfa, las pruebas beta, las pruebas de estrés, las pruebas de construcción, las pruebas de posibilidad de uso, u otros tipos de pruebas más especializadas.

Se han publicado dos estándares sobre dicho tema: IEEE Std 829-1998, *IEEE Standard for Software Test Documentation* and IEEE Std 1008-1987, *IEEE Standard for Software Unit Testing*.

Se pueden ver también los sub-temas correspondientes en el KA de Pruebas del Software: 2.1.1 Pruebas

Unitarias y 2.1.2 Pruebas de Integración para un material de referencia más especializado.

3.5 Reutilización

[IEEE1517-99; Som05].

Tal y como se afirma en la introducción del (IEEE1517-99):

"El implementar la utilización del software conlleva algo más que crear y utilizar librerías de recursos. Requiere formalizar la práctica de la reutilización por medio de la integración de procesos y actividades de reutilización en el ciclo de vida del software." Sin embargo, la reutilización tiene suficiente importancia en la construcción del software como para dedicarle aquí un tema.

Las tareas relacionadas con la reutilización en la construcción del software durante su codificación y pruebas son:

- La selección de unidades, bases de datos, procedimientos de pruebas o datos de pruebas reutilizables.
- La evaluación de la posibilidad de reutilización del código o de las pruebas.
- Comunicar la información sobre reutilización realizada en el código nuevo, los procedimientos de pruebas o los datos de pruebas.

3.6 Calidad de la Construcción

[Bec99; Hun00; IEEE12207-95; Mag93; McC04]

Existen numerosas técnicas para garantizar la calidad del código mientras está siendo elaborado. Las técnicas más importantes utilizadas para la construcción incluyen:

- Las pruebas unitarias y las pruebas de integración (tal y como se describen en el punto 3.4 *Pruebas de Construcción*)
- El desarrollo de primero-haz-pruebas (ver también el KA de las Pruebas del Software, punto 2.2 *Objetivos de las Pruebas*)
- El código paso a paso
- Utilización de aserciones
- Depuración
- Revisiones Técnicas (ver también el KA de la Calidad del Software, sub-punto 2.3.2 *Revisiones Técnicas*)
- Análisis estático (IEEE1028) (ver también el KA de la Calidad del Software, punto 2.3 *Revisiones y Auditorias*)

La técnica o técnicas específicas elegidas dependen de la naturaleza del software que se está construyendo, así como del conjunto de habilidades de los ingenieros del software que llevan a cabo la construcción.

Las actividades de calidad de la construcción se distinguen de las otras actividades de calidad por su enfoque. Las actividades de calidad de la construcción se centran en el código y en los artefactos que están estrechamente relacionados con el código: diseños en pequeña escala –en oposición a otros artefactos que están menos directamente ligados al código, tales como requisitos, diseños de alto nivel y planes.

3.7 Integración

[Bec99; IEEE12207-95; McC04]

Una actividad clave durante la construcción es la integración de rutinas, clases, componentes y subsistemas construidos por separado. Además, un sistema particular del software podría necesitar ser integrado con otros sistemas de software o de hardware.

Los intereses relacionados con la integración de la construcción incluyen planificar la secuencia en la que se integrarán los componentes, crear andamiajes que soporten versiones provisionales del software, determinar el grado de pruebas y la calidad del trabajo realizado sobre los componentes antes de que sean integrados, y determinar los puntos en el proyecto en los que se prueban las versiones provisionales del software.

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Bec99]	[Ben00]	[Hun00]	[IEEE 1517]	[IEEE 12207.0]	[Ker99]	[Mag93]	[Mcc04]	[Som05]
1. Fundamentos de Construcción de Software									
1.1 Minimizar la Complejidad	c17	c2, c3	c7,c8			c2, c3	c6	c2, c3, c7-c9, c24, c27, c28, c31, c32-c34	
1.2 Anticipación a Cambios		c11,c13-c14				c2, c9		c3-c5, c24, c31, c32, c34	
1.3 Construir para verificar		c4	c21, c23, c34, c43			c1, c5, c6	c2,c3, c5,c7	c8, c20-c23, c31-c34	
1.4 Estándares de Construcción					X			c4	
2. Gestión de la Construcción									
2.1 Modelos de Construcción	c10							c2, c3, c27, c29	
2.2 Plan de Construcción	c12, c15, c21							c3, c4,c21, c27-c29	
2.3 Métricas de la construcción								c25, c28	
3. Consideraciones Prácticas									
3.1 Diseño de la Construcción	c17	c18-c10, p175-6	c33		X		c6	c3, c5, c24	
3.2 Lenguajes de Construcción			c12, c14-c20					C4	
3.3 Codificación		c6-c10			X			c5-c19, c25-c26	
3.4 Pruebas de Construcción	c18		c34, c43		X		c4	c22, c23	
3.5 Reusabilidad				X					c14
3.6 Calidad de Construcción	c18		c18		X		c4, c6, c7	c8, c20-c25	
3.7 Integración	c16				X			c29	

REFERENCIAS RECOMENDADAS PARA LA CONSTRUCCIÓN DE SOFTWARE

[Bec99] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999, Chap. 10, 12, 15, 16-18, 21.

[Ben00a] J. Bentley, *Programming Pearls*, second ed., Addison-Wesley, 2000, Chap. 2-4, 6-11, 13, 14, pp. 175-176.

[Hun00] A. Hunt and D. Thomas, *The Pragmatic Programmer*, Addison-Wesley, 2000, Chap. 7, 8 12, 14-21, 23, 33, 34, 36-40, 42, 43.

[IEEE1517-99] IEEE Std 1517-1999, *IEEE Standard for Information Technology-Software Life Cycle Processes-Reuse Processes*, IEEE, 1999.

[IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std.ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

[Ker99a] B.W. Kernighan and R. Pike, *The Practice of Programming*, Addison-Wesley, 1999, Chap. 2, 3, 5, 6, 9.

[Mag93] S. Maguire, *Writing Solid Code: Microsoft's Techniques for Developing Bug-Free C Software*, Microsoft Press, 1993, Chap. 2-7.

[McC04] S. McConnell, *Code Complete: A PracticalHandbook of Software Construction*, Microsoft Press, second ed., 2004.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005.

APÉNDICE A. LISTA DE LECTURAS ADICIONALES.

(Bar98) T.T. Barker, *Writing Software Documentation: A Task-Oriented Approach*, Allyn & Bacon, 1998.
(Bec02) K. Beck, *Test-Driven Development: By Example*, Addison-Wesley, 2002.
(Fow99) M. Fowler and al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
(How02) M. Howard and D.C. Leblanc, *Writing Secure Code*, Microsoft Press, 2002.
(Hum97b) W.S. Humphrey, *Introduction to the Personal Software Process*, Addison-Wesley, 1997.
(Mey97) B. Meyer, *Object-Oriented Software Construction*, second ed., Prentice Hall, 1997, Chap. 6, 10, 11.
(Set96) R. Sethi, *Programming Languages: Concepts & Constructs*, second ed., Addison-Wesley, 1996, Parts II-V.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE829-98) IEEE Std 829-1998, IEEE Standard for Software Test Documentation, IEEE, 1998.
(IEEE1008-87) IEEE Std 1008-1987 (R2003), IEEE Standard for Software Unit Testing, IEEE, 1987.
(IEEE1028-97) IEEE Std 1028-1997 (R2002), IEEE Standard for Software Reviews, IEEE, 1997.
(IEEE1517-99) IEEE Std 1517-1999, IEEE Standard for Information Technology-Software Life Cycle Processes-Reuse Processes, IEEE, 1999.
(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, Industry Implementation of Int. Std.ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes, IEEE, 1996.

CAPÍTULO 5
PRUEBAS DEL SOFTWARE

ACRÓNIMOS

SRET	Pruebas Orientadas a la Confiabilidad del Software

INTRODUCCIÓN

Hacer pruebas es una actividad que tiene el objetivo de evaluar y mejorar la calidad del producto, identificando defectos y problemas.

Las pruebas del software consisten en verificar el comportamiento de un programa *dinámicamente* a través de un grupo *finito* de casos de prueba, debidamente *seleccionados* del, típicamente, ámbito de ejecuciones infinito, en relación al comportamiento *esperado*. En la definición anterior las palabras en itálica se corresponden con aspectos esenciales en la identificación del "Área de Conocimiento de las Pruebas del Software". En particular:

- *Dinámicamente*: Este término significa que hacer pruebas siempre supone ejecutar el programa con entrada de datos (valorados). Para precisar, es preciso afirmar que la entrada de valores no es siempre suficiente para definir una prueba, dado que un sistema complejo y no determinista podría tener diferentes comportamientos con las misma entrada de datos, dependiendo del estado en el que se encuentre. En cualquier caso, en este KA, mantendremos el término de "entrada de datos", asumiendo la convención de que el término incluye un estado del sistema específico, en los casos en que sea necesario. Existen otras técnicas complementarias a las pruebas, aunque diferentes, descritas en el KA sobre la Calidad del Software.
- *Finito*: Incluso en programas sencillos, teóricamente podría haber tantas pruebas que realizar, que hacer pruebas exhaustivas podría llevar meses o años. Esta es la razón por la que en la práctica el grupo completo de pruebas se podría considerar infinito. Hacer pruebas siempre supone un compromiso entre recursos y calendarios de trabajo limitados, por un lado, y necesidades inherentes de pruebas ilimitadas, por otro.
- *Seleccionados*: La diferencia esencial entre las distintas técnicas de pruebas propuestas se encuentra en cómo se escoge el conjunto de pruebas. Los ingenieros informáticos deben ser conscientes de que criterios de selección distintos pueden producir grados de efectividad muy diferentes. La forma de identificar el criterio de selección de pruebas más apropiado para un conjunto de condiciones particulares es un problema complejo; en la práctica se usa la experiencia en el diseño de pruebas y técnicas de análisis de riesgo.
- *Esperado*: Debería se posible, aunque a veces no sea fácil, decidir si el resultado observado de la ejecución de un programa es aceptable o no, porque si no el esfuerzo de realizar las pruebas sería inútil. El comportamiento observado se puede comprobar con los resultados esperados por el usuario (normalmente conocido como pruebas de validación), con las especificaciones (pruebas de verificación), o, finalmente, con el comportamiento anticipado de requerimientos implícitos o expectativas razonables. Vea más detalles en el KA de Requerimientos del Software, punto 6.4 *Pruebas de Aceptación*.

La apreciación de las pruebas del software ha evolucionado hacia una forma más constructiva. Ya no se asume que realizar pruebas es una tarea que empieza solamente cuando la fase de programación se ha completado, y que tiene el único propósito de detectar errores. Las pruebas del software se ven ahora como una actividad que debería estar presente durante todo el proceso de desarrollo y mantenimiento y es en sí misma una parte importante de la construcción del producto. Es más, la planificación de las pruebas debería empezar en las primeras etapas del proceso de requisitos, mientras que los planes y procedimientos de pruebas deberían desarrollare y posiblemente refinarse sistemáticamente según avanza el desarrollo. La planificación de las pruebas y las propias actividades de diseño constituyen una información muy útil que ayuda a los diseñadores de software a identificar debilidades potenciales (tales como elementos del diseño que han pasado desapercibidos, contradicciones de diseño, u omisiones o ambigüedades en la documentación).

En la actualidad se considera que la prevención es la actitud adecuada en lo que respecta a la calidad: obviamente es mejor evitar problemas que solucionarlos. Realizar pruebas debe verse como un medio para verificar, no sólo si la prevención ha sido efectiva, si no para identificar fallos en aquellos casos en los que, por alguna razón, no lo ha sido. Aunque quizás sea obvio, vale la pena reconocer que, incluso después de una campaña de pruebas extensiva, el software aún podría contener errores. Las acciones de mantenimiento correctivas proporcionan la solución a errores en el software después de que éste ha sido entregado. El KA

del Mantenimiento del Software aborda los temas relacionados con el mantenimiento del software.

En el KA del Mantenimiento del Software (véase punto 3.3 *Técnicas de Gestión de Calidad del Software*), las técnicas de gestión de la calidad del software se dividen entre técnicas *estáticas* (sin ejecución de código) y técnicas *dinámicas* (con ejecución de código). Ambas categorías son útiles. Este KA se centra en técnicas dinámicas.

Las pruebas del software también están relacionadas con la construcción del software (véase la sección 3.4 *Construcción de Pruebas*). Las pruebas de unidad y de integración están íntimamente relacionadas con la construcción del software, si no son parte de la misma.

DIVISIÓN DE TEMAS

La descomposición de temas en el KA de las Pruebas del Software se muestra en la Figura 1.

La primera subárea describe los *Fundamentos de las Pruebas del Software*. Cubre las definiciones básicas del área de pruebas del software, la terminología básica y los términos clave, así como las relaciones con otras actividades.

La segunda subárea, *Niveles de Pruebas*, está formada por dos puntos ortogonales: el primero (2.1) enumera los niveles en que tradicionalmente se subdividen las pruebas para software grande, mientras que el segundo (2.2) considera las pruebas para situaciones o propiedades específicas y se conoce como *objetivos de las pruebas*. No todos los tipos de pruebas se pueden aplicar a todos los productos de software, tampoco se han enumerado todos los tipos posibles.

El objeto y los objetivos de las pruebas determinan la forma en que un grupo de pruebas se identifica, en lo que se refiere a su consistencia – *cuántas pruebas son suficientes para conseguir el objetivo especificado* – y su composición – *qué casos de prueba se deberían seleccionar para conseguir el objetivo especificado* (aunque normalmente la parte "para conseguir el objetivo especificado" es implícita y sólo se usa la primera parte de las dos frases anteriores). Los criterios para responder a la primera cuestión se denominan criterios de *idoneidad las pruebas*, mientras que los que se refieren a la segunda cuestión se denominan criterios de *selección de las pruebas*.

En las últimas décadas se han desarrollado varias *Técnicas de Pruebas* y aún se están proponiendo nuevas técnicas. El conjunto de pruebas comúnmente aceptadas están enumeradas en la subárea 3.
Las *Mediciones de Pruebas* se enumeran en la subárea 4.
Finalmente, los aspectos relacionados con el *Proceso de las Pruebas* están enumeradas en la subárea 5

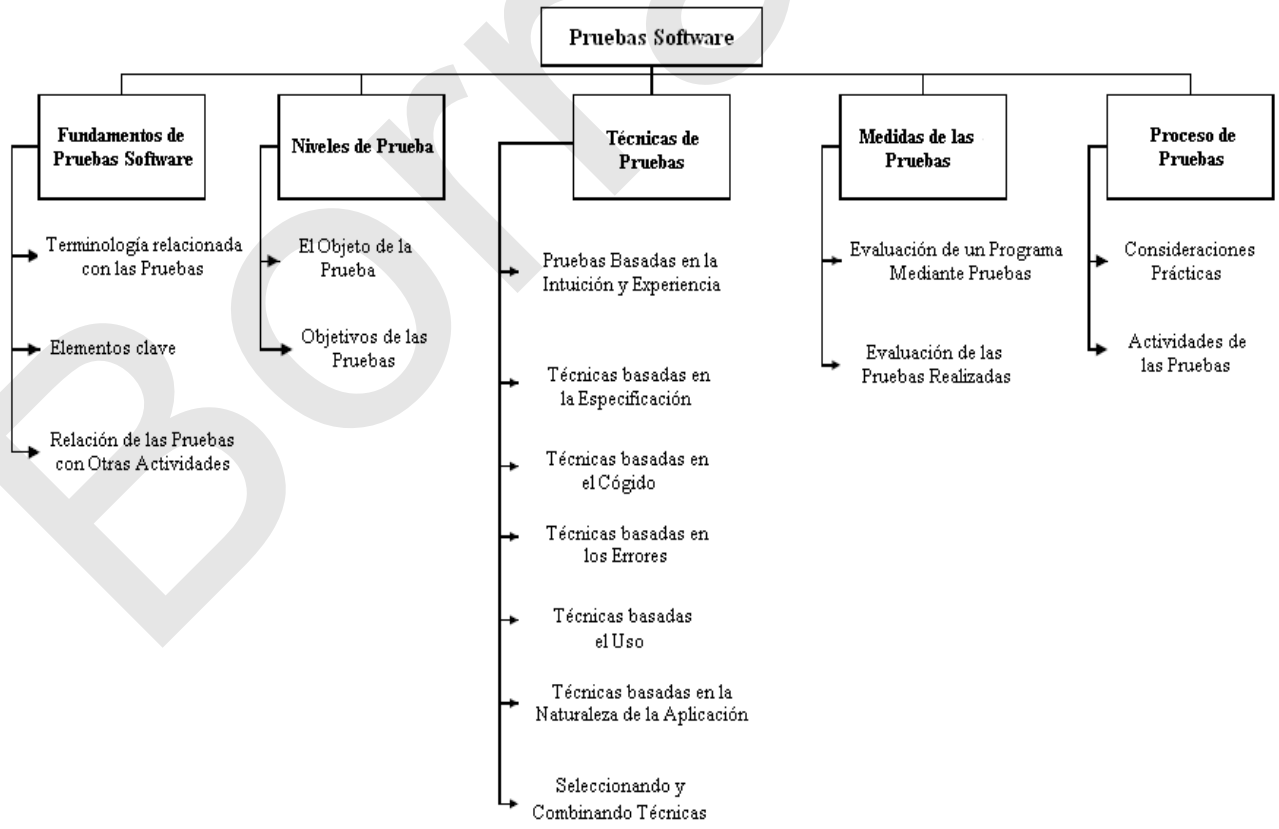


Figura 1 División de los temas para el KA de las Pruebas del Software

1. Fundamentos de las Pruebas del Software

1.1 Terminología relacionada con las pruebas

1.1.1 Definiciones de pruebas y terminología relacionada
[Bei90:c1; Jor02:c2; Lyu96:c2s2.2] (IEEE610.12-90)

Vea una introducción detallada del KA de las Pruebas del Software en las referencias recomendadas.

1.1.2 Errores Vs. Fallos [Jor02:c2; Lyu96:c2s2.2; Per95:c1; Pfl01:c8] (IEEE610.12-90; IEEE982.1-88)

En la bibliografía sobre Ingeniería del Software se usan diversos términos para describir un funcionamiento incorrecto, particularmente *falta, error, fallo* y otros términos. Esta terminología se define detalladamente en el estándar IEEE 610.12-1990, *Standard Glossary of Software Engineering Terminology* (IEEE610-90) y también se discute en el KA de la Calidad del Software. Es esencial distinguir claramente entre la causa de un funcionamiento incorrecto, en cuyo caso se usan términos como *error* y *defecto*, y los efectos no deseados observados en los servicios proporcionados por un sistema, que se llamarán *fallos*. Hacer pruebas puede descubrir fallos, pero es el error el que se puede, y se debe, eliminar.

En cualquier caso, debería aceptarse que no es siempre posible identificar unívocamente las causas de un fallo. No existe ningún criterio teórico que pueda usarse para determinar qué error produce el fallo observado. Podría decirse que hay que arreglar el error para eliminar el problema, pero otros cambios también podrían funcionar. Para evitar ambigüedades, algunos autores prefieren hablar de *entradas que causan fallos* (Fra98) en vez de errores – o lo que es lo mismo, aquellos grupos de entradas de datos que hacen que el fallo aparezca.

1.2 Cuestiones clave

1.2.1 Criterios de selección de pruebas/Criterios de idoneidad de pruebas (o finalización de pruebas)
[Pfl01:c8s7.3; Zhu97:s1.1] (Wey83; Wey91; Zhu97)

Un criterio de selección de pruebas es un medio para decidir cuáles deben ser los casos de prueba adecuados. Un criterio de selección se puede usar para seleccionar casos de pruebas o para comprobar si el grupo de casos de prueba es apropiado – o sea, para decidir si se puede terminar de hacer pruebas. Véase el apartado *Finalización* de la sección 5.1 *Consideraciones prácticas*.

1.2.2 Efectividad de las pruebas/Objetivos para las pruebas
[Bei90:c1s1.4; Per95:c21] (Fra98)

Realizar pruebas consiste en observar un conjunto de ejecuciones del programa. Hay diferentes objetivos que nos pueden guiar en la selección del conjunto de pruebas: la efectividad del grupo de pruebas sólo se puede evaluar en función del objetivo seleccionado.

1.2.3 Realizar pruebas para la identificación de defectos
[Bei90:c1; Kan99:c1]

Cuando realizamos pruebas para la identificación de defectos, una prueba es satisfactoria si produce un error en el sistema. Es éste un enfoque completamente diferente al de realizar pruebas para demostrar que el software satisface las especificaciones u otro conjunto de propiedades deseadas, en cuyo caso una prueba satisfactoria es aquella en la que no se observan errores (al menos significativos).

1.2.4 El problema del oráculo
[Bei90:c1] (Ber96, Wey83)

Un oráculo es cualquier agente (humano o mecánico) que decide si un programa se comporta correctamente durante una prueba y consecuentemente produce un veredicto de "superada" o "fallada". Hay varios tipos diferentes de oráculos, y la automatización de oráculos puede ser muy difícil y cara.

1.2.5 Limitaciones teóricas y prácticas de las pruebas [Kan99:c2] (How76)

La teoría de pruebas advierte en contra de un nivel injustificado de confianza en una serie de pruebas superadas. Desafortunadamente, la mayor parte de los resultados establecidos en la teoría de pruebas son negativos, en el sentido de que establecen aquello que la prueba no puede conseguir, en vez de lo que consiguió. La más famosa cita a este respecto es el aforismo de Dijkstra que dice "las pruebas de un programa se pueden usar para mostrar las presencia de errores, pero nunca para demostrar su ausencia". La razón obvia es que realizar un grupo completo de pruebas no es posible en el software real. Como consecuencia, las pruebas deben dirigirse en función de los riesgos y por tanto pueden verse como una estrategia de gestión de riesgo.

1.2.6 El problema de los caminos no alcanzables
[Bei90:c3]

Los caminos no alcanzables son aquellos caminos de control que no pueden ejecutarse para ninguna entrada de datos. Son un problema importante en las pruebas orientadas por caminos y particularmente en las derivaciones automáticas de entradas de pruebas que se emplean en las técnicas de pruebas basadas en código.

1.2.7 Posibilidad de hacer pruebas
[Bei90:c3, c13] (Bac90; Ber96a; Voa95)

El término "posibilidad de hacer pruebas" tiene dos significados relacionados pero diferentes: por un lado, se refiere al grado de facilidad del software para satisfacer un determinado criterio de cobertura de

pruebas, como se describe en (Bac90); por otro, se define como la probabilidad, posiblemente cualificada estadísticamente, de que los errores del software queden expuestos durante las pruebas, si es erróneo, tal y como se describe en (Voa95, Ber96a). Ambos significados son importantes.

1.3 Relación de las pruebas con otras actividades

Las pruebas del software, aunque diferentes, están relacionadas con las técnicas de gestión de la calidad del software estático, las pruebas de validez del software, la depuración y la programación. Sin embargo, es útil considerar las pruebas desde el punto de vista del analista de calidad del software o certificador.

- Pruebas vs. técnicas de Gestión de Calidad del Software Estático. Véase también el KA de la Calidad del Software, punto 2. *Proceso de Gestión de la Calidad del Software*. [Bei90:c1; Per95:c17] (IEEE1008-87)
- Pruebas vs. Pruebas de Validez y Verificación Formal [Bei90:c1s5; Pfl01:c8].
- Pruebas vs. Depuración. Véase también el KA de la Construcción del Software, punto 3.4 *Pruebas de la construcción* [Bei90:c1s2.1] (IEEE1008-87).
- Pruebas vs. Programación. Véase también el KA de la Construcción del Software, punto 3.4 *Pruebas de la construcción* [Bei90:c1s2.1] (IEEE1008-87).
- Pruebas y Certificación (Wak99).

2 Niveles de Pruebas

2.1 El objeto de la prueba

Las pruebas del software se realizan normalmente a diferentes *niveles* durante los procesos de desarrollo y mantenimiento. Esto significa que el objeto de las pruebas puede cambiar: un módulo, un grupo de dichos módulos (relacionados por propósito, uso, comportamiento, o estructura), o un sistema completo. [Bei90:c1; Jor02:c12;Pfl01:c8] Conceptualmente se pueden distinguir tres grandes niveles de pruebas, llamadas de Unidad, de Integración y del Sistema. No hay un modelo de proceso implícito, ni se asume que ninguno de estos tres niveles tiene mayor importancia que los otros dos.

2.1.1 Pruebas de Unidad
[Bei90:c1; Per95:c17; Pfl01:c8s7.3] (IEEE1008-87)

Las pruebas de unidad verifican el funcionamiento aislado de partes del software que se pueden probar independientemente. Dependiendo del contexto, estas partes podrían ser subprogramas individuales o un componente más grande formado por unidades muy relacionadas. Hay una definición más precisa de prueba de unidad en el estándar IEEE de pruebas de unidad del software (IEEE1008-87), que también describe un método integrado para realizar y documentar pruebas de unidad sistemáticamente. Normalmente, las pruebas de unidad se realizan con acceso al código fuente y con el soporte de herramientas de depuración, pudiendo implicar a los programadores que escribieron el código

2.1.2 Pruebas de Integración
[Jor02:c13, 14; Pfl01:c8s7.4]

Una prueba de integración es el proceso de verificar la interacción entre componentes de software. Estrategias clásicas de integración, como arriba-abajo o abajo-arriba, se usan, tradicionalmente, con software estructurado jerárquicamente.

Las estrategias modernas de integración están dirigidas por la arquitectura, lo que supone integrar los componentes de software o subsistemas basándose en caminos de funcionalidad identificada. Las pruebas de integración son una actividad continua, que sucede en cada fase en que los ingenieros de software tienen que hacer abstracciones de las perspectivas de bajo nivel y concentrarse en las perspectivas del nivel que están integrando. Con la excepción de software sencillo y pequeño, las estrategias de pruebas de integración sistemáticas e incrementales son preferibles a probar todos los componentes juntos al final, lo que se conoce (de forma gráfica), como pruebas en "big bang".

2.1.3 Pruebas del sistema
[Jor02:c15; Pfl01:c9]

Las pruebas del sistema se ocupan del comportamiento de un sistema completo. La mayoría de los fallos funcionales deberían haber sido identificados antes, durante las fases de pruebas de unidad y pruebas de integración. Las pruebas del sistema se consideran normalmente como las apropiadas para comparar el sistema con los requisitos no funcionales del sistema, como seguridad, velocidad, exactitud y confiabilidad. Las interconexiones externas con otras aplicaciones, utilidades, dispositivos hardware o con el sistema operativo, también se evalúan en este nivel. Véase el KA de Requisitos del Software para más información acerca de requisitos funcionales y no funcionales.

2.2 *Objetivos de las pruebas*
[Per95:c8; Pfl01:c9s8.3]

Las pruebas se realizan en relación a conseguir un determinado objetivo, que se ha definido más o menos explícitamente y con diversos niveles de precisión. Definir el objetivo, en términos precisos y cuantitativos, permite establecer controles en el proceso de las pruebas.

Las pruebas se pueden realizar para verificar propiedades distintas. Se pueden asignar casos de prueba para comprobar que las especificaciones funcionales se han implementado correctamente, a lo que la literatura se refiere como pruebas de *conformidad*, pruebas de *corrección* o pruebas de *funcionalidad*. Sin embargo, también se pueden hacer pruebas a otras muchas propiedades no funcionales, como rendimiento, confiabilidad y facilidad de uso, entre otras muchas.

Otros objetivos importantes de las pruebas incluyen (aunque no se limitan a) mediciones de confiabilidad, evaluación de la facilidad de uso y aceptación, para los cuales se utilizarían métodos diferentes. Se debe tener en cuenta que los objetivos de las pruebas varían con el objeto de las pruebas; en general, propósitos diferentes son tratados con diferentes niveles de pruebas.

Las referencias recomendadas para este punto describen el conjunto de objetivos de pruebas potenciales. Los puntos enumerados seguidamente son los que se citan más frecuentemente en la literatura. Téngase en cuenta que algunos tipos de pruebas son más apropiados para paquetes de software hechos a medida, pruebas de *instalación*, por ejemplo; mientras otros son más apropiados para productos más genéricos, como pruebas *beta*.

2.2.1 Pruebas de aceptación/calificación
[Per95:c10; Pfl01:c9s8.5] (IEEE12207.0-96:s5.3.9)

Las pruebas de aceptación comparan el comportamiento del sistema con los requisitos del cliente, sea cual sea la forma en que éstos se hayan expresado. El cliente realiza, o especifica, tareas típicas para comprobar que se satisfacen sus requisitos o que la organización los ha identificado para el mercado al que se destina el software. Esta actividad de pruebas puede incluir o no a los desarrolladores del sistema.

2.2.2 Pruebas de instalación
[Per95:c9; Pfl01:c9s8.6]

Normalmente, cuando las pruebas de aceptación han terminado, el software se puede comprobar una vez instalado en el entorno final. Las pruebas de instalación se pueden ver como pruebas del sistema realizadas en relación con los requisitos de la configuración de hardware. Los procedimientos para la instalación también se podrían verificar.

2.2.3 Pruebas alfa y beta
[Kan99:c13]

A veces, antes de poner el software en distribución, éste se proporciona a un grupo representativo de usuarios potenciales para que puedan usarlo en pruebas en las instalaciones del desarrollador (pruebas *alpha*) o externamente (pruebas *beta*). Dichos usuarios notifican problemas con el producto. Normalmente, el uso de versiones alfa y beta sucede en entornos no controlados y no siempre se le hace referencia en los planes de pruebas.

2.2.4 Pruebas de conformidad/pruebas funcionales/pruebas de corrección
[Kan99:c7; Per95:c8] (Wak99)

Las pruebas de conformidad tienen el objetivo de verificar si el comportamiento del software se corresponde con las especificaciones.

2.2.5 Materialización de la confiabilidad y evaluación [Lyu96:c7; Pfl01:c9s.8.4] (Pos96)

Las pruebas, al ayudar a identificar errores, son un medio para mejorar la confiabilidad. Por contraste, generando casos de prueba aleatorios siguiendo el perfil de operaciones, se pueden derivar aproximaciones estadísticas de confiabilidad. Cuando se usan modelos que potencian la confiabilidad, ambos objetivos se pueden alcanzar al mismo tiempo (véase también el punto 4.1.4 Pruebas de ejecución, evaluación de la confiabilidad)

2.2.6 Pruebas de regresión
[Kan99:c7; Per95:c11, c12; Pfl01:c9s8.1] (Rot96)

Según (IEEE610.12-90), las pruebas de regresión son "pruebas selectivas que se repiten en un componente para verificar que los cambios no han producido efectos indeseados..." En la práctica, la idea es demostrar que cierto software que previamente pasó un conjunto de pruebas, aún las pasa. Beizer (Bei90) las define como cualquier repetición de pruebas que tiene como objetivo demostrar que el comportamiento del software no ha cambiado, excepto en aquellos aspectos en que se haya requerido así. Por supuesto se tiene que llegar a un compromiso entre realizar pruebas de regresión cada vez que se hace un cambio y los medios de que se dispone para realizar las pruebas.

Las pruebas de regresión se pueden realizar en cada uno de los niveles de pruebas descritos en el punto 2.1 *El objeto de la prueba* y son válidas tanto para pruebas funcionales como no funcionales.

2.2.7 Pruebas de rendimiento
[Per95:c17; Pfl01:c9s8.3] (Wak99)

Estas pruebas tienen el objetivo de verificar que el software alcanza los requerimientos de rendimiento especificados, particularmente los de capacidad y tiempo de respuesta. Un tipo particular de pruebas de rendimiento son las pruebas de volumen (Per95:p185, p487; Pfl01:p401), en los que las limitaciones internas del programa o sistema se ponen a prueba.

2.2.8 Pruebas de desgaste
[Per95:c17; Pfl01:c9s8.3]

Las pruebas de desgaste hacen funcionar el software a la máxima capacidad para la que fue diseñado, y por encima de ella.

2.2.9 Pruebas de continuidad.

Un grupo de pruebas se ejecuta en dos versiones diferentes de un producto software y los resultados se comparan.

2.2.10 Pruebas de recuperación
[Per95:c17; Pfl01:c9s8.3]

El objetivo de las pruebas de recuperación es verificar la capacidad del software para reiniciarse después de un "desastre".

2.2.11 Pruebas de configuración
[Kan99:c8; Pfl01:c9s8.3]

En los casos en los que el software se construye para dar servicio a distintos usuarios, las pruebas de configuración analizan el software en las diferentes configuraciones especificadas.

2.2.12 Pruebas de facilidad de uso
[Per95:c8; Pfl01:c9s8.3]

Este proceso evalúa lo fácil que le resulta usar y aprender a usar el software al usuario, incluyendo la documentación del usuario, la efectividad de las funciones del software para soportar las tareas de usuario y, finalmente, la habilidad de recuperarse de errores provocados por el usuario.

2.2.13 Desarrollo dirigido por pruebas
[Bec02]

El desarrollo dirigido por pruebas no es una técnica en sí misma, pero promociona el uso de pruebas como una parte subordinada al documento de especificación de requisitos en vez de una comprobación independiente de que el software implementa dichos requerimientos correctamente.

3 Técnicas de pruebas

Uno de los objetivos de las pruebas es revelar el máximo número posible de fallos potenciales y muchas técnicas se han desarrollado con este objetivo, intentando "romper" el programa ejecutando una o más pruebas seleccionadas de un cierto grupo de ejecuciones considerado equivalente. El principio subyacente de estas técnicas es tratar de ser lo más sistemático posible identificando un conjunto representativo de comportamientos del programa; por ejemplo, identificando subclases del dominio de entrada de datos, de los escenarios, de los estados y del flujo de datos.

Es difícil encontrar una base homogénea para clasificar todas las técnicas, por lo que la aquí utilizada debe entenderse como un compromiso. La clasificación se basa en cómo los ingenieros del software generan las pruebas basándose en su intuición y experiencia, en las especificaciones, la estructura del código, los errores a descubrir (reales o artificiales), el uso de campos de entrada de datos o, en último término, la naturaleza de la aplicación. Algunas veces, estas técnicas se clasifican como de *caja blanca* (también conocidas como *caja de cristal*), si las pruebas están basadas en información acerca de cómo se ha diseñado o programado el software, o como de *caja negra* si los casos de prueba se basan solamente en el comportamiento de la entrada y salida de datos. Una última categoría se basa en el uso combinado de dos o más técnicas. Obviamente, no todo el mundo usa estas técnicas con la misma frecuencia. La siguiente lista incluye las técnicas que los ingenieros de software deberían conocer.

3.1 Pruebas basadas en la intuición y experiencia del ingeniero de software

3.1.1 Pruebas ad hoc
[Kan99:c1]

Quizás la técnica usada más globalmente continúan siendo las pruebas ad hoc: las pruebas se generan a partir la habilidad, intuición y experiencia en programas similares del ingeniero de software. Las pruebas ad hoc pueden ser útiles para identificar casos de prueba especiales, aquellos que no se pueden extraer fácilmente mediante técnicas formales.

3.1.2 Pruebas por exploración

Las pruebas por exploración se definen como aprendizaje, diseño de pruebas y ejecución de pruebas al mismo tiempo. Esto significa que las pruebas no se definen primero como parte de un plan de pruebas establecido, si no que se diseñan, ejecutan y se modifican dinámicamente. La efectividad de las pruebas por exploración se basa en el conocimiento del ingeniero de software, que se puede derivar de varias fuentes: el comportamiento observado del producto durante las pruebas, su familiaridad con la aplicación, la plataforma o el proceso de fallos, los posibles tipos de errores y fallos, el riesgo asociado con un producto en particular, etc. [Kan01:c3]

3.2 Técnicas basadas en la especificación

3.2.1 Particiones de equivalencia
[Jor02:c7; Kan99:c7]

El dominio de la entrada de datos se subdivide en colecciones de subconjuntos, o clases de equivalencia, las cuales se consideran equivalentes de acuerdo con la relación especificada. Un grupo representativo de pruebas (a veces solo uno) se toma de cada clase.

3.2.2 Análisis de los valores límite
[Jor02:c6; Kan99:c7]

Casos de prueba se seleccionan en y cerca de los límites del dominio de las variables de la entrada de datos, basándose en la idea de que una gran parte de los errores se concentran cerca de los valores extremos de la entrada de datos. Una extensión de esta técnica son las *pruebas de robustez*, donde se seleccionan casos de prueba que se encuentran fuera del dominio de las variables de la entrada de datos, para comprobar la robustez del programa con entradas de datos erróneas e inesperadas.

3.2.3 Tablas de decisión
[Bei90:c10s3] (Jor02)

Las tablas de decisión representan relaciones lógicas entre condiciones (mayoritariamente entradas) y acciones (mayoritariamente salidas). Los casos de prueba se derivan sistemáticamente considerando cada combinación de condiciones y acciones posibles. Una técnica relacionada es el *gráfico causa-efecto*. [Pfl01:c9]

3.2.4 Basadas en máquinas de estado finito
[Bei90:c11; Jor02:c8]

Al modelar un programa como una máquina de estado finito, se pueden seleccionar las pruebas de manera que cubran estados y sus transiciones.

3.2.5 Pruebas basadas en las especificaciones formales
[Zhu97:s2.2] (Ber91; Dic93; Hor95)

Si las especificaciones se proporcionan en un lenguaje formal, es posible realizar una derivación automática de los casos de prueba funcionales y, al mismo tiempo, proporcionar unos resultados de referencia, un oráculo, que se usa para comprobar los resultados de las pruebas. Existen métodos para derivar casos de prueba de especificaciones basadas en el modelo (Dic93, Hor95) o especificaciones algebraicas. (Ber91)

3.2.6 Pruebas aleatorias
[Bei90:c13; Kan99:c7]

En este caso las pruebas se generan de una manera completamente aleatoria, lo que no debe confundirse con las pruebas estadísticas basadas en el perfil operativo descritas en el punto 3.5.1 *Perfil Operativo*. Esta forma de realizar pruebas se incluye en la categoría de entradas basadas en la especificación, ya que el domino de las entradas de datos se debe conocer para ser capaces de seleccionar elementos aleatorios del mismo.

3.3 Técnicas basadas en el código

3.3.1 Criterio basado en el flujo de control
[Bei90:c3; Jor02:c10] (Zhu97)

Los criterios de cobertura están basados en el flujo de control se usan para cubrir todos los bloques de código o líneas de código individuales o una combinación especifica de los mismos. Hay varios criterios de cobertura propuestos, como cobertura de condición/decisión. El criterio basado en el flujo de control más efectivo son las pruebas de caminos, cuyo objetivo es verificar todos los caminos de control de tipo entrada-salida del gráfico de flujos. Como, en general, las pruebas de caminos no son posibles debido a los bucles, en la práctica se usan otros criterios menos exigentes, como pruebas de líneas de código, pruebas de condiciones y pruebas de decisión. La idoneidad de dichas pruebas se mide en porcentajes; por ejemplo, cuando las pruebas han ejecutado todas las condiciones al menos una vez, se dice que se ha conseguido una cobertura de condiciones del 100%.

3.3.2 Criterio basado en el flujo de datos
[Bei90:c5] (Jor02; Zhu97)

En las pruebas basadas en el flujo de datos, el gráfico de flujos de control tiene anotaciones con información acerca de como las variables del programa se definen, usan y destruyen. El criterio más efectivo, todos los caminos de uso-definición, requiere que para cada variable, se ejecute cada uno de los segmentos del camino del flujo de control de esa variable a un usa de esa definición. Para reducir el número de caminos necesarios, se emplean estrategias menos efectivas como todas las definiciones y todos los usos.

3.3.3 Modelos de referencia para pruebas basadas en el código (gráfico de flujos, gráfico de llamadas)
[Bei90:c3; Jor02:c5]

Aunque no es una técnica en sí misma, la estructura de control de un programa se representa usando gráficos de flujo en las técnicas de pruebas basadas en código. Un gráfico de flujo es un gráfico dirigido cuyos nodos y arcos se corresponden con elementos del programa. Por ejemplo, los nodos podrían representar líneas de código o secuencias de líneas de código ininterrumpidas y los arcos la transferencia de control entre nodos.

3.4 *Técnicas basadas en errores*
(Mor90)

Con diferentes niveles de formalización, las técnicas basadas en errores idean casos de prueba que están especialmente orientados a descubrir categorías de errores probables o predefinidos.

3.4.1 Conjeturar errores
[Kan99:c7]

En la conjetura de errores, los casos de pruebas se han diseñado específicamente por ingenieros de software intentando imaginar los errores más probables en un programa determinado. La historia de errores descubiertos en proyectos anteriores es una buena fuente de información, como lo es también la experiencia del ingeniero.

3.4.2 Pruebas por mutación
[Per95:c17; Zhu97:s3.2-s3.3]

Un mutante es una versión ligeramente modificada de un programa al que se le está haciendo pruebas, diferenciándose tan solo en un pequeño cambio sintáctico. Cada caso de prueba se aplica al original y a los mutantes generados: si una prueba consigue identificar la diferencia entre el programa y el mutante, se dice que se ha "matado" al mutante. Esta técnica se concibió originalmente para evaluar un conjunto de pruebas (véase 4.2), las pruebas por mutación son un criterio de pruebas en sí mismas: o se generan pruebas aleatorias hasta que se han matado los mutantes suficientes, o se diseñan pruebas específicas para matar a los mutantes supervivientes. En el último caso, las pruebas por mutación se pueden clasificar como técnicas basadas en código. El efecto de acoplamiento, que es la base asumida en las pruebas de mutación, consiste en asumir que buscando errores sintácticos simples, se encontrarán otros más complejos pero existentes. Para que esta técnica sea efectiva, se debe poder derivar un número importante de mutantes de una manera sistemática.

3.5 Técnicas basadas en el uso

3.5.1 Perfil operativo
[Jor02:c15; Lyu96:c5; Pfl01:c9]

Durante pruebas para la evaluación de la confiabilidad, el entorno de pruebas debe reproducir el entorno operativo del software tan fielmente como sea posible. La idea es deducir la futura confiabilidad del software durante su use real desde los resultados de las pruebas.

3.5.2 Pruebas Orientadas a la Confiabilidad del Software
[Lyu96:c6]

Las pruebas orientadas a la confiabilidad del software (SRET) son un método de pruebas que forma parte del proceso de desarrollo completo, donde la realización de pruebas está "diseñada y guiada por los objetivos de confiabilidad y el uso relativo esperado y lo críticas que sean las distintas funciones en ese ámbito"

3.6 Técnicas basadas en la naturaleza de la aplicación

Las técnicas anteriores se pueden aplicar a cualquier tipo de software. Sin embargo, para algunos tipos de aplicaciones, es necesario conocimientos específicos adicionales para derivar las pruebas. La siguiente lista proporciona unas cuantas áreas de pruebas especializadas, basándose en la naturaleza de la aplicación que se está comprobando:

- Pruebas Orientadas a Objetos [Jor02:c17; Pfl01:c8s7.5] (Bin00)
- Pruebas basadas en componentes
- Pruebas para Internet
- Pruebas para GUI [Jor20]
- Pruebas para programas concurrentes (Car91)
- Pruebas de conformidad de protocolos (Pos96; Boc94)
- Pruebas para sistemas de tiempo real (Sch94)
- Pruebas para sistemas de seguridad crítica (IEEE1228-94)

3.7 Seleccionando y combinando técnicas

3.7.1 Funcional y estructuralmente
[Bei90:c1s.2.2; Jor02:c2, c9, c12; Per95:c17] (Pos96)

Las técnicas de pruebas basadas en las especificaciones y el código se contrastan frecuentemente como pruebas funcionales vs estructurales. Estos dos métodos de selección de pruebas no se deber ver como alternativos si no como complementarios; de hecho, usan fuentes de información diferentes y se ha comprobado que remarcan diferentes tipos de problemas. Estas técnicas se pueden combinar, dependiendo del presupuesto para pruebas.

3.7.2 Deterministas vs aleatorias
(Ham92; Lyu96:p541-547)

Los casos de pruebas se pueden seleccionar de una forma determinista, de acuerdo con una de las varias técnicas enunciadas, o seleccionadas aleatoriamente de una distribución de entradas de datos, como se hace normalmente en las pruebas de confiabilidad. Existen varias comparaciones analíticas y empíricas que analizan las condiciones en que uno de los métodos es más efectivo que el otro.

Para conseguir esto, se le asigna una probabilidad de distribución, o perfil, a las entradas de datos, basándose en la frecuencia en que suceden durante el funcionamiento real.

4 Medidas de las pruebas

Algunas veces, las técnicas de pruebas se confunden con los objetivos de las pruebas. Las técnicas de pruebas se deben ver como medios que ayudan a conseguir los objetivos de las pruebas. Por ejemplo, la cobertura de condiciones es una técnica de pruebas muy popular. Conseguir el valor de la cobertura de condiciones no debería ser un objetivo de las pruebas en sí mismo: es solo un medio para mejorar las posibilidades de encontrar fallos realizando pruebas sistemáticas en cada condición del programa para un punto de decisiones. Para prevenir dichas interpretaciones erróneas, debería hacerse una distinción muy clara entre las medidas de las pruebas, que proporcionan una evaluación del programa que se está comprobando, basada en los resultados observados de las pruebas y aquellas que evalúan la completitud del conjunto de pruebas. Se proporciona más información acerca de medidas para programas en el KA de la Gestión del la Ingeniería del Software, punto 6, *Medidas en la ingeniería del software*. Se puede encontrar más información en el KA de la Gestión del la Ingeniería del Software, punto 4, *Proceso y medidas del producto*.

Las medidas se consideran, normalmente, como esenciales en los análisis de calidad. Las medidas también se pueden utilizar para optimizar la planificación y ejecuciones de las pruebas. La gestión de pruebas puede utilizar varios procesos para medir o vigilar el progreso realizado. Las medidas relacionadas con el proceso de gestión de pruebas se abordan en el punto 5.1 *Consideraciones prácticas*.

4.1 *Evaluación de un programa durante las pruebas* (IEEE982.1-98)

4.1.1 Medidas para ayudar en la planificación y diseño de pruebas de programas
[Bei90:c7s4.2; Jor02:c9] (Ber96; IEEE982.1-88)

Las medidas basadas en el tamaño de un programa (por ejemplo, número de líneas de código o métodos) o en la estructura de un programa (como la complejidad), se usan para guiar a las pruebas. Las medidas estructurales pueden incluir medidas entre módulos del programa, en términos de la frecuencia en que cada módulo llama a los otros.

4.1.2 Tipos de errores, clasificación y estadísticas
[Bei90:c2; Jor02:c2; Pfl01:c8] (Bei90; IEEE1044-93; Kan99; Lyu96)

La literatura de pruebas es rica a la hora de clasificar y analizar errores. Con el objetivo de hacer las pruebas más efectivas, es importante saber que tipos de errores se pueden encontrar en un programa que se está comprobando y la frecuencia relativa en que estos errores han sucedido antes. Esta información puede ser muy útil para realizar predicciones de calidad y también para mejorar el proceso. Se puede encontrar más información en el KA de la Calidad del Software, punto 3.2 *Caracterización de defectos*. Existe un estándar del IEEE acerca de como clasificar "anomalías del software" (IEEE1044-93).

4.1.3 Densidad de fallos
[Per95:c20] (IEEE982.1-88; Lyu96:c9)

Un programa que se está comprobando se puede valorar contando y clasificando los errores descubiertos por su tipo. Parra cada tipo de error, la densidad de errores se mide como la razón entre el número de errores encontrados y el tamaño del programa.

4.1.4 Vida de las pruebas, evaluación de confiabilidad
[Pfl01:c9] (Pos96:p146-154)

Una estimación estadística de la confiabilidad del software, que se puede conseguir mediante la realización y evaluación de la confiabilidad (véase punto 2.2.5), se puede usar para evaluar un producto y decidir si las pruebas se pueden detener o no.

4.1.5 Modelos de crecimiento de la confiabilidad
[Lyu96:c7; Pfl01:c9] (Lyu96:c3, c4)

Los modelos de crecimiento de la confiabilidad proporcionan una predicción de confiabilidad basada en los fallos observados durante la realización y evaluación de la confiabilidad (véase punto 2.2.5). Estos modelos asumen, en general, que los errores que causan los fallos observados se han arreglado (aunque algunos modelos también aceptan arreglos imperfectos), y por tanto, el producto muestra una confiabilidad incremental de promedio. Existen docenas de modelos publicados en la actualidad. Muchos se basan en algunas presunciones comunes, y otros no. Mayoritariamente, estos modelos se dividen en modelos de *cuenta de fallos* y *tiempo entre fallos*.

4.2 Evaluación de las pruebas realizadas

4.2.1 Medidas de la cobertura/completitud
[Jor02:c9; Pfl01:c8] (IEEE982.1-88)

Varios criterios de idoneidad de las pruebas necesitan que los casos de pruebas ejecuten sistemáticamente un conjunto de elementos identificados en el programa o en la especificación (véase punto 3). Para evaluar la completitud de las pruebas realizadas, los ingenieros de pruebas pueden monitorizar los elementos cubiertos y su número total. Por ejemplo, es posible medir el porcentaje de condiciones cubiertas ejecutadas entre las definidas en la especificación. La idoneidad de los criterios basados en código necesita la instrumentación adecuada del programa que se está comprobando.

4.2.2 Introducción de errores
[Pfl01:c8] (Zhu97:s3.1)

Algunas veces se introducen errores artificialmente en un programa antes de comprobarlo. Cuando las pruebas se realizan, algunos de estos errores aparecerán y posiblemente algunos otros que ya estaban en el software también aparecerán. En teoría, dependiendo de cuál de los errores artificiales aparezca y cuántos de ellos, se puede evaluar la efectividad de las pruebas y se puede estimar el número restante de errores genuinos. En la práctica, los matemáticos estadísticos se cuestionan la distribución y representatividad de los errores introducidos en relación con los errores genuinos y el tamaño pequeño de la muestra en la que se basa cualquier extrapolación. Algunos incluso afirman que esta técnica debería usarse con sumo cuidado, ya que introducir errores en el software acarrea el riesgo obvio de olvidarlos allí.

4.2.3 Puntuación de la mutación
[Zhu97:s3.2-s3.3]

En las pruebas por mutación (véase el punto 3.4.2), la razón de mutantes matados por número total de mutantes generados puede ser una medida de la efectividad del conjunto de pruebas realizadas.

4.2.4 Comparación y efectividad relativa de las diferentes técnicas
[Jor02:c9, c12; Per95:c17; Zhu97:s5] (Fra93; Fra98; Pos96: p64-72)

Se han llevado a cabo varios estudios para comparar la efectividad relativa de las diferentes técnicas de pruebas. Es importante ser preciso acerca de la propiedad contra la cual las técnicas se han calificado; ¿cual, por ejemplo, es el significado exacto dado al término "efectividad"? Las interpretaciones posibles son: el número de pruebas necesarias para encontrar el primer fallo, la razón entre el número de errores encontrados durante las pruebas y todos los errores encontrados durante y después de las pruebas, o cual fue la mejora de la confiabilidad. Se han llevado a cabo comparaciones analíticas y empíricas entre las diferentes técnicas, de acuerdo con cada uno de los significados de efectividad especificados antes.

5 El Proceso de las Pruebas

Los conceptos de pruebas, estrategias, técnicas y medidas han de ser integrados en un proceso definido y controlado, que debe ser gestionado por personas. El proceso de las pruebas soporta actividades y sirve de guía a los equipos de pruebas, desde la planificación de las pruebas hasta la evaluación de los resultados de las pruebas, de tal manera que se puede proporcionar una garantía justificada de que los objetivos de las pruebas se conseguirán de una manera económica.

5.1 Consideraciones prácticas

5.1.1 Actitudes y programación egoless
[Bei90:c13s3.2; Pfl01:c8]

Un elemento muy importante para el éxito de las pruebas es una actitud de colaboración respecto a las actividades de pruebas y garantía de calidad. Los jefes de proyecto tienen un papel fundamental en fomentar una recepción favorable en general respecto al descubrimiento de fallos durante el desarrollo y mantenimiento; particularmente, previniendo que los programadores se obsesionen con quien es el dueño del código, de tal forma que ninguno se sienta responsable por los fallos que aparezcan en su código.

5.1.2 Guías para las pruebas
[Kan01]

Se pueden guiar las fases de pruebas con varios mecanismos, por ejemplo; en pruebas basadas en el riego, que usa los riesgos en el producto para asignar prioridad y centrar la atención de las estrategias de pruebas; o en las pruebas basadas en situaciones, en las que los casos de pruebas se definen y basan en escenarios de software especificados.

5.1.3 Gestión del proceso de las pruebas
[Bec02: III; Per95:c1-c4; Pfl01:c9] (IEEE1074-97; IEEE12207.0-96:s5.3.9, s5.4.2, s6.4, s6.5)

Las actividades de pruebas realizadas a diferentes niveles (véase punto 2, *Niveles de pruebas*) se deben organizar, junto con las personas, herramientas, normas y medidas, en un proceso bien definido que será una parte integral del ciclo de vida del software. En el estándar IEEE/EIA 12207.0, las pruebas no se describen como un proceso independiente, si no que los principios de las actividades de las pruebas se encuentran incluidos con los cinco procesos primarios del ciclo de vida y con los procesos de soporte. En el estándar IEEE 1074, las pruebas se agrupan con otras actividades de evaluación como una parte integral del ciclo de vida completo.

5.1.4 Documentación y productos de las pruebas
[Bei90:c13s5; Kan99:c12; Per95:c19; Pfl01:c9s8.8] (IEEE829-98)

La documentación es una parte integral de la formalización del proceso de las pruebas. El estándar del IEEE Estándar para la Documentación de las Pruebas del Software (IEEE829-98) proporciona una buena descripción de los documentos de las pruebas y su relación entre cada uno y con el proceso de las pruebas. La documentación de pruebas puede incluir, entre otros, el Plan de Pruebas, la Especificación del Diseño de las Pruebas, la Especificación del Procedimiento de las Pruebas, la Especificación de los Casos de Pruebas, el Diario de las Pruebas y el Informe de Problemas o de Incidentes durante las Pruebas. El software que se está comprobando se documenta como el Artículo en Pruebas. La documentación de las pruebas se debe generar y actualizar continuamente, con el mismo nivel de calidad que cualquier otro tipo de documentación en la ingeniería del software.

5.1.5 Equipo de pruebas interno vs equipo independiente
[Bei90:c13s2.2-c13s2.3; Kan99:c15; Per95:c4; Pfl01:c9]

La formalización del proceso de pruebas también puede formalizar la organización del equipo de pruebas. El equipo de pruebas puede estar compuesto por miembros internos (parte del equipo del proyecto, involucrados o no en la construcción del software), o de miembros externos, con la esperanza de contar con una perspectiva independiente y sin prejuicios, o, finalmente, de miembros internos y externos. La decisión puede ser afectada por consideraciones como coste, planificación, nivel de madurez de las organización involucradas y como de crítica sea la aplicación.

5.1.6 Estimación coste/esfuerzo y otras medidas del proceso
[Per95:c4, c21] (Per95: Appendix B; Pos96:p139-145; IEEE982.1-88)

Los jefes de proyectos pueden usar varias medidas, acerca de los recursos invertidos en las pruebas y de la efectividad de las varias fases de pruebas en encontrar fallos, para controlar y mejorar el proceso de las pruebas. Estas medidas de las pruebas pueden cubrir, entre otros, aspectos como el número de casos de pruebas especificados, el número de casos de pruebas ejecutados, el número de casos de pruebas superados y el número de casos de pruebas no superados.

La evaluación de los informes de las fases de pruebas se puede combinar con análisis de las raíces de las causas para evaluar la efectividad del proceso de las pruebas en encontrar errores tan pronto como sea posible. Dicha evaluación se puede asociar con el análisis de riesgos. Lo que es más, los recursos que merece la pena invertir en las pruebas deberían ser proporcionales al uso/importancia de la aplicación: diferentes técnicas tienen distinto coste y proporcionan diferentes niveles de seguridad en la confiabilidad del producto.

5.1.7 Finalización
[Bei90:c2s2.4; Per95:c2]

Se debe tomar una decisión acerca de cuantas pruebas son suficientes y cuando la fase de pruebas se puede finalizar. Las medidas concienzudas, como las conseguidas mediante cobertura de código o completitud funcional y la estimación de densidad de errores o de confiabilidad operativa, proporcionan un suporte muy útil, pero no son suficientes por sí mismas. Esta decisión también incluye consideraciones acerca del coste y los riesgos en que se incurrirá debido a los fallos potenciales que aún queden, en vez del coste que conllevaría continuar realizando pruebas. Véase también el punto 1.2.1 *Criterios de selección de pruebas/Criterios de idoneidad de pruebas*.

5.1.8 Reutilización de pruebas y patrones de pruebas
[Bei90:c13s5]

Con el objetivo de realizar pruebas o mantenimiento de una forma organizada y efectiva respecto al coste, los medios usados para realizar pruebas en cada parte del

software se deberían reutilizar de una forma sistemática. Dicho repositorio de material de pruebas debe estar bajo el control de un software de gestión de configuraciones, de forma que los cambios en los requerimientos del software o el diseño queden reflejados en cambios en el alcance de las pruebas realizadas.

Las soluciones adoptadas para realizar pruebas en determinados tipos de aplicaciones bajo determinadas circunstancias, teniendo en cuenta los motivos detrás de las decisiones que se han tomado, forman un patrón de pruebas que se puede documentar y ser reutilizado en proyectos similares.

5.2 Actividades de las pruebas

En este punto, se verá una pequeña introducción a las actividades del software; gestionar con éxito las actividades relacionada con las pruebas, como la siguiente descripción da a entender, depende en gran medida del proceso de Gestión de Configuración del Software.

5.2.1 Planificación
[Kan99:c12; Per95:c19; Pfl01:c8s7.6] (IEEE829-98:s4; IEEE1008-87:s1-s3)

Como cualquier otro aspecto de la gestión de proyectos, las actividades de las pruebas se deben planificar. Algunos aspectos clave de la planificación de las pruebas incluyen la coordinación de personal, la gestión de instalaciones y equipos disponibles (que pueden incluir soportes magnéticos, planes de pruebas y procedimientos) y planificar en caso de posibles situaciones no deseables. Si se mantiene más de una línea base del software al mismo tiempo, una importante consideración de planificación es el tiempo y esfuerzo necesario para asegurarse de que se ha usado la configuración correcta para establecer el entorno de pruebas.

5.2.2 Generación de casos de pruebas
[Kan99:c7] (Pos96:c2; IEEE1008-87:s4, s5)

La generación de caos de pruebas se basa en el nivel de pruebas que se vaya a realizar y en las técnicas de pruebas a usar. Los casos de pruebas deberían estar bajo el control de un software de gestión de configuraciones e incluir los resultados esperados para cada prueba.

5.2.3 Desarrollo en el entorno de pruebas
[Kan99:c11]

El entorno usado para las pruebas debería ser compatible con las herramientas de ingeniería de software. Debería facilitar el desarrollo y control de casos de pruebas y la anotación y recuperación de los resultados esperados, los scripts y otros materiales de pruebas.

5.2.4 Ejecución
[Bei90:c13; Kan99:c11] (IEEE1008-87:s6, s7)

La ejecución de las pruebas deberían incluir un principio básico de experimentación científica: todos los pasos durante las pruebas se deberían realizar y documentar de una forma lo suficientemente clara, que cualquier otra persona debería ser capaz de reproducir los resultados. Por tanto, las pruebas deben realizarse de acuerdo con los procedimientos documentados y usando una versión claramente definida del software que se está comprobando.

5.2.5 Evaluación de los resultados de las pruebas
[Per95:c20,c21] (Pos96:p18-20, p131-138)

Los resultados de las pruebas se deben evaluar para determinar si las pruebas han sido satisfactorias o no. En la mayoría de los casos, "satisfactorias" significa que el software se ha ejecutado como se esperaba y no ha tenido ningún resultado inesperado importante. No todos los resultados inesperados son necesariamente errores, ya que se podría considerar que algunos son simple ruido. Antes de que se pueda arreglar un error, se necesita realizar un análisis y algún trabajo de depuración para identificarlo, aislarlo y describirlo. Cuando los resultados de las pruebas son particularmente importantes, puede que se convoque una revisión formal para evaluarlas.

5.2.6 Notificación de problemas/Diario de pruebas
[Kan99:c5; Per95:c20] (IEEE829-98:s9-s10)

Las actividades de las pruebas se pueden añadir a un diario de pruebas para identificar cuando una prueba se ha ejecutado, quien la ha realizado, que configuración del software se ha utilizado y cualquier otra información relevante de identificación. Resultados inesperados o incorrectos se pueden añadir a un sistema de notificación de problemas, cuyos datos serán la base para procesos de depuración posteriormente y para arreglar los errores que causaron problemas durante las pruebas. Las anomalías no clasificadas como errores también se podrían documentar, en caso de que más tarde resulte que producen problemas más serios de lo que se pensó originalmente. Los informes de pruebas también son una entrada para los procesos de requerimientos de cambió de gestión (véase el KA de la Gestión de la Configuración del Software, punto 3, *Control de la configuración del software*)

5.2.7 Seguimiento de defectos
[Kan99:c6]

Los fallos observados durante las pruebas son, en la mayoría de los casos, debidos a errores o defectos en el software. Dichos defectos se pueden analizar para determinar cuando fueron introducidos en el software, que clase de error produjo que se aparecieran (por ejemplo requerimientos definidos pobremente, declaraciones incorrectas de variables, fallo de memoria o errores de programación) y cuando deberían haber sido observados en el software por primera vez. La información del seguimiento de defectos se usa para determinar qué aspectos de la ingeniería del software necesitan mejorase y la efectividad de análisis y pruebas anteriores.

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Bec02]	[Bei09]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
1. Fundamentos de las Pruebas Software									
1.1Terminología relacionada									
Definiciones de pruebas y terminología		c1	c2			c2s2.2			
Errores Vs. Fallos			c2			c2s2.2	c1	c8	
1.2 Cuestiones Clave									
Criterios de selección de Pruebas/ Criterios de idoneidad de Pruebas								c8s7.3	s1.1
Efectividad de las Pruebas/ Objetivos de las Pruebas		c1s1.4					c21		
Realizar pruebas para la identificación de defectos		c1		c1					
El problema del oráculo		c1							
Limitaciones teóricas y prácticas de las pruebas				c2					
El problema de los caminos no alcanzables		c3							
Posibilidad de hacer pruebas		c3, c13							
1.3 Relación de las pruebas con otras actividades									
2. Niveles de Pruebas									
2.1El objeto de la prueba		c1	c13					c8	
Pruebas de Unidad		c1					c17	c8s7.3	
Pruebas de Integración			c13, c14					c8s7.4	
Pruebas del sistema			c15					c9	
2.2 Objetivos de las Pruebas							c8	c9s8.3	
Pruebas de aceptación/ calificación							c10	c9s8.5	
Pruebas de Instalación							c9	c9s8.6	
Pruebas Alfa y Beta				c13					
Pruebas de conformidad/Pruebas funcionales/Pruebas de corrección				c7			c8		
Materialización de la confiabilidad y evaluación						c7		c9s8.4	
Pruebas de Regresión				c7			c11, c12	c9s8.1	
Pruebas de Rendimiento							c17	c9s8.3	
Pruebas de Desgaste							c17	c9s8.3	
Pruebas de Continuidad									
Pruebas de Recuperación							c17	c9s8.3	
Pruebas de Configuración								c9s8.3	
Pruebas de Facilidad de Uso				c8			c8	c9s8.3	
Desarrollo dirigido por Pruebas	III								

	[Bec02]	[Bei09]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
3. Técnicas de Pruebas									
3.1Pruebas Basadas en la Intuición y Experiencia									
Pruebas ad hoc				c1					
Pruebas por Exploración					c3				
3.2 Basadas en la Especificación									
Particiones de Equivalencia			c7	c7					
Análisis de los Valores Límite			c6	c7					
Tablas de Decisión		c10s3						c9	
Basadas en Máquinas de Estado Finito		c11	c8						
Basadas en las especificaciones formales									s2.2
Pruebas aleatorias		c13		c7					
3.3 Basadas en el código									
Criterio basado en el Flujo de Control		c3	c10					c8	
Criterio basado en el Flujo de Datos		c5							
Modelos de referencia para pruebas basadas en código		c3	c5						
3.4 Basadas en errores									
Conjeturar errores				c7					
Pruebas por mutación							c17		s3.2, s3.3
3.5 Basadas en el uso									
Perfil operativo			c15			c5		c9	
Pruebas Orientadas a la Confiabilidad del Software						c6			
3.6 Basadas en la Naturaleza de la Aplicación									
Pruebas Orientadas a Objetos									
Basadas en Componentes			c17				c8s7.5		
Pruebas para Internet									
Pruebas para GUI			c20						
Pruebas para Programas Concurrentes									
Pruebas de conformidad de Protocolos									
Pruebas para Sistemas de Tiempo Real									
Pruebas para Sistemas de Seguridad Crítica									
3.7 Seleccionando y combinando técnicas						c17			
Funcional y Estructuralmente		c1s2.2	c1, c11s11.3						
Deterministas Vs. Aleatorias									

	[Bec02]	[Bei09]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
4. Medidas de las Pruebas									
4.1 Evaluación de un programa									
Medidas para ayudar en la planificación y diseño de pruebas de programas		c7s4.2	c9						
Tipos de errores, clasificación y Estadísticas		c2	c1					c8	
Densidad de fallos							c20		
Vida de las pruebas								c9	
Modelos de crecimiento de la Confiabilidad						c7		c9	
4.2 Evaluación de las pruebas realizadas									
Medidas de la cobertura/completitud			c9					c8	
Introducción de errores								c8	
Puntuación de la mutación									s3.2, s3.3
Comparación y efectividad relativa de las técnicas			c8, c11				c17		s5
5. El proceso de las pruebas									
5.1 Consideraciones prácticas									
Actitudes y programación egoless		c13s3.2						c8	
Guías para las pruebas	III				c5				
Gestión del proceso de pruebas							c1-c4	c9	
Documentación y productos de las pruebas		c13s5		c12			c19	c9s8.8	
Equipo de Pruebas Interno Vs. Equipo Independiente		c13s2.2, c1s2.3		c15			c4	c9	
Estimación Coste/Esfuerzo y otras Medidas del Proceso							c4, c21		
Finalización		c2s2.4					c2		
Reutilización de pruebas y patrones de pruebas		c13s5							
5.2 Actividades de Pruebas									
Planificación				c12			c19	c87s7.6	
Generación de casos de Prueba				c7					
Desarrollo del entorno de Pruebas				c11					
Ejecución		c13		c11					
Evaluación de los resultados							c20, c21		
Notificación de Problemas/ Diario de pruebas				c5			c20		
Seguimiento de los defectos				c6					

REFERENCIAS RECOMENDADAS PARA LA GESTIÓN DEL SOFTWARE

[Bec02] K. Beck, *Test-Driven Development by Example*, Addison-Wesley, 2002.

[Bei90] B. Beizer, *Software Testing Techniques*, International Thomson Press, 1990, Chap. 1-3, 5, 7s4, 10s3, 11, 13.

[Jor02] P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, second edition, CRC Press, 2004, Chap. 2, 5-10, 12-15, 17, 20.

[Kan99] C. Kaner, J. Falk, and H.Q. Nguyen, Testing Computer Software, second ed., John Wiley & Sons, 1999, Chaps. 1, 2, 5-8, 11-13, 15.

[Kan01] C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing*, Wiley Computer Publishing, 2001.

[Lyu96] M.R. Lyu, *Handbook of Software Reliability Engineering*, Mc-Graw-Hill/IEEE, 1996, Chap. 2s2.2, 5-7.

[Per95] W. Perry, *Effective Methods for Software Testing*, John Wiley & Sons, 1995, Chap. 1-4, 9, 10-12, 17, 19-21.

[Pfl01] S. L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001, Chap. 8, 9.

[Zhu97] H. Zhu, P.A.V. Hall and J.H.R. May, "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys*, vol. 29, iss. 4 (Sections 1, 2.2, 3.2, 3.3), Dec. 1997, pp. 366-427.

APÉNDICE A. LISTA DE LECTURAS ADICIONALES

(Bac90) R. Bache and M. Müllerburg, "Measures of Testability as a Basis for Quality Assurance," *Software Engineering Journal*, vol. 5, March 1990, pp. 86-92.

(Bei90) B. Beizer, *Software Testing Techniques*, International Thomson Press, second ed., 1990.

(Ber91) G. Bernot, M.C. Gaudel and B. Marre, "Software Testing Based On Formal Specifications: a Theory and a Tool," *Software Engineering Journal*, Nov. 1991, pp.387-405.

(Ber96) A. Bertolino and M. Marrè, "How Many Paths Are Needed for Branch Testing?" *Journal of Systems and Software*, vol. 35, iss. 2, 1996, pp. 95-106.

(Ber96a) A. Bertolino and L. Strigini, "On the Use of Testability Measures for Dependability Assessment," *IEEE Transactions on Software Engineering*, vol. 22, iss.2, Feb. 1996, pp. 97-108.

(Bin00) R.V. Binder, *Testing Object-Oriented Systems Models, Patterns, and Tools*, Addison-Wesley, 2000.

(Boc94) G.V. Bochmann and A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing," presented at *ACM Proc. Int'l Symp. on Software Testing and Analysis* (ISSTA '94), Seattle, Wash., 1994.

(Car91) R.H. Carver and K.C. Tai, "Replay and Testing for Concurrent Programs," *IEEE Software*, March 1991, pp. 66-74.

(Dic93) J. Dick and A. Faivre, "Automating the Generation and Sequencing of Test Cases from Model-Based Specifications," presented at *FME '93: Industrial-Strength Formal Methods*, LNCS 670, Springer-Verlag, 1993.

(Fran93) P. Frankl and E. Weyuker, "A Formal Análisis of the Fault Detecting Ability of Testing Methods," *IEEE Transactions on Software Engineering*, vol. 19, iss. 3, March 1993, p. 202.

(Fran98) P. Frankl, D. Hamlet, B. Littlewood, and L. Strigini, "Evaluating Testing Methods by Delivered Reliability," *IEEE Transactions on Software Engineering*, vol. 24, iss. 8, August 1998, pp. 586-601.

(Ham92) D. Hamlet, "Are We Testing for True Reliability?" *IEEE Software*, July 1992, pp. 21-27.

(Hor95) H. Horcher and J. Peleska, "Using Formal Specifications to Support Software Testing," *Software Quality Journal*, vol. 4, 1995, pp. 309-327.

(How76) W. E. Howden, "Reliability of the Path Analysis Testing Strategy," *IEEE Transactions on Software Engineering*, vol. 2, iss. 3, Sept. 1976, pp. 208-215.

(Jor02) P.C. Jorgensen, *Software Testing: A Craftsman's Approach*, second ed., CRC Press, 2004.

(Kan99) C. Kaner, J. Falk, and H.Q. Nguyen, "Testing Computer Software," second ed., John Wiley & Sons, 1999.

(Lyu96) M.R. Lyu, *Handbook of Software Reliability Engineering*, Mc-Graw-Hill/IEEE, 1996.

(Mor90) L.J. Morell, "A Theory of Fault-Based Testing," *IEEE Transactions on Software Engineering*, vol. 16, iss. 8, August 1990, pp. 844-857.

(Ost88) T.J. Ostrand and M.J. Balcer, "The Category-Partition Method for Specifying and Generating Functional Tests," *Communications of the ACM*, vol. 31, iss. 3, June 1988, pp. 676-686.

(Ost98) T. Ostrand, A. Anodide, H. Foster, and T. Goradia, "A Visual Test Development Environment for GUI Systems," presented at *ACM Proc. Int'l Symp. On Software Testing and Analysis* (ISSTA '98), Clearwater Beach, Florida, 1998.

(Per95) W. Perry, *Effective Methods for Software Testing*, John Wiley & Sons, 1995.

(Pfl01) S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice-Hall, 2001, Chap. 8, 9.

(Pos96) R.M. Poston, *Automating Specification-Based Software Testing*, IEEE, 1996.

(Rot96) G. Rothermel and M.J. Harrold, "Analyzing Regression Test Selection Techniques," *IEEE Transactions on Software Engineering*, vol. 22, iss. 8, Aug. 1996, p. 529.

(Sch94) W. Schütz, "Fundamental Issues in Testing Distributed Real-Time Systems," *Real-Time Systems Journal*, vol. 7, iss. 2, Sept. 1994, pp. 129-157.

(Voa95) J.M. Voas and K.W. Miller, "Software Testability: The New Verification," *IEEE Software*, May 1995, pp. 17-28.

(Wak99) S. Wakid, D.R. Kuhn, and D.R. Wallace, "Toward Credible IT Testing and Certification," *IEEE Software*, July-Aug. 1999, pp. 39-47.

(Wey82) E.J. Weyuker, "On Testing Non-testable Programs," *The Computer Journal*, vol. 25, iss. 4, 1982, pp. 465-470.

(Wey83) E.J. Weyuker, "Assessing Test Data Adequacy through Program Inference," *ACM Trans. On Programming Languages and Systems*, vol. 5, iss. 4, October 1983, pp. 641-655.

(Wey91) E.J. Weyuker, S.N. Weiss, and D. Hamlet, "Comparison of Program Test Strategies," presented at *Proc. Symp. on Testing, Analysis and Verification (TAV 4)*, Victoria, British Columbia, 1991.

(Zhu97) H. Zhu, P.A.V. Hall, and J.H.R. May, "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys*, vol. 29, iss. 4, Dec. 1997, pp. 366-427.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

(IEEE829-98) IEEE Std 829-1998, *Standard for Software Test Documentation*, IEEE, 1998.

(IEEE982.1-88) IEEE Std 982.1-1988, *IEEE Standard Dictionary of Measures to Produce Reliable Software*, IEEE, 1988.

(IEEE1008-87) IEEE Std 1008-1987 (R2003), *IEEE Standard for Software Unit Testing*, IEEE, 1987.

(IEEE1044-93) IEEE Std 1044-1993 (R2002), *IEEE Standard for the Classification of Software Anomalies*, IEEE, 1993.

(IEEE1228-94) IEEE Std 1228-1994, *Standard for Software Safety Plans*, IEEE, 1994.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996 // ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

CAPITULO 6
MANTENIMIENTO DEL SOFTWARE

ACRÓNIMOS

CMMI Capability Maturity Model Integration
ICSM International Conference on Software Maintenance
SCM Software Configuration Management
SQA Software Quality Assurance
V&V Verification and Validation
Y2K Year 2000

INTRODUCCIÓN:

Los esfuerzos de desarrollo de software dan como resultado la entrega de un producto de software que satisfaga las exigencias del usuario.
Como consecuencia el producto de software debe cambiarse o desarrollarse. Una vez en la operación, los defectos son destapados y emergen las nuevas exigencias del usuario.
La fase de mantenimiento del ciclo de vida comienza después de un período de garantía pero las actividades de mantenimiento ocurren mucho antes.

El mantenimiento de software es una parte del ciclo de vida de software. Sin embargo, históricamente, no ha recibido el mismo grado de atención que otras fases del ciclo de vida.
Históricamente, el desarrollo de software ha tenido un perfil mucho más alto que el mantenimiento de software en la mayor parte de organizaciones.
Esto ha cambiado ahora, las organizaciones se esfuerzan en exprimir al máximo su inversión de desarrollo de software por lo que desean que el software funcione tanto tiempo como sea posible. Las preocupaciones sobre el efecto 2000, enfocó una atención significativa en la fase de mantenimiento de software, y el paradigma Open Source ha atraído la atención a la cuestión de mantener software desarrollado por otros.

En la Guía, el mantenimiento de software es definido como la totalidad de actividades requeridas para proporcionar el apoyo rentable al software. Las actividades son realizadas durante la etapa de preentrega, así como durante la etapa de postentrega. Las actividades de preentrega incluyen la planificación para operaciones de postentrega, para la capacidad de mantenimiento, y para la determinación de logística para actividades de transición. Las actividades de postentrega incluyen la modificación de software, el entrenamiento, y el funcionamiento del help desk.
El Mantenimiento de Software KA está relacionado con otros aspectos de ingeniería de software. Por lo tanto, esta descripción KA está vinculada a otros capítulos de la Guía.

DESGLOSE DE PUNTOS DEL MANTENIMIENTO DEL SOFTWARE

El desglose de los puntos del Mantenimiento de Software KA se muestra en la Figura 1.

1. Fundamentos de mantenimiento de software

Esta primera sección presenta los conceptos y la terminología que forman una base subyacente a la comprensión de la función y el alcance de mantenimiento de software. Los puntos proporcionan definiciones y enfatizan por qué existe la necesidad de mantenimiento. Las categorías del mantenimiento de software son críticas a la comprensión de su significado subyacente.

1.1. Definiciones y Terminología.
[IEEE1219-98:s3.1.12; IEEE12207.0-96:s3.1,s5.5;ISO14764-99:s6.1]

El Mantenimiento de Software se define en el estándar IEEE para Mantenimiento de Software, IEEE 1219, como la modificación de un producto de software después de su entrega para corregir los fallos, mejorar el rendimiento u otros atributos, o adaptar el producto a un entorno modificado. La norma también se ocupa de las actividades de mantenimiento antes de la entrega del producto software, pero sólo en un apéndice del estándar.

El estándar IEEE / EIA 12207 para el ciclo de vida presenta esencialmente al mantenimiento como uno de los primeros procesos del ciclo de vida y describe el mantenimiento como el proceso de "modificación del código y la documentación asociada debido a un problema o la necesidad de mejora de un producto de software. El objetivo es modificar el producto software al mismo tiempo que preservar su integridad". ISO / IEC 14764, estándar internacional para el mantenimiento del

software, define el mantenimiento del software en los mismos términos que IEEE / EIA y 12207.
Hace hincapié en la entrega previa de los aspectos de mantenimiento, planificación, por ejemplo.

1.2. *La naturaleza de Mantenimiento*
[Pfl01:c11s11.2]

El mantenimiento de Software sostiene el producto de software en todas partes de su ciclo de vida operacional. La solicitud de modificación son registradas y rastreadas, el impacto de cambios propuestos es determinado, el código y otros artefactos de software son modificados, las pruebas son conducidas, y una nueva versión del producto de software es liberada. Proporcionan también, entrenamiento y el apoyo a los usuarios. Pfleeger [Pfl01] declara que " el mantenimiento tiene un más amplio alcance para rastrear y controlar que el desarrollo". Un mantenedor es definido por el IEEE / EIA 12207 como una organización que realiza actividades de mantenimiento [IEEE12207.0-96].

En este KA, el término a veces se refiere a las personas que realizan esas actividades, contrastándolos con los desarrolladores.

IEEE / EIA 12207 identifica las actividades principales de Mantenimiento de software como: proceso de implementación; problema y análisis de modificación; implementación de la modificación; mantenimiento, revisión y aceptación; migración, y retirada. Estas actividades se examinan en el punto 3.2 de Actividades de Mantenimiento.

Los mantenedores pueden aprender del conocimiento del desarrollo del software. Ponerse en contacto con los desarrolladores y la temprana participación por el mantenedor ayuda a reducir el esfuerzo de mantenimiento.

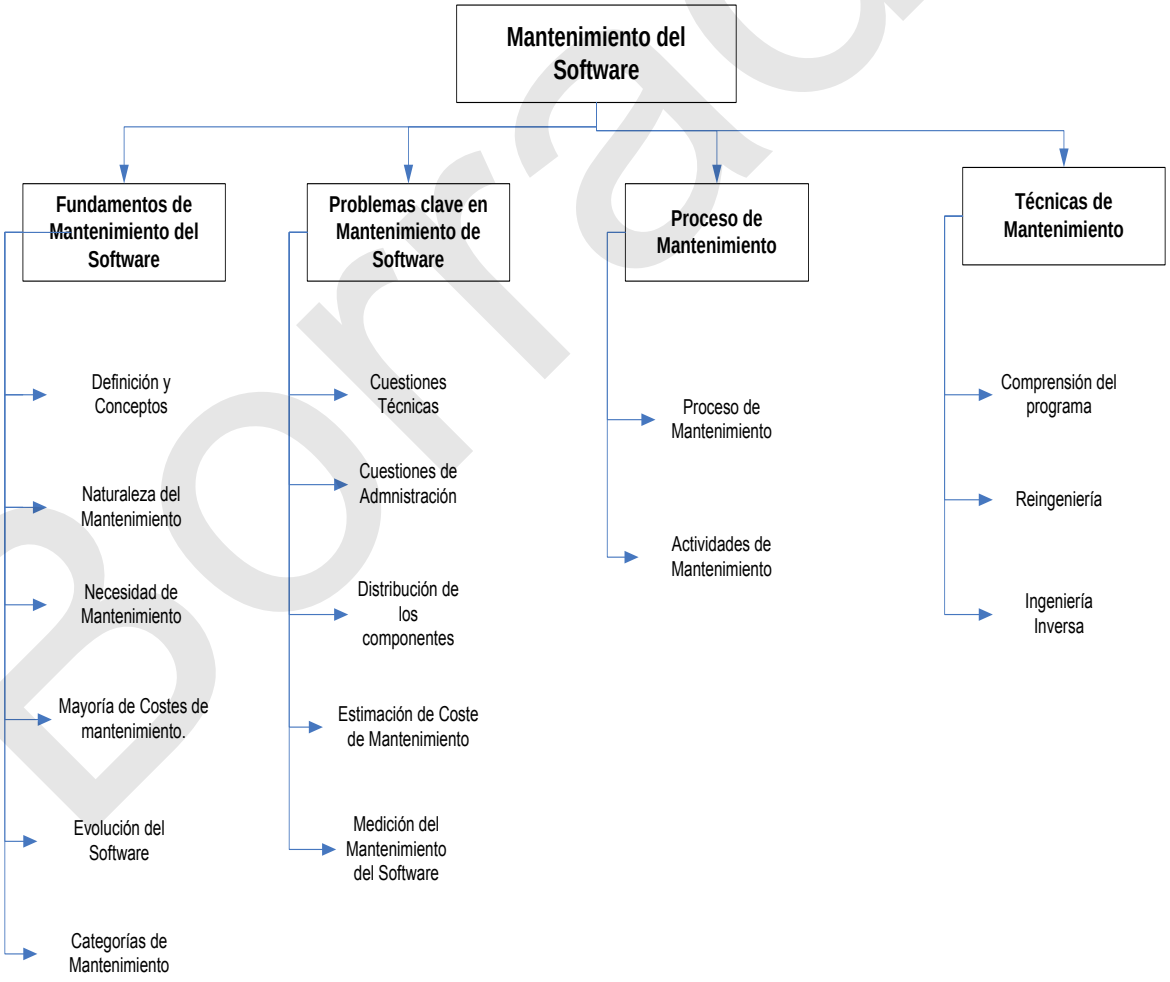


Figura 1 Desglose de los temas para Mantenimiento de Software KA

En algunos casos, el ingeniero de software no puede ser alcanzado o ha seguido adelante con otras tareas, que crean un desafío adicional para el mantenedor. El mantenimiento debe tomar los productos desarrollados, el código, o la documentación, por ejemplo, y apoyarlos inmediatamente y desarrollar/mantenerlos cada vez más sobre el ciclo de vida de software.

1.3. *Necesidad de Mantenimiento*
[Pfl01:c11s11.2; Pig97: C2s2.3; Tak97:c1]

El Mantenimiento es necesario para asegurar que el software sigue satisfaciendo las exigencias del usuario. El mantenimiento es aplicable al software desarrollado usando cualquier modelo de ciclo de vida de software (por ejemplo, en espiral). El sistema se cambia debido a acciones de software correctivas y no correctivas.
El mantenimiento debe ser realizado para:

- Corregir defectos
- Mejorar el diseño
- Llevar a la práctica las mejoras
- El interfaz con otros sistemas
- Adapta programas con diferente hardware diferente, software, características del sistema, e instalaciones de telecomunicaciones para que puedan ser usados
- Emigra software
- Retira el software

Las actividades del mantenedor comprenden cuatro características claves, según Pfleeger [Pfl01]:

- Mantenimiento de control de las funciones cotidianas del software
- Mantenimiento de control de modificación de software
- Perfeccionando funciones existentes
- Impedir la degradación del funcionamiento de software a niveles inaceptables

1.4. *Costes de Mantenimiento.*
[Abr93 :63-90; Pfl01: c11s11.3; Pig97: c3; Pre01: c30s2.1, c30s2.2]

El mantenimiento consume una parte importante de los recursos financieros del ciclo de vida del software. Una percepción común del mantenimiento del software es que se limita a parchear los fallos. Sin embargo, los estudios
y las encuestas a través de los años han indicado que la mayoría, más del 80%, del esfuerzo de mantenimiento del software se utiliza para acciones no correctivas. [Abr93, Pig97, Pre01] Jones (Jon91) describe el camino en cual gerentes de mantenimiento de software a menudo incluyen grupos de mejoras y correcciones en sus informes de gestión. Esta inclusión de solicitudes de mejoramiento con informes de problemas contribuye a algunas de las ideas erróneas en relación con el elevado costo de correcciones. La comprensión de las categorías de mantenimiento del software ayuda a comprender la estructura de costes del mantenimiento del software. Asimismo, la comprensión de los factores que influyen en el mantenimiento de un sistema puede ayudar a contener los costos. Pfleeger [Pfl01] presenta algunos de los factores técnicos y no técnicos que afectan a los gastos de mantenimiento del software, de la siguiente manera:

- El tipo de aplicación
- La novedad del Software
- La disponibilidad del personal
- La vida útil de Software
- Características de Hardware
- La Calidad de diseño del software, construcción, documentación y pruebas

1.5. *Evolución de Software*
[Art88:c1s1.0, s1.1, s1.2, c11s1.1, s1.2; Leh97:108-124], (Bel72)

Lehman abordó por primera vez el mantenimiento del software y la evolución de los sistemas en 1969. Durante un período de veinte años, sus investigaciones condujeron a la formulación de ocho "Leyes de Evolución". [Leh97] Las principales conclusiones incluyen el hecho de que el mantenimiento es una novedad evolutiva y que ayuda a decisiones de mantenimiento entendiendo lo que le pasa a los sistemas (y el software) con el tiempo. Otros declaran que el mantenimiento es un desarrollo continuado, pero hay una entrada extra- la existencia de software grande que nunca se completa y sigue desarrollándose. Como esto se desarrolla, se hace más complejo a no ser que alguna acción sea tomada para reducir esta complejidad.

Ya que el software demuestra el comportamiento regular y las tendencias, estos pueden ser medidos. Se han realizado tentativas de desarrollar modelos proféticos para estimar el esfuerzo de mantenimiento, como resultado se han desarrollado instrumentos de dirección. [Art88], (Bel72).

1.6. *Las categorías de Mantenimiento*
[Art88:c1s1.2; Lie78; Dor02:v1c9s1.5; ieee1219-98:s3.1.1, s3.1.2, s3.1.7, 1.7 un; iso14764-99:s4.1, s4.3, s4.10, s4.11, s6.2; Pig97:c2s2.3]

Lientz y Swanson al principio definieron tres categorías de mantenimiento: correctivo, adaptativo, y perfectivo. [Lie78;
IEEE1219-98] Esta definición más tarde fue puesta al día en el Estándar para el Mantenimiento de Software de la ingeniería de Software, ISO/IEC 14764 para incluir cuatro categorías, así:

- Mantenimiento Correctivo: Modificación reactiva de un producto de software realizado después de entrega para corregir problemas descubiertos,
- Mantenimiento Adaptativo: Modificación de un producto de software realizado después de entrega para guardar (mantener) un producto de software utilizable en un ambiente cambiado o que se cambia.
- Mantenimiento: Perfectivo: Modificación de un software después de la entrega de los productos para mejorar el rendimiento o su mantenibilidad
- Mantenimiento preventivo: Modificación de un software después de la entrega de productos para detectar y corregir fallos latentes en el producto de software antes de que se conviertan en fallos reales.

ISO/IEC 14764 clasifica el mantenimiento adaptativo y perfectivo como mejoras. Agrupa el mantenimiento correctivo y preventivo en una categoría de corrección, como se muestra en la tabla 1. El mantenimiento preventivo, la categoría más reciente, a menudo es realizado sobre productos de software donde la seguridad es crítica.

	Corrección	**Mejoras**
Proactiva	Preventivo	Perfectivo
Reactiva	Correctivo	Adaptativo

Tabla 1: Categorías del Mantenimiento del Software

2. Los Problemas claves en el Mantenimiento de Software

Un número de problemas claves deben ser tratados para asegurar el mantenimiento eficaz de software. Es importante que entienda que el mantenimiento del software provee desafíos de dirección para los ingenieros del software. La tentativa de encontrar un defecto en las 500K líneas de código del software que el ingeniero de software no desarrolló es un buen ejemplo. Asimismo competir con los desarrolladores del software por los recursos es una batalla constante.
La planificación para una futura liberación, cifrando la siguiente liberación y enviando parches de la emergencia para la liberación corriente, también crea un desafío. La sección siguiente presenta técnicas y cuestiones de dirección relacionadas con el mantenimiento del software. Han sido agrupadas bajo los títulos siguientes:

- Cuestiones Técnicas
- Cuestiones de Dirección
- Coste estimado y
- Medidas

2.1. *Cuestiones Técnicas*

2.1.1. Entendimiento limitado.
[Dor02:v1c9s1.11.4; Pfl01:c11s11.3; Tak97:c3]

El entendimiento limitado se refiere a como rápidamente un ingeniero de software puede entender donde hacer un cambio o una corrección en el software que este individuo no desarrolló. La investigación indica que aproximadamente el 40 % al 60 % del esfuerzo de mantenimiento está dedicado al entendimiento del software para ser modificado. Así, la comprensión del software es de gran interés por parte de los ingenieros de software.
La comprensión es más difícil en la representación orientada por texto, en el código original, por ejemplo, donde es a menudo difícil de remontar la evolución de software por sus liberaciones/versiones si los cambios no son documentados y cuando los desarrolladores no están disponibles para explicarlo, que es a menudo el caso.

2.1.2. Pruebas
[Art88:c9; Pfl01:c11s11.3]

El coste de repetir pruebas sobre un pedazo principal de software puede ser significativo en términos de tiempo y dinero.
Las pruebas de regresión, las nuevas pruebas selectivas de un software o el componente para verificar que las modificaciones no han causado efectos no planeados, son importantes para el mantenimiento. Existe también el desafío de coordinar pruebas cuando los miembros del equipo de mantenimiento trabajan sobre problemas diferentes al mismo tiempo. [Plf01] Cuando el software realiza funciones críticas, puede ser imposible llevarlo fuera de línea para probar.
Las pruebas de software KA proporciona la información adicional y se refiere a dicha materia en su punto 2.2.6 pruebas de Regresión.

2.1.3. Análisis de impacto
[Art88:c3; Dor02:v1c9s1.10; Pfl01: C11s11.5]

El análisis de Impacto describe como conducir, con rentabilidad, un análisis completo del impacto de un cambio del software existente. Los mantenedores deben poseer un conocimiento íntimo de la estructura del software y el contenido [Pfl01]. Usan aquel conocimiento para realizar el análisis de impacto, que identifica todos los sistemas y los productos de software afectados por un cambio de software solicitado y se desarrolla una estimación de los recursos para llevar a cabo el cambio. [Art88] Además, el riesgo de hacer el cambio es determinado.
La petición de cambio, a veces llama a una petición de modificación (MR) y a menudo llama un informe del problema (PR), primero debe ser analizada y traducida en términos de software. Es realizado después de que una petición de cambio entra en el proceso de dirección de configuración de software. Arthur [Art88] declara que los objetivos de análisis de impacto son:

- La determinación del alcance de un cambio para planificar y poner en práctica el trabajo

- El desarrollo de las estimaciones exactas de recursos que tuvo que realizar el trabajo
- El análisis del perdidas/beneficio del cambio solicitado
- La comunicación de la complejidad de un cambio dado

La severidad de un problema a menudo se usa para decidir cómo y cuando un problema será fijado. El ingeniero del software entonces identifica los componentes afectados. Proporcionando varias soluciones potenciales y luego una recomendación hecha en cuanto al mejor curso de acción.
El software diseñado con la capacidad de mantenimiento en mente facilita enormemente el análisis de impacto. Para más información en la Dirección de Configuración de Software KA.

2.1.4. Capacidad de mantenimiento [ISO14764-99:s6.8s6.8.1; Pfl01: C9s9.4; ¿Pig97:c16]

¿Cómo uno promueve y lleva a cabo cuestiones de capacidad de mantenimiento durante el desarrollo? El IEEE [IEEE610.12-90] define la capacidad de mantenimiento como la facilidad por la cual el software puede ser mantenido, mejorado, adaptado, o corregido para satisfacer exigencias especificadas. ISO/IEC define la capacidad de mantenimiento como una de las características de calidad (ISO9126-01).

Las subcaracterísticas de capacidad de mantenimiento deben ser especificadas, repasadas, y controladas durante las actividades de desarrollo de software para reducir costes de mantenimiento. Si esto se hace satisfactoriamente, la capacidad de mantenimiento del software se mejorará. Esto es a menudo difícil de alcanzar porque las subcaracterísticas de capacidad de mantenimiento no son un foco importante durante el proceso de desarrollo de software. Los desarrolladores están preocupados por muchas otras cosas y a menudo desatienden las exigencias del mantenedor. Esto a su modo, y a menudo hace que pueda causar una carencia de documentación de sistema, que es una causa principal de dificultades en la comprensión de programa y el análisis de impacto. También se observa que la presencia de los procesos, técnicas, e instrumentos ayudan a mejorar la capacidad de mantenimiento de un sistema.

2.2. *Cuestiones de dirección*

2.2.1. Alineación con objetivos de organización
[Ben00:c6sa; Dor02:v1c9s1.6]

Los objetivos de organización describen como demostrar el rendimiento de la inversión de actividades de mantenimiento de software.
Bennett [Ben00] declara que " el desarrollo de software inicial es por lo general a base de proyecto, con una escala de tiempo definida y el presupuesto.

El énfasis principal debe entregar a tiempo y dentro del presupuesto para encontrar necesidades de usuario. Al contrario el mantenimiento de software a menudo tiene el objetivo de ampliar la vida de software tanto como sea posible. Además, pueden conducirlo la necesidad de encontrar la demanda de usuario de actualizaciones de software y mejoras.
En ambos casos, el rendimiento de la inversión es mucho menos claro, de modo que la opinión en el nivel de dirección sea a menudo de una actividad principal que consume recursos significativos sin la ventaja clara cuantificable para la organización. "
2.2.2. Proveer de personal
[Dek92:10-17; Dor02:v1c9s1.6; Par86: C4s8-c4s11] (Lie81)

El proveer de personal se refiere a como atraer y mantener el personal de mantenimiento de software. El mantenimiento no es visto a menudo como un trabajo encantador. Deklava proporciona una lista de problemas relacionados con el personal, basados en datos de revisión. [Dek92] Por consiguiente, el personal de mantenimiento de software con frecuencia es visto como " ciudadanos de segunda clase " (Lie81) y la moral por lo tanto se ve afectada. [Dor02]

2.2.3. Proceso
[Pau93; Ben00:c6sb; Dor02:v1c9s1.3]

El proceso de Software es un juego de actividades, métodos, prácticas, y las transformaciones que pueblan el empleo para desarrollar y mantener el software y los productos asociados. [Pau93] En el nivel de proceso, las actividades de mantenimiento de software comparten mucho en común con el desarrollo de software (por ejemplo, la dirección de configuración de software es una actividad crucial en ambos). [Ben00] el Mantenimiento también requiere varias actividades que no son encontradas en el desarrollo de software (mirar la sección 3.2 sobre actividades únicas para detalles). Estas actividades presentan desafíos a la dirección. [Dor02]

2.2.4. Los aspectos de organización de mantenimiento
[Pfl01:c12s12.1-c12s12.3; Par86:c4s7; Pig97:c2s2.5; Tak97:c8]

Los aspectos de organización describen como identificar cual organización y/o la función que serán responsables del mantenimiento de software. El equipo que desarrolla el software no necesariamente está asignado a mantener el software una vez que es operacional.
En la decisión donde la función de mantenimiento de software será asignada, las organizaciones de ingeniería del software, por ejemplo, pueden quedarse con el desarrollador original o ir a un equipo distinto (o mantenedor). A menudo, la opción mantenedor es

elegida para asegurar que se desarrolla el software para satisfacer necesidades de usuario. Ya que hay muchos pros y los contras a cada una de estas opciones [Par86, Pig97], la decisión debería ser hecha en una base de caso por caso. Es importante que la delegación o la asignación de la responsabilidad de mantenimiento sea a un grupo solo o a una persona [Pig97], independientemente de la estructura de la organización.

2.2.5. Externalización

[Dor02:v1c9s1.7; Pig97:c9s9.1, s9.2], (Car94;McC02)

La externalización del mantenimiento forma parte de una gran industria.
Las grandes corporaciones externalizan las carteras enteras de sistemas de software, incluyendo el mantenimiento de software. Más a menudo, la opción de externalización es elegida para software menos crítico, como las empresas no están dispuestas a perder el control del software usado en su negocio esencial. Carey (Car94) relata que unos externalizarán sólo si ellos pueden encontrar los modos de mantener el control estratégico. Sin embargo, las medidas de control son difíciles de encontrar. [Dor02] Otro desafío identificado es la transición del software al subcontratado. [Pig97]

2.3 Estimación del coste del mantenimiento.

Los ingenieros de Software deben entender las diferentes categorías de mantenimiento de software, para dirigir la pregunta de estimar el coste de mantenimiento de software. Para planificar objetivos, estimar gastos es un aspecto importante de mantenimiento de software.

2.3.1. Valoración de coste

[Art88:c3; Boe81:c30; Jon98:c27; Pfl01:c11s11.3;Pig97:c8]

Este punto fue mencionado en el subasunto 2.1.3, el Análisis de Impacto, aquel análisis de impacto identifica todos los sistemas y los productos de software afectados por la solicitud de un cambio de software y por tanto se desarrolla una estimación de los recursos que tuvo que utilizar para lograr aquel cambio.
[Art88] las estimaciones de Coste de mantenimiento están afectadas por muchos factores técnicos y no técnicos. ISO/IEC14764 declara que " los dos accesos más populares a la estimación de recursos para el mantenimiento de software son el empleo de modelos paramétricos y el empleo de experiencia " [ISO14764-99:s7.4.1]. Lo q más a menudo se usa es una combinación de ambas.

2.3.2. Modelos paramétricos

[Ben00:s7; Boe81:c30; Jon98:c27; Pfl01:c11s11.3]

Algunos trabajos se han emprendido en la aplicación del coste paramétrico que modela al mantenimiento de software. [Boe81, Ben00] la importancia es que los datos de proyectos pasados son necesarios en el uso de los modelos. Jones [Jon98] habla de todos los aspectos sobre estimar gastos, incluyendo puntos de función (IEEE14143.1-00), y proporciona un capítulo detallado sobre la valoración de mantenimiento.

2.3.3. Experiencia

[ISO14764-00:s7, s7.2, s7.2.1, s7.2.4; Pig97:c8;Sta94]

La Experiencia, en forma de juicio de expertos (usando la técnica Delphi, por ejemplo), analogías, y una estructura de interrupción de trabajo, deberían ser usados para aumentar los datos de los modelos paramétricos. Claramente el mejor acercamiento a la valoración de mantenimiento es el de combinar datos empíricos y experiencia. Deberían proporcionar estos datos como consecuencia del uso de un programa de medida.

2.4. *Medidas de Mantenimiento del Software*

[IEEE1061-98:A.2; Pig97:c14s14.6; Gra87; Tak97:C6s6.1-c6s6.3]

Grady y Caswell [Gra87] hablan del establecimiento de un programa de medida de software extensamente corporativo, en el cual son descritas las formas de medida de mantenimiento de software y la colección de datos. El Software Práctico y la Medida de Sistemas (PSM) describen un proceso de medida que es usado por muchas organizaciones y es bastante práctico. [McG01].

Hay medidas de software que son comunes a todos los esfuerzos, las categorías siguientes se han identificado por el Instituto de Ingeniería de Software (SEI): tamaño; esfuerzo; programa; y calidad. [Pig97] Estas medidas constituyen un buen punto de partida para el mantenedor. La discusión de proceso y la medida de producto están presentes en el Proceso de Ingeniería de Software KA. El programa de medida de software está descrito en la Dirección de Ingeniería del Software KA.

2.4.1. Medidas Específicas

[Car90:s2-s3; IEEE1219-98:Table3; sta94:p239-249] Abran [Abr93]

Presentan técnicas de prueba de referencia internas para comparar diferentes organizaciones de mantenimiento internas.
El mantenedor debe determinar qué medidas son apropiadas por la organización en cuestión. [Ieee1219-98; ISO9126-01; Sta94] sugiere que medidas son más específicas a programas de medida de mantenimiento de software.

La lista incluye un número de medidas para cada una de las cuatro subcaracterísticas de capacidad de mantenimiento:

- Analizar: Las medidas del esfuerzo del mantenedor o recursos gastados en tentativa de

diagnosticar carencias o causas de fracaso, o en identificación de partes para ser modificadas
- Variabilidad: Las medidas del esfuerzo del mantenedor asociado con realización de una modificación especificada
- Estabilidad: Las medidas del comportamiento inesperado de software, incluyendo lo encontrado durante las pruebas
- Testeabilidad: Las medidas del esfuerzo del mantenedor y usuarios en la tentativa de probar el software modificado.

Ciertas medidas de la capacidad de mantenimiento de software pueden obtenerse usando instrumentos comerciales disponibles. (Lag96; Apr00)

3. El Proceso de Mantenimiento

La subárea Proceso de Mantenimiento proporciona referencias y normas para poner en práctica el proceso de mantenimiento de software. El punto de Actividades de Mantenimiento diferencia el mantenimiento del desarrollo y muestra su relación con otras actividades de la ingeniería del software.
La necesidad del proceso de ingeniería de software está bien documentada. Los modelos CMMI se aplican a procesos de mantenimiento de software, y son similares a los procesos de los desarrolladores. [SEI01] los modelos de Capacidad de Madurez del Mantenimiento de Software que se dirigen los procesos únicos de mantenimiento de software son descritos en (Apr03, Nie02, Kaj01).

3.1. *Procesos de Mantenimiento*

[IEEE1219-98:s4; ISO14764-99:s8; ieee12207.0-96:s5.5; Par86:c7s1; Pig97:c5; Tak97:c2]

Los procesos de Mantenimiento proporcionan actividades necesarias y entradas/salidas detalladas a aquellas actividades, y son descritos en normas de mantenimiento de software IEEE 1219 y ISO/IEC 14764.
El modelo de proceso de mantenimiento descrito en el Estándar para el Mantenimiento de Software (IEEE1219) comienza con el esfuerzo de mantenimiento de software durante la etapa de postentrega y habla de artículos como la planificación para el mantenimiento. Aquel proceso es representado en la Figura(el Número) 2.

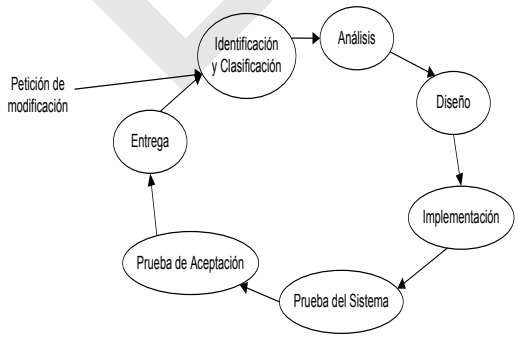


Figura 2 las Actividades de Proceso de Mantenimiento IEEE1219-98

ISO/IEC 14764 [ISO14764-99] es una elaboración del proceso de mantenimiento IEEE/EIA 12207.0-96. Las actividades del proceso de mantenimiento ISO/IEC son similares a aquellas del IEEE, pero están agregadas de manera diferente. Las actividades de proceso de mantenimiento desarrolladas por ISO/IEC se muestran en la Figura 3.

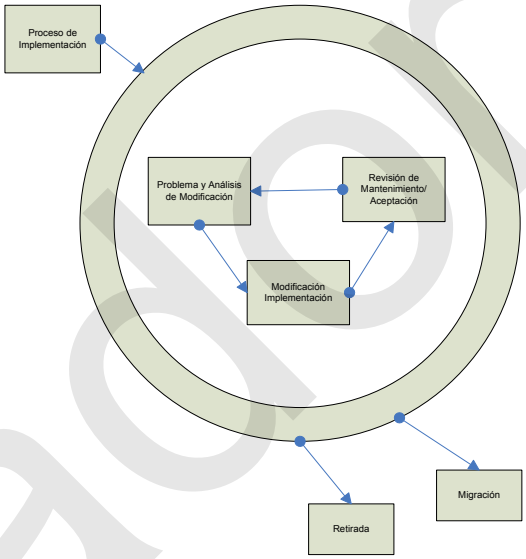


Figura 3 ISO/IEC 14764-00 Proceso de Mantenimiento de Software

Cada uno de las actividades primarias de mantenimiento de software ISO/IEC 14764 son las siguientes:
- La Puesta en práctica de Proceso
- El Problema y el Análisis de Modificación
- La Puesta en práctica de Modificación
- La Revisión/Aceptación de Mantenimiento
- La Migración
- El Retiro de Software

Takang y Grubb [Tak97] proporcionan una historia de modelos de proceso de mantenimiento que conducen hasta el desarrollo del IEEE y modelos de proceso de ISO/IEC. Parikh [Par86] también da una descripción buena de un proceso de mantenimiento genérico. Recientemente, han surgido metodologías ágiles que promueven procesos ligeros. Esta exigencia surge de la demanda cada vez mayor de la vuelta rápida de servicios de mantenimiento. Algunos experimentos con el mantenimiento Extremo son presentados en (Poo01).

3.2. Actividades de Mantenimiento

Como ya ha notado, muchas actividades de mantenimiento son similares a aquellas de desarrollo de software. Los mantenedores realizan el análisis, el diseño, la codificación, pruebas, y la documentación.

Ellos deben rastrear exigencias en sus actividades tal cual hechas en el desarrollo, y la documentación de actualización como el cambio de líneas de fondo. ISO/IEC14764 recomienda que, cuando un mantenedor se refiera a un proceso de desarrollo similar, él debe adaptarlo para encontrar sus necesidades específicas [ISO14764-99:s8.3.2.1, 2]. Sin embargo, para el mantenimiento de software, algunas actividades implican procesos únicos al mantenimiento de software.

3.2.1. Actividades únicas
[Art88:c3; Dor02:v1c9s1.9.1; ieee1219-98:s4.1, s4.2; ISO14764-99:s8.2.2.1, s8.3.2.1; Pfl01:c11s11.2]

Hay un número de procesos, actividades, y prácticas que son únicas al mantenimiento de software, por ejemplo:

- Transición: una secuencia controlada y coordinada de actividades durante las cuales el software es transferido cada vez más del desarrollador al mantenedor [Dek92, Pig97]
- La Aceptación/Rechazo de Petición de Modificación: el trabajo de petición de modificación sobre un cierto tamaño/esfuerzo/complejidad puede ser rechazado por mantenedores y desviado a un desarrollador [Dor02], (Apr01)
- La petición de Modificación y el Escritorio de Ayuda de Informe de Problema: una función de apoyo de usuario final que provoca la evaluación, la ordenación, y de presupuesto de solicitud de modificación [Ben00]
- El Análisis de Impacto (mirar la sección 2.1.3 para detalles)
- El Apoyo de Software: ayuda y aconseja a usuarios que solicitan información (por ejemplo, reglas de gestión, validación, datos que quieren decir y ad hoc solicita/hace un informe)
- Los Acuerdos de Nivel de Servicio (SLAs) y los contratos de mantenimiento especializados (específicos de dominio) que son responsabilidad de los mantenedores (Apr01)

3.2.2. Apoyando actividades
[IEEE1219-98:A.7, 11 un ; IEEE12207.0-96:c6, c7;ITI01; Pig97:c10s10.2, c18]; (Kaj01)

Los mantenedores también puede realizar actividades de apoyo, como la planificación de mantenimiento de software, la dirección de configuración de software, la verificación y la validación, la garantía de calidad de software, revisiones, revisiones de cuentas, y el entrenamiento de usuario.
Otra actividad de apoyo, entrenamiento del mantenedor, también es necesaria. [Pig97; IEEE12207.0-96] (Kaj01)

3.2.3. Actividad de planificación de mantenimiento.
[IEEE1219-98:A.3; ISO14764-99:s7; ITI01;Pig97:c7, c8]

Una actividad importante para el mantenimiento de software es la planificación, y los mantenedores deben dirigir las cuestiones asociadas con un número de perspectivas de planificación:

- La planificación de las actividades (el nivel de organización)
- La planificación de Mantenimiento (el nivel de transición)
- La planificación de Liberación/versión (el nivel de software)
- La planificación de petición de cambio de software

En el nivel de petición individual, la planificación es realizada durante el análisis de impacto (irse al subasunto 2.1.3 Análisis de Impacto para detalles). La liberación/versión que planifica la actividad requiere que el mantenedor [ITI01]:

- Reúna las fechas de disponibilidad de las solicitudes de los individuos.
- Esté de acuerdo con usuarios sobre el contenido de liberaciones/versiones subsecuentes
- Identifique conflictos potenciales y desarrolle alternativas
- Evalúe el riesgo de una liberación dada y desarrolle un plan echarse atrás en caso de los problemas deberían surgir
- Informe a todos los tenedores de apuestas

Mientras que los proyectos de desarrollo de software típicamente pueden durar a partir de algunos meses a algunos años, la fase de mantenimiento por lo general dura muchos años. La fabricación de las estimaciones de recursos es un elemento clave de planificación de mantenimiento. Los recursos deberían ser incluidos en los presupuestos de proyecto que planifican los desarrolladores. La planificación de mantenimiento de software debería comenzar con la decisión de desarrollar un nuevo sistema y debería considerar objetivos de calidad (IEEE1061-98). Un documento de concepto debería ser desarrollado, seguido por un plan de mantenimiento.
El documento de concepto para el mantenimiento [iso14764-99:s7.2] debe dirigirse a:

- el alcance del mantenimiento de software.
- la adaptación del mantenimiento de software
- la identificación de la organización de mantenimiento de software
- una estimación de costes de mantenimiento de software

El siguiente paso debe desarrollar el correspondiente plan de mantenimiento de software. Este plan debería estar preparado durante el desarrollo de software, y debería especificar como los usuarios solicitarán modificaciones de software o relatarán problemas.
El plan de mantenimiento de software [Pig97] es dirigido en IEEE 1219 [IEEE1219-98] Y ISO/IEC 14764.
[ISO14764-99] ISO/IEC14764 proporciona directrices para un plan de mantenimiento.

Finalmente, en el nivel más alto, la organización de mantenimiento tendrá que conducir actividades de planificación de business (recursos presupuestarios, financieros, y humanos) justo como todas las otras divisiones de la organización. El conocimiento de dirección requerido para hacer esto se puede encontrar en el capítulo de Disciplinas Relacionadas de Ingeniería del Software.

3.2.4. Dirección de configuración de software
[Art88:c2, c10; IEEE1219-98:A.11; ieee12207.0-96:s6.2; Pfl01:c11s11.5; Tak97:c7]

El Estándar IEEE para el Mantenimiento de Software, IEEE 1219 [IEEE1219-98], describe la dirección de configuración de software como un elemento crítico del proceso de mantenimiento. Los procedimientos de dirección de configuración de software deberían asegurar la verificación, la validación, y la revisión de cuentas de cada paso requerido para identificar, autorizar, poner en práctica, y liberar el producto de software.

No es suficiente simplemente con rastrear la Modificación Solicitada o los informes de Problema. El producto de software y cualquier cambio hecho deben ser controlados. Este control es establecido poniendo en práctica y haciendo cumplir una dirección de configuración de software aprobada (SCM)

El proceso de Dirección de Configuración de Software KA proporciona los detalles de SCM y habla del proceso por el cual los cambios de software solicitado son sometidos, evaluados, y aprobados. SCM para el mantenimiento de software es diferente de SCM para el software de desarrollo en el número de los pequeños cambios que deben ser controlados sobre el software operacional. El proceso de SCM es puesto en práctica por desarrollador y después de un plan de dirección de configuración y procedimientos. Los mantenedores participan en la Pasarela de Control de Configuración para determinar el contenido de la siguiente liberación/versión.

3.2.5. Calidad de software
[Art98:c7s4; IEEE12207.0-96:s6.3; ieee1219-98:A.7; ISO14764-99:s5.5.3.2]

No es suficiente, tampoco, simplemente esperar lo que aumentó la calidad sea resultado del mantenimiento de software. Debe ser planificado y procesos puestos en práctica para apoyar el proceso de mantenimiento. Las actividades y técnicas para la Garantía de calidad de Software (SQA), V*V, revisiones, y revisiones de cuentas deben ser seleccionadas de común acuerdo con todos los otros procesos para alcanzar el nivel deseado de calidad. También le recomiendan que el mantenedor adapte los procesos de desarrollos de software, técnicos y entregables, por ejemplo probando la documentación, y pruebe resultados.

[ISO14764-99] Más detalles pueden ser encontrados en la Calidad de Software KA.

4. Las técnicas para el Mantenimiento

Este subárea introduce algunas técnicas generalmente aceptadas usadas en el mantenimiento de software.

4.1. *Comprensión de Programa*
[Arn92:c14; Dor02:v1c9s1.11.4; Tak97:c3]

Los Programadores gastan un tiempo considerable en la lectura y el entendimiento de programas para poner en práctica los cambios.
Los navegadores de código son instrumentos claves para la comprensión de programa.
La documentación clara y concisa puede ayudar en la comprensión de programa

4.2. Reingeniería
[Arn92:c1, c3-c6; Dor02:v1c9s1.11.4; IEEE1219-98:La b 2], (Fow99)

Reingeniería se define como el examen y la alteración de software para reconstituirlo en una nueva forma, e incluye la puesta en práctica subsecuente de la nueva forma. Dorfman y Thayer [Dor02] declaran que reingeniería es la forma radical de alteración. Otros creen que la reingeniería puede ser usada para cambios menores. Arnold [Arn92] proporciona un compendio comprensivo de puntos, por ejemplo: conceptos, instrumentos y técnicas, estudios de caso, y riesgos y ventajas asociadas con la reingeniería.

4.3. *Ingeniería de revés*
[Arn92:c12; Dor02:v1c9s1.11.3; IEEE1219-98:B.3; Tak97:c4, Hen01]

La ingeniería de Revés es el proceso de analizar el software para identificar los componentes del software y sus relaciones mutuas y crear las representaciones del software en otra forma o en los niveles más altos de abstracción. La ingeniería de revés es pasiva; esto no cambia el software, o causa el nuevo software. Los esfuerzos de la ingeniería del revés producen gráficos de llamada y gráficos de flujo de control del código original. Un tipo de ingeniería de revés es la nueva documentación. Otro tipo es la recuperación de diseño [Dor02]. La nueva factorización es la transformación de programa que reorganiza un programa sin cambiar su comportamiento, y es una forma de revés que procura mejorar la estructura de programa. (Fow99).

Finalmente, la ingeniería de revés de datos ha ganado en importancia durante estos últimos años donde los esquemas lógicos son recuperados de bases de datos físicas. (Hen01)

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Abr 93]	[Arn 92]	[Art88]	[Ben 00]	[Boe 81]	[Car 90]	[Dek 92]	[Dor 97]	[Hen 01]	[IEEE 610.12-90]	[IEEE 1061-98]	[IEEE 1219-98]	[IEEE 12207.0-96]	[ISO 14764-00]	[Jon 98]	[Leh 97]	[Par 86]	[Pfl 01]	[Pig 97]	[Pre 04]	[Sta 04]	[Tak 97]
1.Fundamentos del Mantenimiento del Software																						
1.1 Definiciones y Terminología	63-90			s4								S3.1.12		s6.1								
1.2 Naturaleza del Mantenimiento													s3, s5.5					c11s11.2				
1.3 Necesidad de Mantenimiento																		c11s11.2	C2s2.3			c1
1.4 Mayoría de los Costes de Mantenimiento	63-90																	c11s11.3	c3	C31		
1.5 Evolución del Software					c1s1.0-c1s1.2,c11s1.1 c11s1.2										108-124							
1.6 Categorías de Mantenimiento	63-90				c1s1.2	s5				v1c9s1.5	s3.1.1,s3.1.2, s3.1.7,A.1.7		s4.1,s4.3s4.10, s4.11,s6.2					c2s2.3				
2.Elementos Clave en el Mantenimiento Software																						
2.1 Elementos Técnicos						s6c																
Comprensión Limitada						s11.4				v1c9s1.11.4							c11s11.3				c3	
Prueba					C9												c11s11.3					
Análisis de Impacto					c3	s10.1, s10.2,s10.3				0.1-v1c9s1.1							c11s11.5					
Mantenibilidad		s3				s5, s9.3, s11.4							s6.8, s6.8.1				c9s9.4	c16				
2.2 Gestión de Elementos																						
Alineamiento con Objetivos organizacionales						s6a				v1c9s1.6												
Personal									10-17	v1c9s1.6						c4s11						

4.3 Ingeniería Inversa	4.2 Reingeniería	4.1 Programa de Comprensión	**4. Técnicas para el Mantenimiento**	Calidad del software	Gestión de la configuración del software	Actividad de planificación del mantenimiento	Actividades de soporte	Actividades únicas	3.2 Actividades de Mantenimiento	3.1 Procesos de Mantenimiento	**3. Proceso de Mantenimiento**	Medidas específicas	2.4 Medidas	Experiencia	Modelos de parametrización	Estimación de costes	2.3 Estimación de Costes de Mantenimiento	Subcontratación	Aspectos organizacionales de mantenimiento	Procesos
				63-90				63-90				63-90	63-90							
													A.2							
c12	c1,c3-c6	c14																		
				c7s4	c2,c10			c3								c3				
s9.2, s10.3,s11.2, s11.3,s11.5	s9.2, s11.4, s11.5	s11.4		s7, s8, s9.2,s9.3, s11.4,s11.5	s2, s6b,s9.2, s10.1,s11.4			s6b	s9.1	s8		s9.2	s9.1, s9.2, s10.1		s7	s7				
															c30	c30		s7	s6a, s7	s6b
												s2-s3								
*																				
v1c9s1.1 1.3	v1c9s1.1 1.4	v1c9s1.1 1.4						v1c9s1.9 .1										v1c9s 1.7		v1c9s1.3
B.3	B.2			A.7	A.11	A.3	A.7,A.11	s4.1-s4.2		s4		Tabla 3								
				s6.3	s6.2		c6,c7		s5.5											
				s5.5.3.2		s7		s8.2.2.1, s8.3.2.1		s8				s7,s7.2, s7.2.1, s7.2.4						
															c27	c27				
										c7s1									c4s7	
					c11s11.5			c11s11.2								c11s11.3	c11s11.3			c12s12.1 - c12s12.3	
						c7,c8	c10s10.2 , c18			c5			c14s14.6	c8	c8	c8		c9s9.1- c9s9.2	c2s2.5	
												239-249								
C4		c3			c7					c2			c6s6.1- c6s6.3						c8	

RECOMMENDED REFERENCES FOR SOFTWARE MAINTENANCE

[Abr93] A. Abran and H. Nguyenkim, "Measurement of the Maintenance Process from a Demand-Based Perspective," *Journal of Software Maintenance: Research and Practice*, vol. 5, iss. 2, 1993, pp. 63-90.

[Arn93] R.S. Arnold, *Software Reengineering*: IEEE Computer Society Press, 1993.

[Art98] L.J. Arthur, *Software Evolution: The Software Maintenance Challenge*, John Wiley & Sons, 1988.

[Ben00] K.H. Bennett, "Software Maintenance: A Tutorial," in *Software Engineering*, M. Dorfman and R. Thayer, eds., IEEE Computer Society Press, 2000.

[Boe81] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[Car90] D.N. Card and R.L. Glass, *Measuring Software Design Quality*, Prentice Hall, 1990.

[Dek92] S. Dekleva, "Delphi Study of Software Maintenance Problems," presented at the International Conference on Software Maintenance, 1992.

[Dor02] M. Dorfman and R.H. Thayer, eds., *Software Engineering (Vol. 1 & Vol. 2)*, IEEE Computer Society Press, 2002.

[Gra87] R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice Hall, 1987.

[Hen01] J. Henrard and J.-L. Hainaut, "Data Dependency Elicitation in Database Reverse Engineering," *Proc. of the 5th European Conference on Software Maintenance and Reengineering* (CSMR 2001), IEEE Computer Society Press, 2001.

[IEEE610.12-90] IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

[IEEE1061-98] IEEE Std 1061-1998, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE, 1998.

[IEEE1219-98] IEEE Std 1219-1998, *IEEE Standard for Software Maintenance*, IEEE, 1998.

[IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/ IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

[ISO9126-01] ISO/IEC 9126-1:2001, *Software Engineering-Product Quality-Part 1: Quality Model*, ISO and IEC, 2001.

[ISO14764-99] ISO/IEC 14764-1999, *Software Engineering-Software Maintenance*, ISO and IEC, 1999.

[ITI01] IT Infrastructure Library, "Service Delivery and Service Support," Stationary Office, Office of Government of Commerce, 2001.

[Jon98] T.C. Jones, *Estimating Software Costs*, McGraw- Hill, 1998.

[Leh97] M.M. Lehman, "Laws of Software Evolution Revisited," presented at EWSPT96, 1997.

[Lie78] B. Lienz, E.B. Swanson, and G.E. Tompkins, "Characteristics of Applications Software Maintenance," *Communications of the ACM*, vol. 21, 1978.

[Par86] G. Parikh, *Handbook of Software Maintenance*, John Wiley & Sons, 1986.

[Pfl01] S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001.

[Pig97] T.M. Pigoski, *Practical Software Maintenance: Best Practices for Managing your Software Investment*, first ed., John Wiley & Sons, 1997.

[Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, sixth ed., McGraw-Hill, 2004.

[SEI01] Software Engineering Institute, "Capability Maturity Model Integration, v1.1," CMU/SEI-2002-TR-002, ESC-TR-2002-002, December 2001.

[Sta94] G.E. Stark, L.C. Kern, and C.V. Vowell, "A Software Metric Set for Program Maintenance Management," *Journal of Systems and Software*, vol. 24, iss. 3, March 1994.

[Tak97] A. Takang and P. Grubb, *Software Maintenance Concepts and Practice*, International Thomson Computer Press, 1997.

APPENDIX A. LIST OF FURTHER READINGS

(Abr93) A. Abran, "Maintenance Productivity & Quality Studies: Industry Feedback on Benchmarking," presented at the Software Maintenance Conference (ICSM93), 1993.

(Apr00) A. April and D. Al-Shurougi, "Software Product Measurement for Supplier Evaluation," presented at FESMA2000, 2000.

(Apr01) A. April, J. Bouman, A. Abran, and D. Al-Shurougi, "Software Maintenance in a Service Level Agreement: Controlling the Customer's Expectations," presented at European Software Measurement Conference, 2001.

(Apr03) A. April, A. Abran, and R. Dumke, "Software Maintenance Capability Maturity Model (SM-CMM): Process Performance Measurement," presented at 13th International Workshop on Software Measurement (IWSM 2003), 2003.

(Bas85) V.R. Basili, "Quantitative Evaluation of Software Methodology," presented at First Pan-Pacific Computer Conference, 1985.

(Bel72) L. Belady and M.M. Lehman, "An Introduction to Growth Dynamics," *Statistical Computer Performance Evaluation*, W. Freiberger, ed., Academic Press, 1972.

(Ben00) K.H. Bennett and V.T. Rajlich, "Software Maintenance and Evolution: A Roadmap," *The Future of Software Engineering*, A. Finklestein, ed., ACM Press, 2000.

(Bol95) C. Boldyreff, E. Burd, R. Hather, R. Mortimer, M. Munro, and E. Younger, "The AMES Approach to Application Understanding: A Case Study," presented at the International Conference on Software Maintenance, 1995.

(Boo94) G. Booch and D. Bryan, *Software Engineering with Ada*, third ed., Benjamin/Cummings, 1994.

(Cap94) M.A. Capretz and M. Munro, "Software Configuration Management Issues in the Maintenance of Existing Systems," *Journal of Software Maintenance: Research and Practice*, vol. 6, iss. 2, 1994.

(Car92) J. Cardow, "You Can't Teach Software Maintenance!" presented at the Sixth Annual Meeting and Conference of the Software Management Association, 1992.

(Car94) D. Carey, "Executive Round-Table on Business Issues in Outsourcing - Making the Decision," *CIO Canadá*, June/July 1994.

(Dor97) M. Dorfman and R.H. Thayer, eds., "Software Engineering," IEEE Computer Society Press, 1997.

(Dor02) M. Dorfman and R.H. Thayer, eds., *Software Engineering (Vol. 1 & Vol. 2)*, IEEE Computer Society Press, 2002.

(Fow99) M. Fowler et al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

(Gra87) R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice Hall, 1987.

(Gra92) R.B. Grady, *Practical Software Metrics for Project Management and Process Management*, Prentice Hall, 1992.

(Jon91) C. Jones, *Applied Software Measurement*, McGraw-Hill, 1991.

(Kaj01) M. Kajko-Mattson, "Motivating the Corrective Maintenance Maturity Model (Cm3)," presented at Seventh International Conference on Engineering of Complex Systems, 2001.

(Kaj01a) M. Kajko-Mattson, S. Forssander, and U. Olsson, "Corrective Maintenance Maturity Model: Maintainer's Education and Training," presented at International Conference on Software Engineering, 2001.

(Kho95) T.M. Khoshgoftaar, R.M. Szabo, and J.M. Voas, "Detecting Program Module with Low Testability," presented at the International Conference on Software Maintenance-1995, 1995.

(Lag96) B. Laguë and A. April, "Mapping for the ISO9126 Maintainability Internal Metrics to an Industrial Research Tool," presented at SESS, 1996.

(Leh85) M.M. Lehman and L.A. Belady, *Program Evolution: Processes of Software Change*, Academic Press, 1985.

(Leh97) M.M. Lehman, "Laws of Software Evolution Revisited," presented at EWSPT96, 1997.

(Lie81) B.P. Lientz and E.B. Swanson, "Problems in Application Software Mainteanance," *Communications of the ACM*, vol. 24, iss. 11, 1981, pp. 763-769.

(McC02) B. McCracken, "Taking Control of IT Performance," presented at InfoServer LLC, 2002.

(Nie02) F. Niessink, V. Clerk, and H. v. Vliet, "The IT Capability Maturity Model," release L2+3-0.3 draft, 2002, available at http://www.itservicecmm.org/doc/itscmm-123-0.3.pdf.

(Oma91) P.W. Oman, J. Hagemeister, and D. Ash, "A Definition and Taxonomy for Software Maintainability," University of Idaho, Software Engineering Test Lab Technical Report 91-08 TR, November 1991.

(Oma92) P. Oman and J. Hagemeister, "Metrics for Assessing Software System Maintainability," presented at the International Conference on Software Maintenance '92, 1992.

(Pig93) T.M. Pigoski, "Maintainable Software: Why You

Want It and How to Get It," presented at the Third

Software Engineering Research Forum - November 1993,
University of West Florida Press, 1993.
(Pig94) T.M. Pigoski, "Software Maintenance," *Encyclopedia of Software Engineering*, John Wiley & Sons, 1994.
(Pol03) M. Polo, M. Piattini, and F. Ruiz, eds., "Advances
in Software Maintenance Management: Technologies and
Solutions," Idea Group Publishing, 2003.
(Poo00) C. Poole and W. Huisman, "Using Extreme Programming in a Maintenance Environment," *IEEE Software*, vol. 18, iss. 6, November/December 2001, pp. 42-50.
(Put97) L.H. Putman and W. Myers, "Industrial Strength Software - Effective Management Using Measurement," 1997.
(Sch99) S.R. Schach, *Classical and Object-Oriented Software Engineering with UML and C++*, McGraw-Hill,
1999.
(Sch97) S.L. Schneberger, *Client/Server Software Maintenance*, McGraw-Hill, 1997.
(Sch87) N.F. Schneidewind, "The State of Software Maintenance," *Proceedings of the IEEE*, 1987.
(Som96) I. Sommerville, *Software Engineering*, fifth ed., Addison-Wesley, 1996.
(Val94) J.D. Vallett, S.E. Condon, L. Briand, Y.M. Kim, and V.R. Basili, "Building on Experience Factory for Maintenance," presented at the Software Engineering Workshop, Software Engineering Laboratory, 1994.

APPENDIX A. LIST OF STANDARDS

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

(IEEE1061-98) IEEE Std 1061-1998, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE, 1998.

(IEEE1219-98) IEEE Std 1219-1998, *IEEE Standard for Software Maintenance*, IEEE, 1998.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology - Software Life Cycle Processes*, IEEE, 1996.

(IEEE14143.1-00) IEEE Std 14143.1-2000// ISO/IEC14143-1:1998, *Information Technology - Software Measurement-Functional Size Measurement - Part 1: Definitions of Concepts*, IEEE, 2000.

(ISO9126-01) ISO/IEC 9126-1:2001, *Software Engineering-Product Quality - Part 1: Quality Model*, ISO and IEC, 2001.

(ISO14764-99) ISO/IEC 14764-1999, *Software Engineering - Software Maintenance*, ISO and IEC, 1999.

(ISO15271-98) ISO/IEC TR 15271:1998, *Information Technology - Guide for ISO/IEC 12207, (Software Life Cycle Process)*, ISO and IEC, 1998. [Abr93]

CAPÍTULO 7
GESTIÓN DE CONFIGURACIÓN DEL SOFTWARE

ACRÓNIMOS

CCB	Tarjeta de Control de la Configuración
CM	Gestión de configuración
FCA	Auditoría de la Configuración Funcional
MTBF	Tiempo significativo entre fallos.
PCA	Auditoría de la Configuración Física
SCCB	Consejo de Control de Configuración del Software
SCI	Elemento de configuración de software
SCM	Gestión de la configuración del software
SCMP	Plan de gestión de la configuración del software
SCR	Petición de cambios del software
SCSA	Contabilidad del Estado de la Configuración del Software
SEI/CMMI	Instituto de Ingenieros Software/ Modelo de la Capacidad de Madurez Integrado
SQA	Garantía de calidad del software.
SRS	Especificación de Requisitos Software
USNRC	Comisión Reguladora de Energía Nuclear de los Estados Unidos

INTRODUCCIÓN:

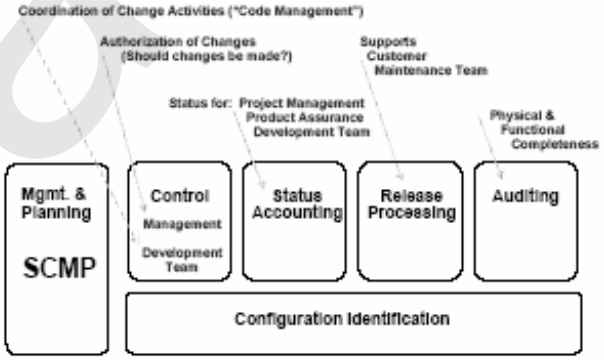
Un sistema se puede definir como una colección de componentes que se organizan con el objetivo de proporcionar una función o conjunto de funciones determinadas (IEEE 610.12-90). La *configuración* de un sistema son las características funcionales y/o físicas del hardware, firmware, software o una combinación de las mismas, según lo dispuesto en la documentación técnica y el resultado obtenido en un producto. (Buc96) También se puede considerar como una colección de versiones específicas de elementos de hardware, firmware o software que se combinan de acuerdo con un proceso de construcción específico para un satisfacer un propósito particular. Por tanto la *gestión de configuración* es la disciplina de identificar la configuración de un sistema en momentos diferentes con el propósito de controlar de una manera sistemática los cambios en la configuración y mantener la integridad y el seguimiento de de los cambios en la configuración durante el ciclo de vida del sistema. (Ber97) Se define formalmente (IEEE610.12-90) como "Una disciplina que establece dirección y seguimiento técnicos y administrativos a: la identificación y documentación de las características funcionales y físicas de un elemento de configuración, toma notas y produce informes de cambios en el proceso y en el estado de implementación y verifica el cumplimiento de los requerimientos especificados."

La *gestión de la configuración del software* (SCM) es un proceso que soporta el ciclo de vida del software (IEEE12207.0-96) que beneficia a la gestión de proyectos, las actividades de desarrollo y mantenimiento, las actividades de garantía y a los clientes y usuarios del producto final.

El concepto de gestión de configuración es aplicable a todos los elementos que se pueden controlar, aunque existen algunas diferencias de implementación entre CM del hardware y CM del software.

La SCM está íntimamente relacionada con la actividad de garantía de calidad del software (SQA). Tal y como se define en el AC de la Calidad del Software, los procesos de la SQA proporcionan la garantía de que los procesos y productos de software en el ciclo de vida del proyecto cumplen los requerimientos especificados gracias a la planificación, la promulgación y ejecución de un conjunto de actividades que proporcionan la confidencia necesaria de que se está teniendo en cuenta la calidad al construir el software. Las actividades de la SCM ayudan a conseguir estos objetivos de la SQA. En el contexto de algunos proyectos (véase, por ejemplo, IEEE730-02), requerimientos específicos de las SQA requieren ciertas actividades de la SCM.

Las actividades de la SCM son: gestión y planificación de los procesos de la SCM, identificación de la configuración del software, control de la configuración del software, responsabilidad del estado de la configuración del software, auditoría de la configuración del software y gestión del lanzamiento y entrega del software.



La figura 1 muestra una representación estilizada de estas actividades.

Figura 1 Actividades SCM

El KA de la Gestión de Configuración del Software se relaciona con el resto de KAs, ya que el objetivo de la configuración del software es el producto construido y usado durante todo el proceso de ingeniería del software.

DIVISIÓN DE PUNTOS PARA LA SCM

2. Gestión del proceso de la SCM

La SCM controla la evolución e integridad de un producto identificando sus elementos, gestionando y controlando los cambios y verificando, guardando y produciendo informes de la información de configuración. Desde la perspectiva del ingeniero de software, la SCM facilita las actividades del desarrollo e implementación de cambios. El éxito de una implementación de la SCM requiere una planificación y gestión cuidadosas. Lo que al mismo tiempo requiere que

se conozca el contexto de organización y las restricciones impuestas en el diseño e implementación del proceso de la SCM.

1.1 Contexto de Organización para la SCM [Ber92 :c4; Dar90:c2; IEEE828-98:c4s2.1]

Para planificar un proceso de la SCM para un proyecto se necesita comprender el contexto de organización y la relación entre los distintos elementos de la organización. La SCM interacciona con otras actividades o elementos de la organización.

Los elementos de la organización responsables de los procesos de soporte de la ingeniería del software se pueden estructurar de diferentes formas. Aunque la responsabilidad de realizar algunas de las tareas de la SCM se podría asignar a otra de las partes de la organización como por ejemplo la organización de desarrollo, normalmente es un elemento definido de la organización o un individuo especialmente designado quien tiene la responsabilidad general de la SCM.

El software se desarrolla frecuentemente como una parte de un sistema mayor que contiene elementos de hardware y firmware. En ese caso, las actividades de la SCM suceden en paralelo con las actividades de CM del hardware y firmware y debe ser consistente con la CM del sistema. Buckley [Buc96:c2] describe la SCM en este contexto. Tenga en cuenta que el firmware contiene hardware y software, así que son aplicables los conceptos de CM de ambos, hardware y software.

La SCM puede interactuar con la actividad de la garantía de la calidad de la organización en lo que se refiere a temas como la gestión de registros y de elementos no válidos. Respecto a gestión de registros, algunos elementos bajo el control de la SCM podrían ser también registros del proyecto, dependiendo del programa de garantía de calidad de la organización. Normalmente la gestión de elementos no válidos es responsabilidad de la actividad de garantía de la calidad; sin embargo, la SCM pude ayudar mediante el seguimiento e informes de elementos de configuración del software que caigan en esta categoría.

Quizás su relación más cercana sea con las organizaciones de desarrollo de software y mantenimiento.

Muchas de las tareas de control de configuración del software se realizan en este contexto. Frecuentemente las mismas herramientas proporcionan soporte para el desarrollo, mantenimiento y la SCM.

1.2 Restricciones y Consejos para el proceso de la SCM [Ber92:c5; IEEE828-98:c4s1,c4s2.3; Moo98]

Las restricciones y consejos para el proceso de la SCM pueden venir de diferentes fuentes. Las normas y procedimientos definidos a nivel corporativo o de la organización pueden tener influencia o prescribir el diseño e implementación de los procesos de la SCM en un determinado proyecto. Además, el contrato entre el proveedor y el cliente podría contener estipulaciones que afecten los procesos de la SCM. Por ejemplo, podría ser necesaria una auditoría de configuración o podría ser necesario poner ciertos elementos bajo el control de la CM. Cuerpos de regulación externos podrían imponer restricciones a productos de software que se vayan a desarrollar cuando estos puedan afectar potencialmente a la seguridad pública (véase, por ejemplo, USNRC1.169-97). Finalmente, el proceso del ciclo de vida del software elegido para un proyecto de software en particular y las herramientas elegidas para la implementación de dicho software, afectan el diseño e implementación de los procesos de la SCM. [Ber92]

Las "mejores prácticas", como se reflejan en los estándares de la ingeniería del software publicados por varias organizaciones de estándares, se pueden usar como consejo para el diseño e implementación de un proceso de la SCM. Moore [Moo98] proporciona una guía para dichas organizaciones y sus estándares. Las mejores prácticas se reflejan también en modelos de mejora del

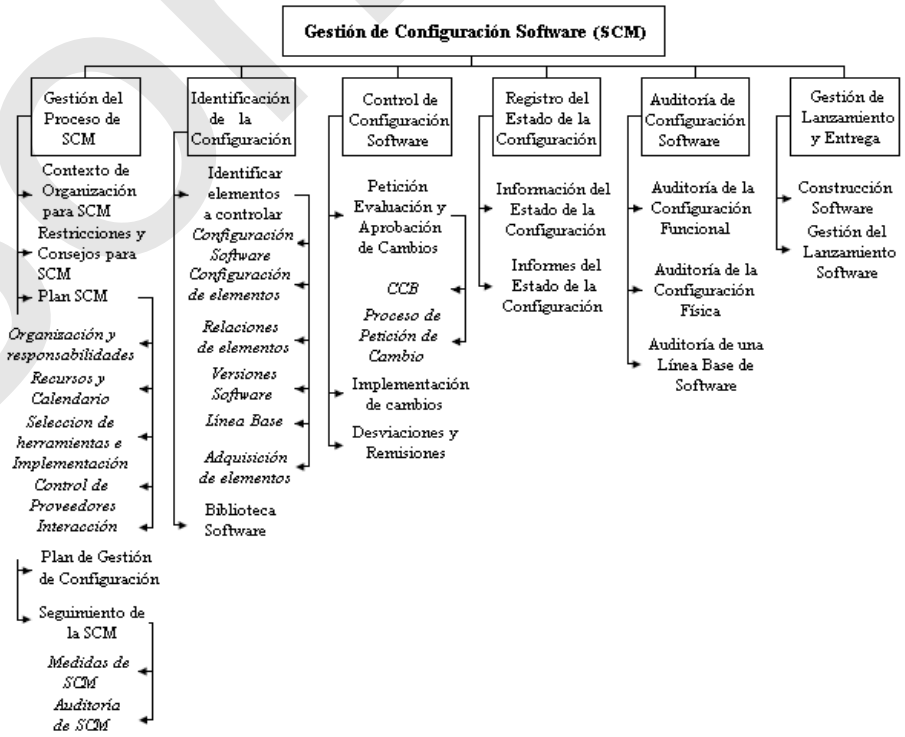


Figura 2 División de los puntos de función para el KA de la

proceso y valoración de procesos como el Modelo de la Capacidad de Madurez Integrado del Instituto de la Ingeniería del Software (SEI/CMMI) (SEI01) y el estándar ISO/IEC15504 de la Valoración del Proceso de la Ingeniería del Software (ISO/IEC15504-98).

1.3 Planificar la SCM

[Dar90:c2; IEEE12207.0-96 :c6.s2.1;

Som01:c29]

La planificación de un proceso de la SCM para un proyecto dado debería ser consistente con el contexto de la organización, las restricciones que sean aplicables, los consejos comúnmente aceptados y la naturaleza del proyecto (por ejemplo, tamaño lo crítico que sea). La actividades más importantes cubiertas son: Identificación del Configuración del Software, Control de la Configuración del Software, Responsabilidad del Estado de la Configuración del Software, Auditoría de la Configuración del Software y la Gestión de Lanzamiento y Entrega del Software. Además, se suelen considerar puntos como organización y responsabilidades, recursos y calendarios, selección de herramientas e implementación, control de proveedores y subcontratas y control de la interacción. Los resultados de la planificación de actividades se registran en un Plan de SCM, que normalmente está sujeto a revisión y auditoría de la SQA.

1.3.1 Organización y responsabilidades de la SCM [Ber92:c7; Buc96:c3; IEEE828-98:c4s2]

Para prevenir confusión acerca de quién debe realizar tareas de la SCM determinadas, se deben identificar claramente las organizaciones involucradas en el proceso de la SCM. Responsabilidades específicas para una actividad o tarea de la SCM también deben ser asignadas a organizaciones, bien por título o por elemento de la organización. Se debe identificar también la autoridad general y las vías de información de la SCM, aunque se podría realizar como parte de la fase de planificación de la garantía de la calidad o al nivel de la gestión de proyectos.

1.3.2 Recursos y planificación de la SCM [Ber92:c7; Buc96:c3; IEEE828-98:c4s4; c4s5]

Planificar para la SCM ayuda identifica las necesidades de personal y herramientas que se requieren para realizar las tareas y actividades de la SCM. Aborda cuestiones de planificación estableciendo las secuencias de tareas de la SCM necesarias e identifica sus relaciones con los planes de proyecto y los hitos establecidos en la fase de planificación del proyecto. También se especifican las necesidades de formación requeridas para la implementación de los planes y formación de personal.

1.3.3 Selección e implementación de herramientas [Ber92:c15; Con98:c6; Pre01:c31]

Las actividades de la SCM son soportadas por diferentes tipos de habilidad de herramientas y procedimientos para su uso. Dependiendo de la situación, se pueden combinar estas habilidades de herramientas con herramientas manuales, herramientas automáticas que proporcionan una habilidad simple de la SCM, herramientas automáticas que integran una colección de habilidades de la SCM (e incluso otras habilidades) o entornos de herramientas integrados que cubren las necesidades de múltiples participantes en el proceso de ingeniería del software (por ejemplo, SCM, desarrollo, V&V). El apoyo de herramientas automáticas se comienza a ser más importante y difícil de establecer, según los proyectos crecen en tamaño y el entorno del proyecto se hace más complejo. Las habilidades de estas herramientas proporcionan apoyo para:

- La biblioteca de la SCM
- Procedimientos de aprobación y de petición de cambios del software (SCR)
- Tareas de gestión de cambios y código (y los productos relacionados)
- Informes del estado de configuración del software y reunión de medidas de la SCM
- Auditoría de la configuración del software
- Gestión y seguimiento de la documentación del software
- Construcción del software
- Gestión y seguimiento de los lanzamientos del software y su distribución

Las herramientas utilizadas en estas áreas, también pueden proporcionar medidas para mejorar el proceso. Royce [Roy98] describe siete medidas centrales útiles para gestionar procesos de la ingeniería del software. La información disponible de las varias herramientas de la SCM es afín al indicador de Trabajo y Progreso de Royce y a sus indicadores de calidad de Cambio de Tráfico y Estabilidad, Ruptura y Modularidad, Repetición del Trabajo y Adaptabilidad y TMEF (Tiempo medio entre fallos) y Madurez. Los informes para estos indicadores se pueden organizar de varias maneras, como por elemento de configuración del software o por tipo del cambio requerido.

La figura 3 muestra una asignación representativa entre las habilidades de las herramientas y procedimiento a Actividades de la SCM.

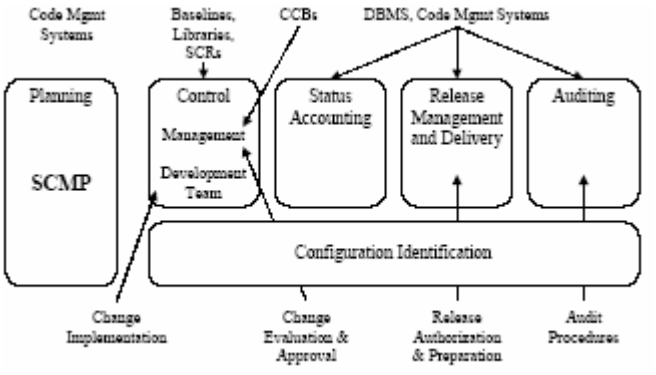


Figura 3 Caracterización de herramientas de SCM y procedimientos relacionados

En este ejemplo, el sistema de gestión de código soporta la utilización de bibliotecas de software al controlar el acceso a los elementos de la biblioteca, coordinar las actividades de múltiples usuarios y ayudar a hacer cumplir los procedimientos operativos. Otras herramientas soportan el proceso de construir software y producir documentación de los elementos de software contenidos en las bibliotecas. Herramientas para la gestión de peticiones de cambios del software soportan los procedimientos de control de cambios que se aplican a elementos de software. Otras herramientas pueden proporcionar gestión de bases de datos y habilidad para generar informes usados para las actividades de gestión,

desarrollo y garantía de calidad. Como se ha mencionado antes, las habilidades de diferentes tipos de herramientas se podrían integrar en sistemas de la SCM, los cuales, a su vez, tienen un acoplamiento alto con otras actividades del software.
Al planificar, el ingeniero de software elige las herramientas de la SCM apropiadas para el trabajo. La planificación debe considerar los problemas que podrían surgir durante la implementación de dichas herramientas, particularmente si se necesita algún cambio cultural. Una introducción de los sistemas de la SCM y de consideraciones en su selección se puede encontrar en [Dar90:c3, AppA] se puede encontrar un caso de estudio de selección de un sistema de la SCM en [Mid97]. Información complementaria acerca de herramientas de la SCM se pueden encontrar en al AC de Métodos y Herramientas de la Ingeniería del Software.

1.3.4 Control Proveedores/Subcontratas [Ber92:c13; Buc96:c11; IEEE828-98:c4s3.6]

Un proyecto de software podría hacer uso o adquirir productos software que se hayan comprado, como compiladores u otras herramientas. La planificación de la SCM considera si y como se pondrán estos elementos bajo control de configuración (por ejemplo, integrados en bibliotecas de proyecto) y como se evaluarán y gestionarán los cambios y actualizaciones.
Consideraciones similares son aplicables al software subcontratado. En este caso, también se deben establecer los requerimientos de la SCM a ser impuestos a los procesos de la SCM del subcontratista como parte del subcontrato y los medios para monitorizar que se cumple con ellos. El último incluye consideraciones acerca de que información de la SCM ha de estar disponible para poder monitorizar, de una manera eficiente, que se cumplen los requerimientos.

1.3.5 Control de Interacción [IEEE828-98:c4s3.5]

Cuando un elemento de software interacciona con otro elemento de software o hardware, un cambio a cualquiera de los dos elementos puede afectar al otro. La planificación para los procesos de la SCM considera como se identificarán los elementos que interaccionan y como se gestionarán y comunicarán los cambios a dichos elementos. El papel de la SCM puede ser parte de proceso de nivel de sistema más general para el control y especificación de las interacciones y podría afectar a las especificaciones de las interacciones, planes de control de las interacciones e documentos de control de las interacciones. En este caso, la planificación de la SCM para el control de las interacciones ocurre dentro del contexto del proceso de nivel de sistema. Se puede encontrar una discusión del rendimiento de las actividades del control de las interacciones en [Ber92:c12].

1.4 Plan de la SCM [Ber92:c7; Buc96:c3; Pau93:L2-81]

Los resultados de la planificación de la SCM para un proyecto dado se reflejan en un Plan de Gestión de la Configuración del Software, que es un "documento vivo" que sirve como referencia para los procesos de la SCM. El documento se mantiene (o sea, se actualiza y aprueba) según vaya siendo necesario durante el ciclo de vida del software. Al implementar un SCMP, normalmente es necesario desarrollar un conjunto de procedimientos subordinados más detallados, que definirán la forma en que se realizan requerimientos específicos durante las actividades del día a día.
Se pueden utilizar varias fuentes de información para encontrar consejo, basado en la información producida durante la actividad de planificación, acerca de la creación y mantenimiento de un SCMP, como en [IEEE828-98:c4]. Esta referencia proporciona requerimientos para la información que un SCMP ha de contener. También define y describe seis categorías de información de la GCS que se han de incluir en un SCMP:

- Introducción (propósito, extensión, términos usados)
- Gestión de la SCM (organización, responsabilidades, autoridades, normas aplicables, directivas y procedimientos)
- Actividades de la SCM (identificación de la configuración, control de la configuración, etc.)
- Planificación de la SCM (coordinación con otras actividades del proyecto)
- Recursos de la SCM (herramientas, recursos físicos y recursos humanos)
- Mantenimiento del SCMP

1.5 Seguimiento de la Gestión del la Configuración del Software [Pau93:L2-87]

Después de que el proceso de la SCM de ha implementado, puede ser necesario un cierto nivel de seguimiento para asegurarse de que las previsiones de la SCM se llevan a cabo adecuadamente (véase, por ejemplo [Buc96]). Probablemente habrá requerimientos específicos de la SQA para asegurarse de que se cumplen los procesos y procedimientos específicos de la SCM. Esto podría requerir que una autoridad de la SCM se asegure de que aquellos que tengan responsabilidades asignadas realizan las tareas de la SCM definidas de una manera correcta. Este seguimiento podría ser realizado, como parte de una actividad de cumplimiento de auditoría, por la autoridad de la garantía de la calidad del software.
El uso de herramientas integradas de la SCM con posibilidades de control de procesos puede hacer la tarea de seguimiento más fácil. Algunas herramientas facilitan comprobar que el proceso se cumple al mismo tiempo que proporcionan al ingeniero de software la flexibilidad de adaptar procedimientos. Otras herramientas se aseguran de que el proceso se siga, dejando menos flexibilidad al ingeniero de software. Los requerimientos de seguimiento y los niveles de flexibilidad que se proporcionarán al ingeniero de software son criterios importantes durante la selección de herramientas.

1.5.1 Medidas y mediciones de la SCM [Buc96:c3; Roy98]

Se pueden asignar medidas de la SCM para proporcionar información específica acerca de la evolución del producto o una visión interna de cómo funcionan los procesos de la SCM. Un objetivo relacionado con el seguimiento de los procesos de la SCM es descubrir oportunidades para mejorar los procesos. Las mediciones de procesos de la SCM proporcionan un buen medio para monitorizar la efectividad de las actividades de la SCM de una manera continuada. Estas mediciones son útiles para caracterizar el estado actual del proceso y para proporcionar una base para hacer comparaciones con el

tiempo. Los análisis de las mediciones pueden proporcionar ideas que lleven a cambios en el proceso y a las correspondientes actualizaciones del SCMP.
Las bibliotecas de software y las diferentes habilidades de las herramientas de la SCM proporcionan fuentes para extraer información acerca de las características de los procesos de la SCM (e información del proyecto y de gestión). Por ejemplo, sería útil para evaluar los criterios usados para determinar qué niveles de autoridad son los óptimos para autorizar ciertos tipos de cambios, el tener información acerca del tiempo necesario para realizar distintos tipos de cambios.
Se debe tener cuidado en asegurarse de mantener el objetivo del seguimiento en los descubrimientos que se pueden ganar de las mediciones y no en las mediciones en sí mismas. El KA del Proceso de la Ingeniería del Software presenta una discusión acerca de medidas del producto y de los procesos. El programa de medidas del software se describe en el KA de la Gestión del Ingeniería del Software.

1.5.2 Auditorías durante el proceso de la SCM [Buc96:c15]

Se pueden llevar a cabo auditorías durante el proceso de ingeniería del software para investigar el estado actual de elementos específicos de la configuración o para evaluar la implementación del proceso de la SCM. Las auditorías durante el proceso de la SCM proporcionan un mecanismo más formal para monitorizar aspectos seleccionados del proceso y se podría coordinar con la función del SQA. Véase también el punto 5 *Auditando la Configuración del Software*.

3. Identificación de la Configuración del Software [IEEE12207.0-96:c6s2.2]

La actividad de la identificación de la configuración del software identifica elementos que se han de controlar, establece métodos de identificación para los elementos y sus versiones y establece las herramientas y técnicas que se usarán para adquirir y gestionar los elementos controlados. Estas actividades proporcionan la base para las otras actividades de la SCM.

2.1 Identificando los Elementos a Controlar [Ber92:c8; IEEE828-98:c4s3.1; Pau93:L2-83; Som05:c29]

Un primer paso para controlar cambios es identificar los elementos de software a ser controlados. Esto requiere comprender la configuración del software en el contexto de la configuración del sistema, seleccionando elementos de configuración de software, desarrollando estrategias para etiquetar elementos de software y describir las relaciones entre ellos e identificar las líneas base que se usarán, además de los procedimientos de adquisición de elementos para una línea base.

2.1.1 Configuración del software [Buc96:c4; c6, Pre04:c27]

Una configuración del software es el conjunto de características funcionales y físicas del software tal y como se han definido en la documentación técnica o conseguido en un producto final (IEEE610.12-90). Se puede ver como parte de una configuración del sistema más general.

2.1.2 Elemento de configuración del software [Buc96:c4;c6; Con98:c2; Pre04:c27]

Un elemento de configuración de software (SCI) es una agregación de software, asignado para tener gestión de configuración y se trata como una sola entidad en el proceso de la SCM (IEEE610.12-90). La SCM controla un conjunto variado de elementos aparte del código. Los planes, documentación de especificaciones y diseño, material de pruebas, herramientas de software, código fuente y ejecutable, bibliotecas de código, datos y diccionarios de datos y documentación para la instalación, mantenimiento, operación y uso del software, están entre los elementos de software con potencial para convertirse en SCIs.
La selección de SCIs es un proceso importante en el que se ha de conseguir un equilibrio entre proporcionar una visibilidad adecuada para el control del proyecto y proporcionar un número manejable de elementos a controlar. Se puede encontrar una lista con criterios para la selección de SCIs en [Ber92].

2.1.3 Relaciones entre elementos de la configuración del software [Con98:c2; Pre04:c27]

La relación estructural entre los SCIs seleccionados y sus partes constituyentes, afectan a otras actividades o tareas de la SCM, como la construcción del software o el análisis del impacto de los cambios propuestos. El seguimiento adecuado de estas relaciones es también importante como soporte a las operaciones de seguimiento. La necesidad de asignar los elementos identificados a la estructura del software y la necesidad de soportar la evolución de los elementos de software y sus relaciones, se debería considerar durante el diseño de los métodos de identificación para los SCIs.

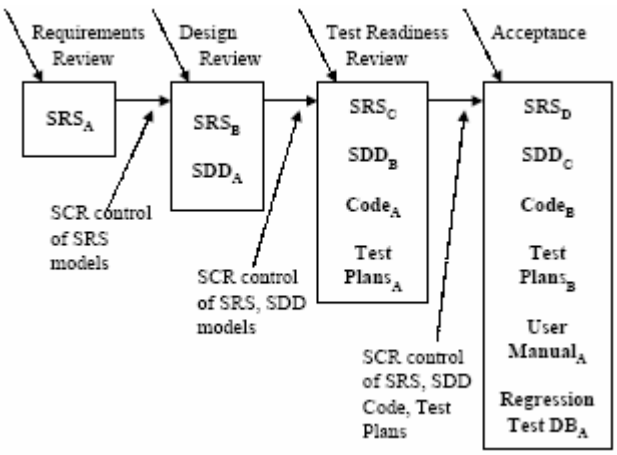
2.1.4 Versiones del software [Bab86:c2]

Los elementos de software evolucionan al mismo tiempo que el proyecto de software avanza. Una *versión* de un elemento de software es un elemento identificado y especificado particularmente. Se puede pensar en ella como el estado de un elemento que evoluciona. [Con98:c3-c5] Una *revisión* es una nueva versión de un elemento que reemplazará la versión anterior. Una *variante* es una nueva versión de un elemento que se añadirá la configuración sin reemplazar la versión anterior.

2.1.5 Línea base [Bab86:c5; Buc96:c4; Pre04:c27]

La línea base de un software es un conjunto de elementos

Figura 4 Adquisición de elementos



de configuración del software que se han designado formalmente y fijados en un momento determinado durante el ciclo de vida del software. El término se usa también para referirse a una versión en particular de un elemento de la configuración del software acordada previamente. En cualquiera de los casos, la línea base solo se puede cambiar por medio de procedimientos de control de cambios formales. Una línea base representa, junto con todos los cambios aprobados para la línea base, la configuración actual aprobada.
Algunas líneas base comúnmente utilizadas son la funcional, la asignada, la de desarrollo y la de productos (véase por ejemplo [Ber92]). La línea base funcional se corresponde con los requerimientos del sistema ya verificados. La línea base asignada se corresponde con las especificaciones de los requerimientos del sistema y las especificaciones de los requerimientos de las interacciones entre software. La línea base de desarrollo representa la configuración evolutiva del software en momentos determinados durante el ciclo de vida del proyecto. La autoridad de cambios para dicha línea base es normalmente la responsabilidad de la organización de desarrollo, pero se podría compartir con otras organizaciones (por ejemplo la de SCM o la de Pruebas). La línea base de un producto se corresponde con el producto de software completo, entregado para integración de sistemas. Las líneas base a usar en un proyecto determinado, junto con los niveles de autoridad asociados necesarios para la aprobación de cambios, se identifican normalmente en el SCMP.

2.1.6 Adquisición de elementos de configuración del software [Buc96:c4]

Diferentes elementos de configuración del software se ponen bajo el control de la SCM en momentos distintos; lo que significa que se añaden a una línea base en particular en momentos específicos del ciclo de vida del software. El evento que da comienzo al proceso es la terminación de alguna forma de tarea formal de aceptación, como una revisión formal. La figura 2 caracteriza el crecimiento de elementos en una línea base durante el ciclo de vida. Esta figura se basa en el modelo de cascada solamente por motivos ilustrativos; los subscripts usados en la figura indican la versión de los elementos durante su evolución. La petición de cambios del software (SCR) se describe en el punto 3.1 *Petición, Evaluación y Aprobación de Cambios del Software*.
Seguidamente de la adquisición de un SCI, los cambios a dicho elemento se deben aprobar formalmente de la manera apropiada para el elemento y la línea base involucrados, como se define en el SCMP. Después de la aprobación, el elemento se incorpora en la línea base del software siguiendo el procedimiento apropiado.

2.2 Biblioteca de Software [Bab86:c2; c5; Buc96:c4; IEEE828- 98:c4s3.1; Pau93:L2-82; Som01:c29]

Una biblioteca de software es una colección controlada de software y los documentos relacionados, y está diseñada para ayudar en el desarrollo del software, su uso y mantenimiento (IEEE610.12-90). También tiene un papel durante las actividades de gestión de lanzamientos y entrega de software. Se pueden usar varios tipos de bibliotecas de software, cada uno se corresponde con un nivel de madurez determinado del elemento de software. Por ejemplo, una biblioteca de desarrollo podría dar soporte durante la codificación y una biblioteca de soporte de proyectos podría dar soporte a las pruebas, mientras que una biblioteca maestra se podría utilizar en el producto final. Se ha de asociar el nivel apropiado de control de la SCM (la línea base asociada y el nivel de autoridad para cambios) a cada biblioteca. La seguridad, en términos de control de acceso y medios de copia de seguridad, es un aspecto clave en la gestión de bibliotecas. Un modelo de una biblioteca de software se puede encontrar en [Ber92:c14].
La herramienta/s que se usan en cada biblioteca deben soportar los controles de la SCM que sean necesarios para dicha biblioteca, en términos de control de los SCIs y de acceso a la biblioteca. En el nivel de la biblioteca de desarrollo, esto significa la capacidad de gestión de código que dará servicio a desarrolladores, ingenieros de mantenimiento y SCM. Está enfocada a gestionar las versiones de los elementos de software al mismo tiempo que da soporte a las actividades de múltiples desarrolladores. A mayores niveles de control, el acceso es más restringido y el principal usuario en la SCM.
Estas bibliotecas son también una fuente importante de información para mediciones del trabajo realizado y del progreso.

4. Control de la Configuración del Software [IEEE12207.0-96:c6s2.3; Pau93:L2-84]

Al control de la configuración del software le concierne la gestión de cambios durante el ciclo de vida del software. Cubre los procesos que determinan los cambios que se realizarán, la autoridad requerida para aprobar ciertos cambios, el soporte para la implementación de dichos cambios y el concepto de desviación formal de los requerimientos del proyecto, además de las cancelaciones de requerimientos. La información derivada de estas actividades es útil para medir el tráfico de cambios y ruptura y aspectos por rehacer.

3.1 Petición, Evaluación y Aprobación de Cambios del Software [IEEE828-98:c4s3.2; Pre04:c27;Som05:c29]

El primer paso para gestionar cambios en elementos controlados es determinar los cambios a realizar. El proceso de petición de cambio del software (véase la Figura 5) proporciona procedimientos formales para recoger y registrar peticiones de cambios, evaluando el coste e impacto potencial de un cambio propuesto y aceptar, modificar o rechazar el cambio propuesto. Las peticiones de cambios de elementos de la configuración del software los puede originar cualquiera durante cualquier momento del ciclo de vida del software y puede incluir una solución propuesta y una prioridad. Una fuente de petición de cambios es la iniciación de acciones

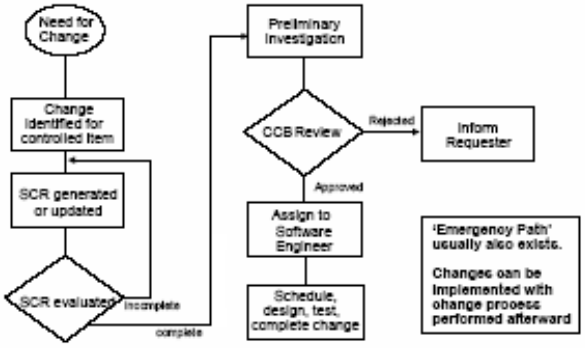


Figura 5 Flujo de un Proceso de Control de Cambios

correctivas en respuesta a los informes de problemas. El tipo de cambio (por ejemplo, un defecto o mejora) se registra normalmente en la SCR, sin importar la fuente. Esto proporciona la oportunidad de seguir defectos y recoger mediciones de la actividad de cambios por tipo de cambio. Una vez se ha recibido un SCR, se realiza una evaluación técnica (también conocida como análisis del impacto) para determinar el tamaño de las modificaciones necesarias en caso de que se aceptara la petición de cambio. Para realizar esta tarea es importante un buen entendimiento de las relaciones entre elementos de software (y posiblemente hardware). Finalmente, la evaluación de los aspectos técnicos y de gestión de la petición de cambios, será realizada por una autoridad establecida, de acuerdo con la línea base afectada, el SCI involucrado y la naturaleza del cambio y entonces se aceptará, modificará, rechazará o pospondrá el cambio propuesto.

3.1.1 Consejo de Control de la Configuración del Software [Ber92:c9; Buc96:c9,c11; Pre04:c27]

La autoridad para aceptar o rechazar los cambios propuestos, es normalmente la responsabilidad de una entidad conocida como Consejo de Control de la Configuración (CCB). En proyectos pequeños, dicha autoridad normalmente reside en el Jefe de Proyecto o algún otro individuo elegido, en vez de en un consejo de varias personas. Puede haber múltiples niveles de autoridad de cambios, dependiendo de una variedad de criterios, como cuan crítico sea el elemento involucrado, la naturaleza del cambio (por ejemplo, el impacto en el presupuesto y planificación), o el momento actual en el ciclo de vida. La composición de CCBs que se utilice para un sistema determinado varia en relación a estos criterios (siempre atendería un representante de la SCM). Cuando el alcance de la autoridad de un CCB es está limitado solamente al software, se le conoce como Consejo de Control de Configuración del Software (CCBS). Las actividades del CCB están sujetas normalmente a auditorías de la calidad de software o revisiones.

3.1.2 Proceso de petición de cambios del software [Buc96:c9,c11; Pre04:c27]

Un proceso efectivo de petición de cambio del software (SCR) requiere el uso de herramientas de soporte y procedimientos, desde formularios de papel y un procedimientos documentado hasta la herramienta electrónica para generar peticiones de cambio, imponiendo el flujo del proceso de cambios, capturando las decisiones del CCB y produciendo información del proceso de cambio. Un enlace entre las habilidades de esta herramienta y el sistema de informe de errores puede facilitar el seguimiento de soluciones para los informes de errores. Las descripciones del proceso de cambios y los formularios de soporte (información) aparecen en gran número de las referencias, por ejemplo [Ber92:c9].

3.2 *Implementando Cambios en el Software* [Bab86:c6; Ber92:c9; Buc96:c9,c11; IEEE828-98:c4s3.2.4; Pre04:c27; Som05:c29]

Las PCBs aprobadas se implementan utilizando los procedimientos de software definidos, de acuerdo con los requerimientos de planificación aplicables. Como se podría implementar simultáneamente un número de PCBs, es necesario proporcionar los medios para seguir que PCBs se añaden a que versiones de software y líneas bases particulares. Como parte de la finalización del proceso de cambios, los cambios completados podrían sufrir auditorías de configuración y verificación de la calidad del software. Esto incluye asegurarse de que solo se han realizado los cambios aprobados. El proceso de petición de cambios mencionado anteriormente, añadirá la información de la aprobación para el cambio a la documentación de la SCM (y otras).
La implementación real de un cambio está soportada por las habilidades de la herramienta de bibliotecas, que proporciona gestión de versiones y soporte para el almacenamiento de código. Estas herramientas proporcionan como mínimo habilidades para llevar a cabo el control de de las versiones asociadas. Herramientas más potentes pueden dar soporte al desarrollo en paralelo y entornos geográficamente distribuidos. Estas herramientas podrían aparecer como aplicaciones especializadas separadas, bajo el control de un grupo independiente de la SCM. También podrían aparecer integradas como parte del entorno de la ingeniería del software. Finalmente, podrían ser tan elementales como un sistema de control de cambios rudimentario proporcionado por el sistema operativo.

3.3 *Desviaciones y Remisiones* [Ber92:c9; Buc96:c12]

Las limitaciones impuestas al esfuerzo de la ingeniería del software o las especificaciones producidas durante las actividades de desarrollo podrían contener necesidades que no pueden ser satisfechas en el punto designado del ciclo de vida. Una remisión es la autorización para abandonar una necesidad antes del desarrollo del elemento. Un rechazo es la autorización para utilizar un elemento, después de su desarrollo, que se aleja de la necesidad de alguna manera. En estos casos se usa un procedimiento formal para ganar la aprobación para la desviación o remisión de las necesidades.

5. Registro del Estado de la Configuración del Software

[IEEE12207.0-96:c6s2.4; Pau93:L2-85; Pre04:c27;Som05:c29]

La contabilidad del estado de la configuración del software (SCSA) es la actividad de registrar y proporcionar la información necesaria para una gestión efectiva de la configuración del software

4.1 Información del Estado de la Configuración del Software [Buc96:c13; IEEE828-98:c4s3.3]

La actividad de la SCSA diseña y opera un sistema para la captura y generación de los informes necesarios durante el ciclo de vida. Como en cualquier sistema de información, se debe identificar, recoger y mantener la información del estado de la configuración que se ha de gestionar según las configuraciones evolucionan. Se necesitan varias mediciones e información para dar soporte al proceso de la SCM y para cubrir las necesidades de informes del estado de la configuración de las actividades de gestión, ingeniería del software y otras actividades relacionadas. Los tipos de información disponible incluyen la identificación de la configuración aprobada y la identificación y estado de implementación actual de cambios, desviaciones y remisiones. Se puede

encontrar una lista parcial con elementos de datos importantes en [Ber92:c10]
Es necesario algún tipo de soporte de herramientas automáticas para llevar a cabo las tareas de recogida de datos y generación de informes de la SCSA. Podría ser una habilidad de la base de datos o una herramienta independiente o la habilidad del entorno de una herramienta integrada más grande.

4.2 Informes del Estado de la Configuración del Software [Ber92:c10; Buc96:c13]

Los informes generados pueden ser usados por varios elementos de la organización o del proyecto, incluyendo el equipo de desarrollo, el equipo de mantenimiento, la gestión del proyecto y las actividades de calidad de software. Los informes pueden tener la forma de respuestas inmediatas a preguntas específicas o ser informes prediseñados producidos periódicamente. Alguna de la información producida por las actividades de contabilidad del estado durante el curso del ciclo de vida podría acabar siendo registros de la garantía de la calidad.
Además de informar del estado actual de la configuración, la información obtenida por la SCSA puede usarse como base para varias mediciones útiles para la gestión, desarrollo y SCM. Un ejemplo podría ser el número de cambios pedidos por ECS y el tiempo medio necesario para implementar una petición de cambio.

5. Auditoría de la Configuración del Software [IEEE828-98:c4s3.4; IEEE12207.0-96:c6s2.5;Pau93:L2-86; Pre04:c26c27]

La auditoría de software es una actividad que se realiza para evaluar independientemente la conformidad de productos de software y procesos con las regulaciones, estándares, guías, planes y procedimientos (IEEE1028-97). Las auditorías se llevan a cabo de acuerdo con un proceso bien definido que consiste en varias responsabilidades y papeles de auditoría. En consecuencia, cada auditoría se debe planear con cuidado. Una auditoría requiere un número de personas que realizarán una variedad de tareas en un periodo de tiempo bastante reducido. Herramientas que den soporte a la planificación y ejecución de la auditoría pueden facilitar el proceso enormemente. Se pueden encontrar consejos para realizar auditorías de software en varias referencias, como [Ber92:c11, Buc96:c15] y (IEEE1028-97).
La actividad de auditoría de la configuración del software determina el grado en que un elemento satisface las características funcionales y físicas. Se pueden realizar auditorías informales de este tipo en momentos clave del ciclo de vida. Hay dos tipos de auditorías que podrían ser requeridas por el contrato (por ejemplo, en contratos para software crítico): la Auditoría de la Configuración Funcional (FCA) y la Auditoría de la Configuración Física (PCA). El completar con éxito estas auditorías puede ser un prerrequisito para establecer la línea base del producto. Buckley [Buc96:c15] contrasta los objetivos de las FCA y PCA en los contextos de software y hardware y recomienda que se evalúe cuidadosamente la necesidad de una FCA y PCA de software antes de realizarlas.

5.1 Auditoría de la Configuración Funcional del Software

El propósito de la FCA del software es asegurarse de que el elemento de software que se audita es consistente con la especificación. Los resultados de la verificación y validación del software son actividades clave como entrada de datos para esta auditoría.

5.2 Auditoría de la Configuración Física del Software

El propósito de la auditoría de la configuración física del software (PCA) es asegurarse de que el diseño y la documentación de referencia son consistentes con el producto de software tal y como se ha construido.

5.3 Auditorías durante el proceso de una Línea Base de Software

Tal y como se menciona anteriormente, las auditorías se pueden llevar a cabo durante el proceso de desarrollo para investigar el estado actual de un elemento de la configuración específico. En dicho caso, se podría aplicar una auditoría a elementos seleccionados de la línea base para asegurarse de que el rendimiento es consistente con las especificaciones o para asegurarse de que la documentación continua siendo consistente con el elemento de la línea base que se está desarrollando.

6. Gestión del Lanzamiento y Distribución del Software [IEEE12207.0-96:c6s2.6]

El término “lanzamiento” se usa en este contexto para referirse a la distribución un elemento de la configuración del software fuera de la actividad de desarrollo. Esto incluye tanto lanzamientos internos como la distribución a clientes. Cuando una versión diferente de un elemento de software está disponible para ser entregada, como las versiones para diferentes plataformas o versiones con diferentes capacidades, es normalmente necesario preparar una versión específica y empaquetar los materiales adecuados para distribuirla. La biblioteca de software es un elemento clave para realizar las tareas de lanzamiento y distribución.

6.1 Construcción del Software [Bab86:c6; Som05:c29]

La construcción del software es la actividad de combinar la versión correcta de los elementos de configuración del software, usando la configuración de datos apropiada, en un programa ejecutable para su distribución a los clientes u otros receptores, como la actividad de pruebas. Las instrucciones de construcción se aseguran de que se toman los pasos de construcción adecuados y en la secuencia correcta. Además de construir software para un nuevo lanzamiento, la SCM normalmente necesita ser capaz de reproducir lanzamientos previos para recuperación, pruebas, mantenimiento u otros propósitos de lanzamiento adicionales.
El software se construye usando versiones particulares de la herramientas de soporte, como compiladores. Podría ser necesario reconstruir una copia exacta de un elemento de configuración que se haya construido previamente. En ese caso, las herramientas de soporte y las instrucciones de construcción asociadas deben de estar bajo el control de la SCM para asegurarse de la disponibilidad de la las versiones correctas de las herramientas.
Las habilidades de una herramienta son útiles para seleccionar la versión correcta de elementos de software para un entorno destino determinado y para el proceso de

construir el software automáticamente con las versiones seleccionadas y los datos de configuración apropiados. En proyectos grandes con desarrollo en paralelo o en entornos de desarrollo distribuido, estas habilidades de las herramientas son necesarias. La mayoría de los entornos de ingeniería del software proporcionan esta habilidad. Estas herramientas varían en complejidad, desde las que requieren que el ingeniero de software aprenda un lenguaje de guiones específico a soluciones gráficas que ocultan la mayor parte de la complejidad en una solución de construcción "inteligente".

El proceso y los productos de la construcción están sujetos, normalmente, a verificación de la calidad del software. Los resultados de un proceso de construcción se podrían necesitar para futuras referencias y podrían convertirse en registros de la garantía del software.

6.2 Gestión del Lanzamiento del Software [Som05:c29]

La gestión de lanzamiento del software conlleva la identificación, empaquetamiento y distribución de los elementos de un producto, por ejemplo, programas ejecutables, documentación, notas de lanzamiento y datos de configuración. Dado que los cambios pueden ocurrir constantemente, una de las preocupaciones en la gestión de lanzamientos es determinar cuándo realizar un lanzamiento. La severidad de los problemas solucionados por el lanzamiento afecta a esta decisión (Som01). La tarea de empaquetamiento debe identificar que elementos del producto se deben distribuir y por tanto seleccionar las variantes correctas de dichos elementos, dada la aplicación que se le quiere dar al producto. La información que documenta el contenido físico del lanzamiento se conoce como documento de descripción de la versión. Las notas del lanzamiento normalmente describen nuevas habilidades, problemas conocidos y requisitos necesarios de la plataforma para la operación adecuada del producto. El paquete que se distribuirá también contiene instrucciones de instalación o actualización. El último se puede complicar porque algunos usuarios podrían tener productos que son antiguos por varias versiones. Finalmente, en algunos casos, se podrían requerir la actividad de gestión de lanzamientos para el seguimiento de la distribución del producto a varios clientes o sistemas objetivo. Un ejemplo sería el caso en el que se requiriese que un proveedor tiene que notificar a un cliente de nuevos problemas.

Las habilidades de una herramienta son necesarias para dar soporte a estas funciones de gestión de los lanzamientos. Es útil tener una conexión con las habilidades de la herramienta para dar soporte a los procesos de peticiones de cambios, de tal forma que se puedan relacionar los contenidos de un lanzamiento con los SCR que se han recibido. Esta habilidad de las herramientas también podría mantener información en varias plataformas destino y de varios entornos de clientes.

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Bab86]	[Ber92]	[Buc96]	[Con98]	[Dar90]	[IEEE828-98]	[IEEE1220 7.0-96]	[Mid97]	[Moo98]	[Pau93]	[Pre04]	[Roy98]	[Som05]
1.Gestión del proceso SCM													
1.1Contexto de Organización para SCM		c4	c2		c2	c4s2.1							
1.2Restricciones y Consejos para SCM		c5				c4s1, c4s2.3	6.2.1		*				
1.3Planificación de SCM					c2								c29
Organización y responsabilidades		c7	c3			c4s2							
Recursos y planificación		c7	c3			c4s4, c4s5							
Selección de herramientas e implementación		c15		c6	c3, App A			*				*	
Control Proveedores/Subcontratas		c13	c11			c4s3.6							
Control de Interacción		c12				c4s3.5							
1.4Plan de SCM		c7	c3			c4				L2-81			
1.5Seguimiento de la SCM			*			c4				L2-87			
Métricas y mediciones			c3									188-202, 283-298	
Auditorias durante el proceso de SCM			c15										
2. Identificación de la Configuración Sw							c6s2.2						
2.1Identificando los elementos a controlar		c8				c4s3.1				L2-83			c29
Configuración software			c4, c6								c27		
Elementos de configuración software		*	c4, c6	c2							c27		
Relaciones entre elementos de Configuración Sw				c2							c27		
Versiones Software	c2										c27		
Línea Base	c5	*	c4								c27		
Adquisición de elementos de Configuración Sw			c4										
2.2Biblioteca Software	c2, c5	c14	c4			c4s3.1				L2-82			c29
3. Control de la Configuración Sw							c6s2.3			L2-84			
3.1 Petición, Evaluación y Aprobación de Cambios						c4s3.2					c27		c29
Control de la Configuración del Sw		c9	c9, c11								c27		
Proceso de Petición de Cambios Sw		c9	c9, c11								c27		
3.2 Implementando Cambios del Sw	c6	c9	c9, c11			c4s3.2.4					c27		c29
3.3 Desviaciones y Remisiones		c9	c12										
4. Registro del Estado de la Configuración Sw							c6s2.4			L2-85	c27		c29
4.1 Información del Estado de la Configuración Sw		c10	c13			c4s3.3							
4.2 Informes del Estado de la Configuración Software		c10	c13										
5. Auditoría de la Configuración Sw		c11	c15			c4s3.4	c6s2.5			L2-86	c27, c26		
5.1 Auditoría de la Configuración Funcional													
5.2 Auditoría de la Configuración Física													
5.3 Auditoría de una Línea Base de Sw													
6. Gestión del Lanzamiento y Distribución Sw							c6s2.6						
6.1 Construcción Software	c6												c29
6.2 Gestión del Lanzamiento Sw													c29

REFERENCIAS RECOMENDADAS PARA SCM

[Bab86] W.A. Babich, *Software Configuration Management, Coordination for Team Productivity*, Addison-Wesley, 1986.

[Ber92] H.R. Berlack, *Software Configuration Management*, John Wiley & Sons, 1992.

[Buc96] F.J. Buckley, *Implementing Configuration Management: Hardware, Software, and Firmware*, second ed., IEEE Computer Society Press, 1996.

[Con98] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Computing Surveys*, vol. 30, iss. 2, June 1998.

[Dar90] S.A. Dart, *Spectrum of Functionality in Configuration Management Systems*, Software Engineering Institute, Carnegie Mellon University, 1990.

[IEEE828-98] IEEE Std 828-1998, *IEEE Standard for Software Configuration Management Plans*, IEEE, 1998.

[IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

[Mid97] A.K. Midha, "Software Configuration Management for the 21st Century," *Bell Labs Technical Journal*, vol. 2, iss. 1, Winter 1997, pp. 154-165.

[Moo98] J.W. Moore, *Software Engineering Standards: A User's Roadmap*, IEEE Computer Society, 1998.

[Pau93] M.C. Paulk et al., "Key Practices of the Capability Maturity Model, Version 1.1," technical report CMU/SEI-93-TR-025, Software Engineering Institute, Carnegie Mellon University, 1993.

[Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, Sixth ed, McGraw-Hill, 2004.

[Roy98] W. Royce, *Software Project Management, A United Framework*, Addison-Wesley, 1998.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005.

APÉNDICE A. LISTA DE LECTURAS ADICIONALES

(Bab86) W.A. Babich, *Software Configuration Management, Coordination for Team Productivity*, Addison-Wesley, 1986.

(Ber92) H.R. Berlack, *Software Configuration Management*, John Wiley & Sons, 1992.

(Ber97) E.H. Bersoff, "Elements of Software Configuration Management," in *Software Engineering*, M. Dorfman and R.H. Thayer, eds., IEEE Computer Society Press, 1997.

(Buc96) F.J. Buckley, *Implementing Configuration Management: Hardware, Software, and Firmware*, second ed., IEEE Computer Society Press, 1996.

(ElE98) K. El-Emam et al., "SPICE, The Theory and Practice of Software Process Improvement and Capability Determination," presented at IEEE Computer Society, 1998.

(Est95) J. Estublier, "Software Configuration Management," presented at ICSE SCM-4 and SCM-5 Workshops, Berlin, 1995.

(Gra92) R.B. Grady, *Practical Software Metrics for Project Management and Process Management*, Prentice Hall, 1992.

(Hoe02) A. v. d. Hoek, "Configuration Management Yellow Pages," 2002, available at http://www.cmtoday.com/yp/configuration_management.html.(Hum89) W. Humphrey, *Managing the Software Process*,Addison-Wesley, 1989.

(Pau95) M.C. Paulk et al., *The Capability Maturity Model, Guidelines for Improving the Software Process*, Addison-Wesley, 1995.

(Som01a) I. Sommerville, "Software Configuration Management," presented at ICSE SCM-6 Workshop, Berlin, 2001.

(USNRC1.169-97) USNRC Regulatory Guide 1.169, "Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants," presented at U.S. Nuclear Regulatory Commission, Washington, D.C., 1997.

(Vin88) J. Vincent, A. Waters, and J. Sinclair, *Software Quality Assurance: Practice and Implementation*, Prentice Hall, 1988.

(Whi91) D. Whitgift, *Methods and Tools for Software Configuration Management*, John Wiley & Sons, 1991.

.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE730-02) IEEE Std 730-2002, *IEEE Standard for Software Quality Assurance Plans*, IEEE, 2002.

(IEEE828-98) IEEE Std 828-1998, *IEEE Standard for Software Configuration Management Plans*, IEEE, 1998.

(IEEE1028-97) IEEE Std 1028-1997 (R2002), *IEEE Standard for Software Reviews*, IEEE, 1997.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/ IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

(IEEE12207.1-96) IEEE/EIA 12207.1-1996, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes - Life Cycle Data*, IEEE, 1996.

(IEEE12207.2-97) IEEE/EIA 12207.2-1997, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes - Implementation Considerations*, IEEE, 1997.

(ISO15846-98) ISO/IEC TR 15846:1998, *Information Technology - Software Life Cycle Processes - Configuration Management*, ISO and IEC, 1998.

CAPÍTULO 8
GESTIÓN DE LA INGENIERÍA DEL SOFTWARE

ACRÓNIMOS

PMBOK	Guía al Proyecto de Gestión del Cuerpo de Conocimientos
SQA	Garantía de Calidad del Software

INTRODUCCIÓN

La Gestión de la Ingeniería del Software puede definirse como la aplicación para actividades de gestión – planificación, coordinación, mediciones, monitoreo, control e informes – que asegure un desarrollo y mantenimiento del software sistemático, disciplinado y cuantificado (IEEE610.12-90).

El KA de Gestión de la Ingeniería del Software, por tanto, se encarga de la gestión y medición de la ingeniería del software. A pesar de que medir es un aspecto importante en todas las KAs, no es hasta aquí que se presenta el tema de programas de medición.

Aunque por una parte sea verdad afirmar que, en cierto sentido, debiera ser posible gestionar la ingeniería del software de la misma manera que cualquier otro proceso (complejo) existen aspectos específicos de los productos software y de los procesos del ciclo de vida del software que complican una gestión efectiva –sólo algunos de los cuales se apuntan a continuación:

- La percepción de los clientes es tal que con frecuencia existe una falta de aprecio de la complejidad inherente a la ingeniería del software, particularmente en relación al impacto que produce cambiar los requisitos.
- Es casi inevitable que los propios procesos de ingeniería del software generen la necesidad de nuevos o modificados requisitos del cliente.
- Como resultado, el software se construye con frecuencia mediante un proceso iterativo en vez de mediante una secuencia de tareas cerradas.
- La ingeniería del software incorpora necesariamente aspectos de creatividad y de disciplina – mantener un balance apropiado entre los dos es con frecuencia difícil.
- El grado de novedad y de complejidad del software son con frecuencia extremadamente altos.
- La tasa de cambio de la tecnología subyacente es muy rápida.

Con respecto a la ingeniería del software, las actividades de gestión tienen lugar en tres niveles: gestión organizacional y de infraestructura, gestión de proyectos, y programa de planificación y control de mediciones. Estos dos últimos se cubren con más detalle en la descripción de este KA. A pesar de todo, esto no va en detrimento de la importancia de los temas de gestión organizacional.

Dado que la unión con las disciplinas señaladas – obviamente, la de gestión – es importante, se describirá con más detalle que las otras descripciones del KA. Los aspectos de gestión organizacional son importantes en términos de su impacto sobre la ingeniería del software – en gestión de políticas, por ejemplo: políticas organizativas y estándares que proporcionan el marco en el que se desenvuelve la ingeniería del software. Puede que se necesite que estas políticas se vean afectadas por los requisitos de un software de desarrollo y mantenimiento efectivo, y puede que se necesite establecer un número de políticas específicas de ingeniería del software para una gestión eficaz de la ingeniería del software a nivel organizacional. Por ejemplo, normalmente se necesitan políticas para establecer procesos específicos a nivel organizacional o procedimientos para tareas de ingeniería del software tales como el diseño, la implementación, la estimación, el seguimiento y los informes. Tales políticas son esenciales, por ejemplo, para una gestión de la ingeniería del software eficaz a largo plazo, ya que establecen una base consistente sobre la que analizar actuaciones anteriores e implementar mejoras.

Otro aspecto importante de la gestión es la gestión del personal: las políticas y procedimientos para contratar, entrenar y motivar al personal y actuar como mentor del desarrollo de una carrera son importantes no sólo a nivel de proyecto sino también para el éxito a largo plazo de una organización. Todo el personal de desarrollo del software puede haber sido entrenado del mismo modo o puede presentar retos para la gestión del personal (por ejemplo, mantener el capital en un contexto en el que la tecnología subyacente sufre cambios continuos y rápidos). Con frecuencia también se menciona la gestión de la comunicación como un aspecto pasado por alto pero importante de la actuación de los individuos en un campo en el que es necesario un entendimiento preciso de las necesidades del usuario y de los complejos, requisitos y diseños. Finalmente, es necesaria la gestión de la cartera de trabajo, que es la capacidad de tener una visión general, no sólo del conjunto del software en desarrollo sino también del software que ya se está utilizando en la organización. Más aún, la reutilización del software es un factor clave en el mantenimiento y mejora de la productividad y competitividad. Una reutilización eficaz requiere una visión estratégica que refleje el poder único y los requisitos de esta técnica.

Los ingenieros del software deben entender los aspectos de gestión que se encuentran influidos de modo singular por el software, y además conocer sus aspectos más generales, incluso en los primeros cuatro años tras graduarse, como está marcado en la Guía.
La cultura y comportamiento organizacional y la gestión comercial funcional en términos de consecución de metas, aportan la gestión en cadena, la publicidad, la venta y la distribución, todas ellas influyen, aunque sea indirectamente, en el proceso de ingeniería del software de una organización.
La noción de gestión de proyectos tiene que ver con esta KA, como "la construcción de artefactos de software útiles", se gestiona por lo general como (quizás programas de) proyectos individuales. A este respecto, encontramos un amplio respaldo en la Guía al Proyecto de Gestión del Cuerpo de Conocimientos (PMBOK) (PMI00), que en sí misma incluye las siguientes KAs de gestión de proyectos: gestión de integración del proyecto, gestión de objetivos del proyecto, gestión del tiempo del proyecto, gestión del coste del proyecto, gestión de la calidad del proyecto, gestión de los recursos humanos del proyecto y gestión de las comunicaciones del proyecto. Está claro que todos estos temas tienen una relación directa con el KA de Gestión de la Ingeniería del Software. Sería imposible e inadecuado intentar duplicar aquí el contenido de la Guía de la PMBOK. En su lugar, sugerimos que el lector que esté interesado en la gestión de proyectos más allá de lo que es específico a los proyectos de ingeniería del software consulte la propia PMBOK. La gestión de proyectos también se encuentra en el capítulo sobre las Disciplinas Señaladas de la Ingeniería del Software.
El KA de Gestión de la Ingeniería del Software consiste tanto en el proceso de gestión del proyecto de software en sus primeras cinco subáreas, como en la medición de la ingeniería del software en su última subárea. Aunque estos dos temas se suelen considerar como distintos, y de hecho poseen muchos aspectos únicos en sí mismos, su gran cercanía ha llevado a que se les trate de manera conjunta en esta KA. Desafortunadamente, se comparte la percepción común de que la industria del software entrega sus productos tarde, por encima de lo presupuestado, y de pobre calidad e incierta funcionalidad. Una gestión regulada por la medición – un principio presupuesto en cualquier disciplina de verdadera ingeniería – puede ayudar a darle la vuelta a esta percepción. En esencia, una gestión sin medición, cualitativa y cuantitativa, da la sensación de falta de rigor, y medir sin gestionar da la sensación de una falta de fines o de contexto. De igual manera, sin embargo, gestión y medición sin conocimientos de expertos es igualmente ineficaz, por lo que debemos tener cuidado para evitar poner un énfasis excesivo en los aspectos cuantitativos de la Gestión de Ingeniería del Software (GIS). Una gestión eficaz requiere la combinación tanto de números como de experiencia.
Aquí se adoptan las siguientes definiciones de trabajo:

- El *proceso de gestión* se refiere a las actividades que se emprenden para asegurarse de que los procesos de ingeniería del software se realizan de una manera consistente con las políticas, objetivos y estándares de la organización.
- La *medición* se refiere a la asignación de valores y etiquetas a los aspectos de la ingeniería del software (productos, procesos, y recursos según los define [Fen98]) y a los modelos que se derivan de ellos, se hayan desarrollado estos modelos utilizando técnicas estadísticas, conocimientos expertos u otras técnicas.

Las subáreas de gestión del proyecto de ingeniería del software hacen un uso extensivo del subárea de gestión de ingeniería del software.

No es de extrañar que esta KA esté relacionada de cerca con otras en la Guía del SWEBOK, y sería de particular utilidad leer las siguientes descripciones del KA junto con ésta.

- Los Requisitos del Software, en donde se describen algunas de las actividades que tendrán que realizarse durante la fase de definición de Iniciación y Alcance del proyecto.
- La Gestión de Configuración del Software, ya que trata de la identificación, control, consideraciones de estado, y auditoría de la configuración del software, junto con la gestión de entregas y repartos del software
- El Proceso de Ingeniería del Software, porque los procesos y los proyectos están estrechamente relacionados (esta KA también describe la medición de procesos y productos).
- La Calidad del Software, ya que la calidad es un objetivo constante de la gestión y es una meta con muchas actividades que tiene que gestionarse.

DESCOMPOSICIÓN DE LOS TEMAS DE GESTIÓN DE LA INGENIERÍA DEL SOFTWARE

Hemos creado una descomposición basada tanto en temas como en ciclos de vida, ya que el KA de Gestión de Ingeniería del Software se ve aquí como un proceso organizacional que incorpora la noción de gestión de procesos y proyectos. Sin embargo, la base primaria de la descomposición de alto nivel es el proceso de gestionar un proyecto de ingeniería del software. Existen seis subáreas principales. Las primeras cinco subáreas siguen principalmente el Proceso de Gestión IEEE/EIA 12207. Las seis subáreas son:

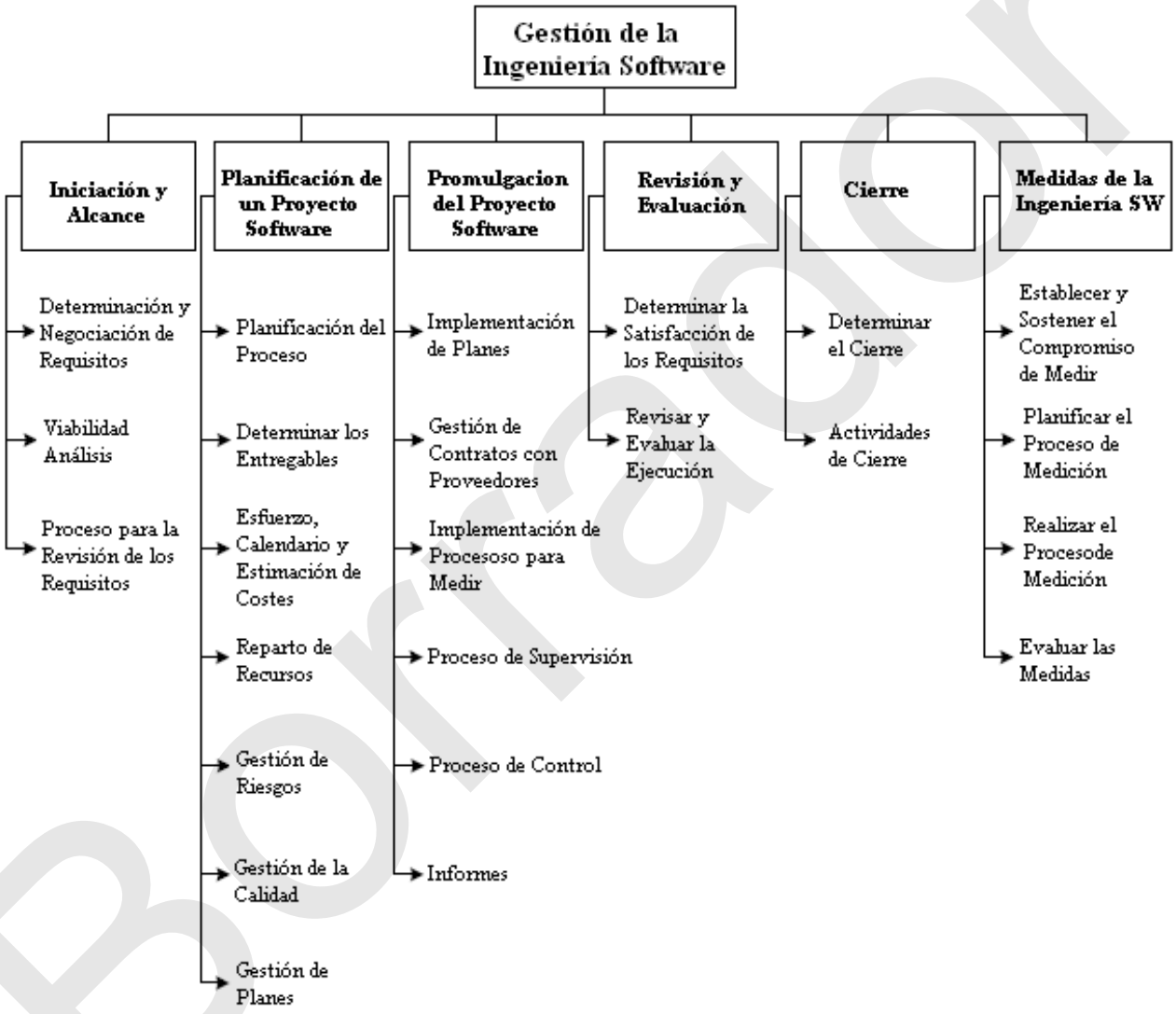
- *Definición de iniciación y alcance*, que trata de la decisión de iniciar un proyecto de ingeniería del software.
- *Planificación del proyecto de software*, que afronta las actividades emprendidas para prepararse para una ingeniería del software exitosa desde una perspectiva de gestión.
- *Promulgación del proyecto de software*, que trata de las actividades de gestión de ingeniería del software

ampliamente aceptadas que tienen lugar durante la ingeniería del software.

- *Repaso y evaluación*, que buscan asegurarse de que el software es satisfactorio.
- *Cierre*, que afronta las actividades de post-realización de un proyecto de ingeniería del software.
- *Medición de la ingeniería del software*, que trata del desarrollo e implementación eficaz de los programas de medición en las organizaciones de ingeniería del software (IEEE12207.0-96)

La descomposición de los temas del KA de Gestión de Ingeniería del Software se muestra en la Figura 1

Figura 1 Descomposición de los temas del KA de Gestión de Ingeniería del Software



1. Iniciación y Alcance

El enfoque de este conjunto de actividades se centra en la determinación eficaz de los requisitos del software por medio de varios métodos de inducción y la valoración de la viabilidad del proyecto desde distintos puntos de vista. Una vez que se ha establecido la viabilidad, la tarea pendiente dentro de este proceso es la especificación de la validación de requisitos y del cambio de procedimientos (ver también el KA de Requisitos del Software).

1.1. Determinación y Negociación de los Requisitos

[Dor02: v2c4; Pf101: c4; Pre04: c7; Som05: c5]

Los métodos de requisitos del software para la inducción de los requisitos (por ejemplo, observación), análisis (por ejemplo, modelado de datos, modelado de casos de uso), especificación y validación (por ejemplo, prototipado) deben seleccionarse y aplicarse, tomando en cuenta las distintas perspectivas del contratista. Esto lleva a la determinación del alcance del proyecto, de los objetivos, y de las restricciones. Ésta siempre es una actividad importante, ya que fija las fronteras visibles para el conjunto de tareas que se emprenden, y sucede así especialmente donde la tarea es de gran novedad. Puede encontrarse información adicional en el KA de los Requisitos del Software.

Viabilidad, Análisis (Técnico, Operacional, Financiero, Social/Político)

[Pre04: c6; Som05: c6]

Se debe asegurar a los ingenieros del software que hay disponibles capacidad y recursos adecuados en forma de personas, pericia, medios, infraestructura y apoyo (sea interna o externamente) para cerciorarse de que el proyecto pueda completarse con éxito de un modo oportuno y rentable (utilizando, por ejemplo, una matriz de requisitos y capacidades). A menudo esto requiere un cálculo del esfuerzo y del coste basado en los métodos adecuados (por ejemplo, técnicas de analogía reguladas por expertos).

Construir para verificar

Dado que los cambios son inevitables, es de vital importancia que desde el inicio se llegue a un acuerdo entre los contratistas acerca de los medios por los que se repasarán y revisarán el alcance y requisitos (por ejemplo, por medio de procedimientos pactados para la gestión de cambios). Esto claramente implica que el alcance y los requisitos no quedarán "grabados en piedra" sino que pueden y deben volverse a revisar en puntos predeterminados según se vaya desenvolviendo el proyecto (por ejemplo, en las revisiones del diseño, en las revisiones de gestión). Si se aceptan los cambios, deberá usarse entonces algún tipo de análisis de trazabilidad y de análisis de riesgos (ver el apartado 2.5 *Gestión de Riesgos*) para determinar el impacto de esos cambios. También resultará útil un acercamiento que gestione los cambios cuando llegue el momento de repasar los resultados del proyecto, ya que el alcance y los requisitos tendrán que ser la base para evaluar el éxito. [Som05: el c6] Ver también la subárea de control de la configuración del software del KA de Gestión de la Configuración del Software.

2. Planificación de un Proyecto de Software

El proceso de planificación iterativa está regulado por el alcance y requisitos, y por el establecimiento de la viabilidad. A estas alturas, se evalúan los procesos del ciclo de vida del software y se selecciona el más apropiado (considerando la naturaleza del proyecto, su grado de novedad, su complejidad funcional y técnica, sus requisitos de calidad, etc.).

Si la situación lo aconseja, se planea entonces el propio proyecto en la forma de una descomposición jerárquica de tareas, se especifican y caracterizan los entregables asociados de cada tarea en términos de calidad y de otros atributos en la línea de los requisitos declarados, y se emprende la descripción detallada del esfuerzo de realización, el calendario y la estimación de costes.

Más adelante se asignan los recursos a las tareas para optimizar la productividad del personal (a nivel de individuo, de equipo y organizacional), el uso de equipos y materiales, y la adhesión a los horarios. Se emprende una gestión de riesgos detallada y se discute el "perfil de riesgo" entre todos los contratistas de relieve, llegando a un acuerdo. Se determinan los procesos comprehensivos de gestión de la calidad del software como parte del proceso en términos de procedimientos y responsabilidades para asegurar la calidad del software, la verificación y la validación, las revisiones y las auditorías (ver el KA de la Calidad del Software). Ya que es un proceso iterativo, resulta de vital importancia que se declaren y pacten con claridad los procesos y responsabilidades para la gestión, repaso y revisión del plan en ejecución.

Planificación de un Proceso

La selección de un modelo adecuado de ciclo de vida del software (por ejemplo, un prototipado evolutivo en espiral) y la adaptación y el despliegue de ciclos de vida del software, se emprenden a la luz del alcance particular y de los requisitos del proyecto. También se seleccionan métodos y herramientas pertinentes. [Dor02: el v1c6,v2c8; Pfl01: el c2; Pre04: el c2; Rei02: el c1,c3,c5; Som05: el c3; Tha97: el c3] A nivel de proyecto, se utilizan métodos y herramientas adecuados para descomponer el proyecto en tareas, con entradas asociadas, resultados y condiciones de finalización de obra (por ejemplo, una estructura para la descomposición del trabajo). [Dor02: el v2c7; Pfl01: el c3; Pre04: el c21; Rei02: el c4,c5; Som05: el c4; Tha97: el c4,c6] Esto influye a su vez en las decisiones sobre el horario y estructura de la organización de alto nivel del proyecto.

Determinar los Entregables

Se especifica y caracteriza el producto o productos de cada tarea (por ejemplo, un diseño arquitectónico, un informe de inspección). [Pfl01: el c3; Pre04: el c24; Tha97: el c4] Se evalúan las oportunidades de reutilizar los componentes del software de desarrollos anteriores o de utilizar productos software del mercado. Se planifica la utilización de terceras personas y del software obtenido y se seleccionan los proveedores.

Esfuerzo, Calendario y Cálculo del Coste

Partiendo de la descomposición de tareas, entradas y resultados, se determina el rango de esfuerzo esperado que se requiere para cada tarea, utilizando un modelo de estimación calibrado basado en datos históricos sobre el esfuerzo empleado, cuando estén disponibles y sean pertinentes, u otros métodos como el juicio de un especialista. Se establecen las dependencias de las tareas y se identifican los cuellos de botella potenciales utilizando los métodos convenientes (por ejemplo, el análisis del camino crítico). Cuando sea posible se solucionan los cuellos de botella, y se elabora el esperado cuadro de tareas con los horarios de inicio, duraciones y horarios de finalización bien planificados (por ejemplo, el diagrama PERT). Los requisitos de recursos (personas, herramientas) se traducen en estimaciones de costo. [Dor02: el v2c7; Fen98: el c12; Pfl01: el c3; Pre04: el c23, el c24,; Rei02: el c5,c6; Som05: el c4,c23; Tha97: el c5] Ésta es una actividad muy iterativa que debe ser negociada y revisada hasta que se alcance un acuerdo general entre los contratistas afectados (principalmente de ingeniería y gestión).

2.4 Reparto de Recursos

[Pf101: c3; Pre04: c24; Rei02: c8,c9; Som05: c4; Tha97: c6,c7]

Los equipos, medios y personas se asocian a las tareas programadas, incluyendo la asignación de responsabilidades de cara a su completa realización (usando, por ejemplo, un diagrama de Gantt). Esta actividad está regulada y limitada por la disponibilidad de los recursos y por su uso óptimo bajo estas circunstancias, así como por temas relacionados con el personal (por ejemplo, productividad de los individuos y equipos, dinámicas de equipo, estructuras organizativas y de equipo).

2.5 Gestión de Riesgos

Se lleva a cabo la identificación y análisis de riesgos (lo que puede salir mal, cómo y por qué, y sus posibles consecuencias), la valoración crítica de riesgos (cuáles son los riesgos más significativos a los que se está expuestos, sobre cuáles podemos hacer algo en cuanto a su influencia), la mitigación de riesgos y la planificación de contingencias (formulando una estrategia para controlar los riesgos y gestionar los perfiles de riesgo). Los métodos de valoración de riesgos (por ejemplo, los árboles de decisión y los procesos de simulación) deben utilizarse para resaltar y evaluar riesgos. A estas alturas se deben determinar las políticas de abandono de proyectos en conversaciones con todos los otros contratistas. [Dor02: el v2c7; Pfl01: el c3; Pre04: el c25; Rei02: el c11; Som05: el c4; Tha97: el c4] La gestión de riesgos del proyecto debe influir en aspectos de riesgo únicos en el software, como la tendencia de los ingenieros del software a agregar utilidades no deseadas o como el controlador de riesgos presente en la naturaleza intangible del software.

2.6 Gestión de Calidad

[Dor02: v1c8,v2c3-c5; Pre04: c26; Rei02: c10; Som05: c24,c25; Tha97: c9,c10]

La calidad se define en términos de atributos pertinentes al proyecto específico y en los de cualquier producto o productos asociados a ella, probablemente tanto en términos cuantitativos como cualitativos. Estas características de la calidad habrán sido determinadas en la especificación de requisitos detallados del software. Ver también el KA de los Requisitos del Software.

Los límites de adhesión a la calidad para cada indicador se colocan de acuerdo a las expectativas que tenga el contratista sobre el software en cuestión. Los procedimientos que hacen referencia a lo largo del proceso a la SQA en curso a lo largo del proceso y a la verificación y validación del producto (entregable) también se especifican en esta fase (por ejemplo, las revisiones técnicas e inspecciones) (ver también el KA de la Calidad del Software).

2.7 Gestión de Planes

[Som05: c4; Tha97: c4]

También se ha de planificar cómo se gestionará el proyecto y cómo se gestionará la planificación. Los informes, la supervisión y el control del proyecto deben encajar en el proyecto de ingeniería del software seleccionado y en las realidades del proyecto, y deben reflejarse en los varios artefactos que se usarán para gestionarlo. Pero, en un contexto en donde los cambios son más una expectativa que un susto, es de vital importancia que se gestionen los propios planes. Esto requiere que sistemáticamente se dirija, supervise, repase, informe y, donde así lo requiera, revise, la adhesión a los planes. Los planes asociados a otros procesos de soporte orientados a gestión (por ejemplo, documentación, gestión de la configuración del software, y resolución de problemas) también necesitan gestionarse de esa misma manera.

3. Promulgación del Proyecto de Software

A continuación se ejecutan los planes y se promulgan los procesos incluidos en los planes. A lo largo de este proceso todo se centra en la adhesión a los planes, con

una expectativa arrolladora de que tal adhesión llevará a la satisfacción plena de los requisitos del contratista y al logro de los objetivos del proyecto. Las actividades actuales de gestión para medir, supervisar, controlar e informar son fundamentales para la promulgación.

3.1 Implementación de Planes

[Pf101: c3; Som05: c4]

Inicia el proyecto y se emprenden las actividades del proyecto según el horario. En el proceso, se utilizan recursos (por ejemplo, esfuerzo del personal, financiación) y se producen entregables (por ejemplo, documentos de diseño de arquitectura, casos de pruebas).

3.2 Gestión de Contratos con Proveedores

[Som05: c4]

Preparar y ejecutar acuerdos con los proveedores, supervisar la actuación del proveedor, y aceptar sus productos, incorporándolos cuando sean adecuados.

3.3 Implementación de Procesos para Medir

[Fen98: c13,c14; Pre04: c22; Rei02: c10,c12; Tha97: c3,c10]

Se promulga el proceso de medición junto con el proyecto del software, asegurándose de que se recogen datos relevantes y útiles (ver también los apartados 6.2 *Planificar el Proceso de Medición* y 6.3 *Realizar el Proceso de Medición*).

3.4 Proceso de Supervisión

[Dor02: v1c8, v2c2-c5,c7; Rei02: c10; Som05: c25; Tha97: c3;c9]

Se evalúa continuamente y a intervalos predeterminados la adhesión a los diferentes planes. Se analizan los resultados y las condiciones de acabado de cada tarea. Se evalúan los entregables en términos de las características que ellos requieren (por ejemplo, por medio de revisiones y auditorías). Se investiga el consumo de fuerzas, la adhesión a horarios, y los costes a día de hoy, y se examina el uso de recursos. Se revisa de nuevo el perfil de riesgo del proyecto y se evalúa la adhesión a los requisitos de calidad.

Se modelan y analizan los datos de medición. Se emprende el análisis de variación basado en la desviación actual de los resultados y valores esperados. Esto puede darse en forma de desbordamiento de costes, equivocaciones en el horario y similares. Se lleva a cabo la identificación de la desviación y el análisis de calidad y otros datos de medición (por ejemplo, el análisis de la densidad de los defectos). Se recalculan la exposición a riesgos y sus influencias y se ejecutan de nuevo los árboles de decisiones, las simulaciones, etc., a la luz de los nuevos datos. Estas actividades permiten la detección de problemas y la identificación de excepciones basada en la superación de los límites existentes. Se informa de los resultados según se vaya necesitando y sobre todo cuando se hayan superado los límites aceptables.

3.5 Proceso de Control

[Dor02: v2c7; Rei02: c10; Tha97: c3,c9]

Los resultados de las actividades de supervisión del proceso proporcionan la base sobre la que se toman las decisiones para actuar. Se pueden hacer cambios al proyecto cuando se juzgue oportuno y cuando se modele y gestione el impacto y los riesgos asociados a éstos. Esto puede tomar la forma de una acción correctiva (por ejemplo, volviendo a probar ciertos componentes), puede que involucre la incorporación de contingencias para evitar sucesos semejantes (por ejemplo, la decisión de utilizar prototipados para ayudar en la validación de los requisitos del software), y/o puede implicar la revisión de los distintos planes y de otros documentos del proyecto (por ejemplo, la especificación de requisitos) para corregir los resultados inesperados y sus implicaciones.

En algunos casos, puede llevar al abandono del proyecto. En todos los casos, se adhiere al control de cambios y a los procedimientos de gestión de configuración del software (ver también el KA de la Gestión de Configuración del Software) se documentan y comunican decisiones a todos los implicados importantes, se repasan los planes y si es necesario se revisan, y todos los datos importantes se graban en la base de datos central (ver también el apartado 6.3 *Llevar a cabo el Proceso de Revisión).*

3.6 Informes

[Rei02: c10; Tha97: c3,c10]

En períodos específicos y concertados, se informa de la adhesión a los planes dentro de la organización (por ejemplo al comité de dirección de cartera del proyecto) y a los contratistas externos (por ejemplo, clientes, usuarios). Informes de esta naturaleza deben orientarse hacia una adhesión global en oposición a los informes detallados que se requieren frecuentemente dentro del equipo de proyecto.

4. Revisión y Evaluación

En puntos críticos del proyecto, se evalúan el progreso global hacia el logro de los objetivos prefijados y la satisfacción de los requisitos del contratista. De igual modo, en hitos particulares se llevan a cabo valoraciones sobre la efectividad del proceso global hasta la fecha, del personal involucrado, y de las herramientas y métodos utilizados.

4.1 Determinar la Satisfacción de los Requisitos

[Rei02: c10; Tha97: c3,c10]

Ya que uno de nuestros objetivos principales consiste en lograr la satisfacción del contratista (usuario o cliente), es importante que el progreso hacia este

objetivo sea evaluado formal y periódicamente. Esto ocurre al lograr los principales hitos del proyecto (por ejemplo, la confirmación de la arquitectura del diseño de software, la revisión técnica de la integración del software). Se identifican variaciones a las expectativas y se llevan a cabo acciones adecuadas. Al igual que en la actividad de control del proceso arriba indicada (ver el apartado 3.5 *Proceso de Control*), en todos los casos, se adhiere al control de cambios y a los procedimientos de gestión de configuración del software (ver también el KA de la Gestión de Configuración del Software) se documentan y comunican decisiones a todos los implicados importantes, se repasan los planes y si es necesario se revisan, y todos los datos importantes se graban en la base de datos central (ver también el apartado 6.3 *Llevar a cabo el Proceso de Revisión).* Se puede encontrar más información en el KA de las Pruebas del Software, en el apartado 2.2 *Objetivos de las Pruebas* y en el KA de la Calidad del Software, en el apartado 2.3 *Revisiones y Auditorías.*

4.2 Revisar y Evaluar la Ejecución

[Dor02: v1c8,v2c3,c5; Pfl01: c8,c9; Rei02: c10; Tha97: c3,c10]

Las revisiones periódicas de lo realizado, dirigidas al personal del proyecto, proporcionan detalles sobre la probabilidad de ser fiel a los planes así como sobre las posibles áreas de dificultad (por ejemplo, conflictos entre miembros del equipo). Se evalúan los distintos métodos, herramientas y técnicas empleadas para ver su eficacia y adecuación, y se valora sistemática y periódicamente el propio proceso para conocer su relevancia, utilidad y eficacia en el contexto del proyecto. Cuando se juzga necesario, se llevan a cabo y se gestionan los cambios.

5. Cierre

El proyecto llega a su fin cuando todos los planes y procesos implicados se han promulgado y completado. En esta fase, se repasan los criterios para el éxito del proyecto. Una vez que se ha establecido el cierre, se llevan a cabo actividades de archivado, post mortem y de mejoras de los procesos.

5.1 Determinar el Cierre

[Dor02: v1c8,v2c3,c5; Rei02: c10; Tha97: c3,c10]

Se han completado las tareas tal y como se especificaron en los planes, y se confirman los criterios para lograr un acabado satisfactorio. Todos los productos planificados han sido entregados con características aceptables.

Se marca y confirma la satisfacción de los requisitos, se han logrado los objetivos del proyecto. Estos procesos por lo general involucran a todos los contratistas y acaban con la documentación tanto de la aceptación del cliente y como de los informes de cualquier otro problema pendiente conocido.

5.2 Actividades de Cierre

[Pf101: c12; Som05: c4]

Tras haberse confirmado el cierre, se archivan los materiales del proyecto de acuerdo a los métodos, localización y duración pactados con los contratistas. La base de datos de medición de la organización se pone al día con los datos finales del proyecto y se emprenden análisis post-proyecto. Se inicia un proyecto post mortem con el fin de analizar los temas, problemas y oportunidades encontrados durante el proceso (particularmente por medio de revisiones y evaluaciones, ver el subárea 4 *Revisión y Evaluación*) y se sacan lecciones del proceso que luego alimentan los conocimientos de la organización y los intentos de mejora (ver también el KA del Proceso de Ingeniería del Software).

6. Medidas de la Ingeniería del Software

[ISO 15939-02]

La importancia de la medición y su papel en las buenas prácticas de gestión está ampliamente reconocido, y es tal que su importancia sólo puede aumentar en los próximos años. Medir con eficacia se ha convertido en una de las piedras angulares de la madurez de una organización. Las palabras claves para la medición del software y para los métodos de medición fueron definidas en [ISO15939-02] basadas en el vocabulario internacional de metrología ISO [ISO93]. No obstante, los lectores encontrarán en la literatura existente diferencias en la terminología; por ejemplo, el término "métrica" a veces se usa en lugar de "medición".

Este apartado sigue el estándar internacional ISO/IEC 15939, que describe el proceso que define las actividades y tareas necesarias para implementar un proceso de medición e incluye, asimismo, un modelo de medición de la información.

6.1 Establecer y Sostener el Compromiso de Medir

- Aceptar los requisitos de medición. Cada tentativa de medición debe estar guiada por objetivos organizacionales, e impulsada por un conjunto de requisitos de medición establecidos por la organización y por el proyecto. Por ejemplo, un objetivo organizacional podría ser “ser los primeros en salir al mercado con los nuevos productos." [Fen98: c3,c13; Pre04: c22] Esto a su vez podría generar un requisito para que se midan los factores que contribuyen a este objetivo, y así se puedan gestionar los proyectos para hacer frente a este objetivo.
  - Definir el alcance de la medición. Se debe establecer la unidad organizacional a la que se va a aplicar cada requisito de medición. Esto puede consistir en un área funcional, en un solo proyecto, en un solo sitio, o incluso en toda la empresa. Todas las subsiguientes tareas de medición relacionadas con este requisito

deben encontrarse dentro del alcance definido. Además, se debe identificar a los contratistas implicados.

- Compromiso de la dirección y del personal con la medición. El compromiso debe establecerse formalmente, debe comunicarse, y debe apoyarse en los recursos (ver el siguiente artículo).

♦ Empeñar recursos para la medición. El compromiso de la organización con la medición es un factor esencial para el éxito, como demuestra la asignación de recursos para llevar a cabo el proceso de medición. El asignar recursos incluye el reparto de responsabilidades para las diferentes tareas del proceso de medición (tales como usuario, analista y bibliotecario) y el proporcionar una financiación, entrenamiento, herramientas, y apoyo adecuados para dirigir el proceso de un modo perdurable.

6.2 Planificar el Proceso de Medición

♦ Caracterizar la unidad organizacional. La unidad organizacional proporciona el contexto para la medición así que es importante hacer explícito este contexto y articular las presunciones que éste incluye y las restricciones que va imponiendo. La caracterización puede darse en términos de procesos organizacionales, dominios de aplicaciones, tecnología e interfaces organizacionales. Un modelo de proceso organizacional también es por lo general un elemento de una caracterización de la unidad organizacional [ISO15939-02: 5.2.1].

♦ Identificar las necesidades de información. Las necesidades de información se basan en las metas, restricciones, riesgos y problemas de la unidad organizacional. Éstas pueden proceder de objetivos comerciales, organizacionales, reguladores y/o del producto. Deben ser identificadas y deben priorizarse. Debe entonces seleccionarse un subconjunto para ser cotejado y los resultados deben ser documentados, comunicados y revisados por los contratistas [ISO 15939-02: 5.2.2].

♦ Seleccionar las mediciones. Se deben elegir las mediciones candidatas al puesto, claramente vinculadas a las necesidades de información. Las mediciones deben seleccionarse en base a las prioridades de las necesidades de información y otros criterios como el coste de recolección de datos, el grado de trastorno del proceso durante la recolección, la facilidad de análisis, la facilidad de obtener datos precisos y consistentes, etc. [ISO15939-02: 5.2.3 y Apéndice C].

♦ Definir la recolección de datos, el análisis y los procedimientos para informar. Esto abarca los procedimientos y horarios de recolección, y la gestión de datos de almacenamiento, verificación, análisis, informes y configuración [ISO15939-02: 5.2.4].

♦ Definir los criterios para evaluar los productos de información. Los criterios para evaluar están influenciados por los objetivos técnicos y comerciales de la unidad organizacional. Los productos de información incluyen los asociados con el producto que está siendo elaborado, así como los asociados con los procesos utilizados para gestionar y medir el proyecto [ISO15939-02: 5.2.5 y Apéndices D y E].

♦ Revisar, aprobar y proporcionar recursos para las tareas de medición.

- El plan de medición debe ser revisado y aprobado por los contratistas adecuados. Esto incluye todos los procedimientos de recolección de datos y los procedimientos de almacenamiento, análisis e informes; los criterios de evaluación; horarios y responsabilidades. Los criterios para revisar estos artefactos tendrían que haberse establecido a un nivel de unidad organizacional o superior y debieran usarse como base para estas revisiones. Tales criterios deben tomar en cuenta experiencias anteriores, disponibilidad de recursos, y trastornos potenciales del proceso cuando se proponen cambios a las prácticas actuales. La aprobación demuestra que existe un compromiso con el proceso de medición [ISO15939-02: 5.2.6.1 y Apéndice F].

- Hay que hacer que los recursos estén disponibles para implementar las tareas de medición planeadas y aprobadas. La disponibilidad de los recursos puede organizarse en aquellos casos en donde los cambios han de pilotarse antes de un amplio despliegue. Se debe prestar atención a los recursos necesarios para un amplio despliegue de los nuevos procedimientos o mediciones [ISO15939-02: 5.2.6.2].

♦ Adquirir y desplegar tecnologías de apoyo. Esto incluye una evaluación de las tecnologías de apoyo disponibles, la selección de las tecnologías más adecuadas, la adquisición de esas tecnologías, y el despliegue de esas tecnologías [ISO 15939-02:5.2.7].

6.3 Realizar el Proceso de Medición

♦ Integrar los procedimientos de medición con los procesos pertinentes. Los procedimientos de medición, tales como la recolección de datos, deben integrarse en los procesos que están midiendo. Esto puede que implique cambiar los procesos actuales para adaptar la recolección de datos o las actividades de generación. Puede también implicar el análisis de los actuales

procesos para minimizar esfuerzos adicionales y evaluaciones del efecto en los empleados, con el fin de asegurarse de que serán aceptados los procedimientos de medición. Se necesita considerar los temas morales y otros factores humanos. Además, los procedimientos de medición deben comunicarse a los proveedores de datos, puede que se tenga que proporcionar entrenamiento, y se debe proporcionar el típico apoyo. De manera similar, el análisis de datos y los procedimientos de información deben tener la típica integración en los procesos organizacionales y/o del proyecto [ISO 15939-02: 5.3.1].

- Recolectar datos. Se debe recolectar, verificar y almacenar datos [ISO 1539-02: 5.3.2].

- Analizar datos y desarrollar productos de información. Se pueden agregar, transformar o recodificar datos como parte del proceso de análisis, utilizando un grado de rigor adecuado a la naturaleza de los datos y las necesidades de información. Los resultados de este análisis son indicadores típicos, tales como gráficas, números u otras indicaciones que han de ser interpretadas, dando lugar a conclusiones iniciales que han de presentar los contratistas. Los resultados y conclusiones han de consultarse, utilizando un proceso definido por la organización (que puede ser formal o informal). Los proveedores de datos y los usuarios de las mediciones deben participar en revisar los datos para asegurar que son significativos y precisos, y que puede llevar a acciones razonables [ISO 15939-02: 5.3.3 y Apéndice G].

- Comunicar los resultados. Los productos de información deben documentarse y comunicarse a los usuarios y contratistas [ISO 15939-02: 5.3.4].

6.4 Evaluar las Mediciones

- Evaluar los productos de información. Evaluar los productos de información contrastándolos con los criterios de evaluación específicos y determinar las fuerzas y debilidades de los productos de información. Esto puede realizarlo un proceso interno o una auditoría externa y debe incluir una retroalimentación de los usuarios de las mediciones. Grabar las lecciones aprendidas en una base de datos adecuada [ISO 15939-02: 5.4.1 y Apéndice D].

- Evaluar el proceso de medición. Evaluar el proceso de medición contrastándolo con los criterios de evaluación específicos y determinar las fuerzas y debilidades de los procesos. Esto puede realizarlo un proceso interno o una auditoría externa y debe incluir una retroalimentación de los usuarios de las mediciones. Grabar las lecciones aprendidas en una base de datos adecuada [ISO 15939-02: 5.4.1 y Apéndice D].

- Identificar las mejoras potenciales. Tales mejoras pueden consistir en cambios en el formato de los indicadores, cambios en las unidades medidas, o en la reclasificación de las categorías. Determinar los costes y beneficios de las mejoras potenciales y seleccionar las acciones de mejora adecuadas. Comunicar las mejoras propuestas al dueño del proceso de medición y a los contratistas para su revisión y aprobación. Comunicar también la falta de mejoras potenciales si el análisis no identifica ninguna mejora [ISO 15939-02: 5.4.2].

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Dor02]	[ISO15239-02]	[Fen98]	[Pfl01]	[Pre04]	[Rei02]	[Som05]	[Tha97]
1.Iniciación y Alcance								
1.1 Determinación y Negociación de Requisitos	v2c4			c4	c7		c5	
1.2 Viabilidad, Análisis					c6		c6	
1.3 Construir para Verificar							c6	
2. Planificación de un Proyecto Software								
2.1 Planificación de un Proceso	v1c6, v2c7, v2c8			c2, c3	c2, c21	c1, c3, c5	c3, c4	c3, c4, c6
2.2 Determinar los entregables				c3	c24			c4
2.3 Esfuerzo, Horario y Cálculo del Coste	v2c7		c12	c3	c23, c24	c5, c6	c4, c23	c5
2.4 Reparto de Recursos				c3	c24	c8, c9	c4	c6, c7
2.5 Gestión de Riesgos	v2c7			c3	c25	c11	c4	c4
2.6 Gestión de Calidad	v1c8, v2c3-c5				c26	c10	c24, c25	c9, c10
2.7 Gestión de Planes							c4	c4
3. Promulgación del Proyecto Software								
3.1 Implementación de Planes				c3			c4	
3.2 Gestión de Contratos con Proveedores							c4	
3.3 Implementación de Procesos para medir			c13, c14		c22	c10, c12		c3, c10
3.4 Proceso de Supervisión	v1c8, v2c2-c5, c7					c10	c25	c3, c10
3.5 Proceso de Control	v2c7					c10		c3, c9
3.6 Informes						c10		c3, c10
4. Revisión y Evaluación								
4.1 Determinar la satisfacción de los Requisitos						c10		c3, c10
4.2 Revisar y Evaluar la Ejecución	v1c8, v2c3, c5			c8, c9		c10		c3, c10
5. Cierre								
5.1 Determinar el Cierre	v1c8, v2c3, c5					c10		c3, c10
5.2 Actividades del Cierre				c12			c4	
6. Medida de la Ingeniería del Software		*						
6.1 Establecer y Sostener el compromiso de Medir			c3, c13		c22			
6.2 Planificar el Proceso de Medición		c5, C,D,E,F						
6.3 Realizar el Proceso de Medición		c5, G						
6.4 Evaluar las Mediciones		c5, D						

REFERENCIAS RECOMENDADAS PARA LA GESTIÓN DEL SOFTWARE

[Dor02] M. Dorfman and R.H. Thayer, eds., *Software Engineering*, IEEE Computer Society Press, 2002, Vol. 1, Chap. 6, 8, Vol. 2, Chap. 3, 4, 5, 7, 8.

[Fen98] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, second ed., International Thomson Computer Press, 1998, Chap. 1-14.

[ISO15939-02] ISO/IEC 15939:2002, *Software Engineering — Software Measurement Process*, ISO and IEC, 2002.

[Pfl01] S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001, Chap. 2-4, 8, 9, 12, 13.

[Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, sixth ed., McGraw-Hill, 2004, Chap. 2, 6, 7, 22-26.

[Rei02] D.J. Reifer, ed., *Software Management*, IEEE Computer Society Press, 2002, Chap. 1-6, 7-12, 13.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005, Chap. 3-6, 23-25.

[Tha97] R.H. Thayer, ed., Software Engineering Project Management, IEEE Computer Society Press, 1997, Chap. 1-10.

APÉNDICE A. LISTA DE LECTURAS ADICIONALES

(Adl99) T.R. Adler, J.G. Leonard, and R.K. Nordgren, "Improving Risk Management: Moving from Risk Elimination to Risk Avoidance*," Information and Software Technology*, vol. 41, 1999, pp. 29-34.

(Bai98) R. Baines, "Across Disciplines: Risk, Design, Method, Process, and Tools," *IEEE Software*, July/August 1998, pp. 61-64.

(Bin97) R.V. Binder, "Can a Manufacturing Quality Model Work for Software?" *IEEE Software*, September/October 1997, pp. 101-102,105.

(Boe97) B.W. Boehm and T. DeMarco, "Software Risk Management," *IEEE Software*, May/June 1997, pp. 17-19.

(Bri96) L.C. Briand, S. Morasca, and V.R. Basili, "Property-Based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, vol. 22, iss. 1, 1996, pp. 68-86.

(Bri96a) L. Briand, K.E. Emam, and S. Morasca, "On the Application of Measurement Theory in Software Engineering," *Empirical Software Engineering*, vol. 1, 1996, pp. 61-88.

(Bri97) L.C. Briand, S. Morasca, and V.R. Basili, "Response to: Comments on 'Property-based Software Engineering Measurement: Refining the Addivity Properties,'" *IEEE Transactions on Software Engineering*, vol. 23, iss. 3, 1997, pp. 196-197.

(Bro87) F.P.J. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, Apr. 1987, pp. 10-19.

(Cap96) J. Capers, *Applied Software Measurement: Assuring Productivity and Quality*, second ed., McGraw-Hill, 1996.

(Car97) M.J. Carr, "Risk Management May Not Be For Everyone," *IEEE Software*, May/June 1997, pp. 21-24.

(Cha96) R.N. Charette, "Large-Scale Project Management Is Risk Management," *IEEE Software*, July 1996, pp. 110-117.

(Cha97) R.N. Charette, K.M. Adams, and M.B. White, "Managing Risk in Software Maintenance," *IEEE Software*, May/June 1997, pp. 43-50.

(Col96) B. Collier, T. DeMarco,and P. Fearey, "A Defined Process for Project Postmortem Review," *IEEE Software*, July 1996, pp. 65-72.

(Con97) E.H. Conrow and P.S. Shishido, "Implementing Risk Management on Software Intensive Projects," *IEEE Software*, May/June 1997, pp. 83-89.

(Dav98) A.M. Davis, "Predictions and Farewells," *IEEE Software*, July/August 1998, pp. 6-9.

(Dem87) T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, Dorset House Publishing, 1987.

(Dem96) T. DeMarco and A. Miller, "Managing Large Software Projects," *IEEE Software*, July 1996, pp. 24-27.

(Fav98) J. Favaro and S.L. Pfleeger, "Making Software Development Investment Decisions," *ACM SIGSoft Software Engineering Notes*, vol. 23, iss. 5, 1998, pp. 69-74.

(Fay96) M.E. Fayad and M. Cline, "Managing Object-Oriented Software Development," *Computer*, September 1996, pp. 26-31.

(Fen98) N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, second ed., International Thomson Computer Press, 1998.

(Fle99) R. Fleming, "A Fresh Perspective on Old Problems," *IEEE Software*, January/February 1999, pp. 106-113.

(Fug98) A. Fuggetta et al., "Applying GQM in an Industrial Software Factory," *ACM Transactions on Software Engineering and Methodology*, vol. 7, iss. 4, 1998, pp. 411-448.

(Gar97) P.R. Garvey, D.J. Phair, and J.A. Wilson, "An Information Architecture for Risk Assessment and Management," *IEEE Software*, May/June 1997, pp. 25-34.

(Gem97) A. Gemmer, "Risk Management: Moving beyond Process," *Computer*, May 1997, pp. 33-43.

(Gla97) R.L. Glass, "The Ups and Downs of Programmer Stress," *Communications of the ACM*, vol. 40, iss. 4, 1997, pp. 17-19.

(Gla98) R.L. Glass, "Short-Term and Long-Term Remedies for Runaway Projects," *Communications of the ACM*, vol. 41, iss. 7, 1998, pp. 13-15.

(Gla98a) R.L. Glass, "How Not to Prepare for a Consulting Assignment, and Other Ugly Consultancy Truths," *Communications of the ACM*, vol. 41, iss. 12, 1998, pp. 11-13.

(Gla99) R.L. Glass, "The Realities of Software Technology Payoffs*," Communications of the ACM,* vol. 42, iss. 2, 1999, pp. 74-79.

(Gra99) R. Grable et al., "Metrics for Small Projects: Experiences at the SED," *IEEE Software*, March/April 1999, pp. 21-29.

(Gra87) R.B. Grady and D.L. Caswell, *Software Metrics: Establishing A Company-Wide Program*. Prentice Hall, 1987.

(Hal97) T. Hall and N. Fenton, "Implementing Effective Software Metrics Programs," *IEEE Software*, March/April 1997, pp. 55-64.

(Hen99) S.M. Henry and K.T. Stevens, "Using Belbin's Leadership Role to Improve Team Effectiveness: An Empirical Investigation," *Journal of Systems and Software*, vol. 44, 1999, pp. 241-250.

(Hoh99) L. Hohmann, "Coaching the Rookie Manager," *IEEE Software*, January/February 1999, pp. 16-19.

(Hsi96) P. Hsia, "Making Software Development Visible," *IEEE Software*, March 1996, pp. 23-26.

(Hum97) W.S. Humphrey, *Managing Technical People: Innovation, Teamwork, and the Software Process*: Addison-Wesley, 1997.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

(Jac98) M. Jackman, "Homeopathic Remedies for Team Toxicity," *IEEE Software*, July/August 1998, pp. 43-45.

(Kan97) K. Kansala, "Integrating Risk Assessment with Cost 8–12 © IEEE – 2004 Version Estimation," *IEEE Software*, May/June 1997, pp. 61-67.

(Kar97) J. Karlsson and K. Ryan, "A Cost-Value Aproach for Prioritizing Requirements," *IEEE Software*, September/October 1997, pp. 87-74.

(Kar96) D.W. Karolak, Software Engineering Risk Management, *IEEE Computer* Society Press, 1996.

(Kau99) K. Kautz, "Making Sense of Measurement for Small Organizations," *IEEE Software*, March/April 1999, pp. 14-20.

(Kei98) M. Keil et al., "A Framework for Identifying Software Project Risks," *Communications of the ACM*, vol. 41, iss. 11, 1998, pp. 76-83.

(Ker99) B. Kernighan and R. Pike, "Finding Performance Improvements," *IEEE Software*, March/April 1999, pp. 61-65.

(Kit97) B. Kitchenham and S. Linkman, "Estimates, Uncertainty, and Risk*," IEEE Software*, May/June 1997, pp. 69-74.

(Lat98) F. v. Latum et al., "Adopting GQM-Based Measurement in an Industrial Environment," *IEEE Software*, January-February 1998, pp. 78-86.

(Leu96) H.K.N. Leung, "A Risk Index for Software Producers," *Software Maintenance: Research and Practice*, vol. 8, 1996, pp. 281-294.

(Lis97) T. Lister, "Risk Management Is Project Management for Adults," IEEE *Software*, May/June 1997, pp. 20-22.

(Mac96) K. Mackey, "Why Bad Things Happen to Good Projects," *IEEE Software*, May 1996, pp. 27-32.

(Mac98) K. Mackey, "Beyond Dilbert: Creating Cultures that Work," *IEEE Software*, January/February 1998, pp. 48-49.

(Mad97) R.J. Madachy, "Heuristic Risk Assessment Using Cost Factors," *IEEE Software*, May/June 1997, pp. 51-59.

(McC96) S.C. McConnell, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, 1996.

(McC97) S.C. McConnell, *Software Project Survival Guide*, Microsoft Press, 1997.

(McC99) S.C. McConnell, "Software Engineering Principles," *IEEE Software*, March/April 1999, pp. 6-8.

(Moy97) T. Moynihan, "How Experienced Project Managers Assess Risk," *IEEE Software*, May/June 1997, pp. 35-41.

(Ncs98) P. Ncsi, "Managing OO Projects Better," *IEEE Software*, July/August 1998, pp. 50-60.

(Nol99) A.J. Nolan, "Learning From Success," *IEEE Software*, January/February 1999, pp. 97-105.

(Off97) R.J. Offen and R. Jeffery, "Establishing Software Measurement Programs," *IEEE Software*, March/April 1997, pp. 45-53.

(Par96) K.V.C. Parris, "Implementing Accountability," *IEEE Software*, July/August 1996, pp. 83-93.

(Pfl97) S.L. Pfleeger, "Assessing Measurement (Guest Editor's Introduction)," *IEEE Software*, March/April 1997, pp. 25-26.

(Pfl97a) S.L. Pfleeger et al., "Status Report on Software Measurement," *IEEE Software*, March/April 1997, pp. 33-43.

(Put97) L.H. Putman and W. Myers, *Industrial Strength Software — Effective Management Using Measurement*, IEEE Computer Society Press, 1997.

(Rob99) P.N. Robillard, "The Role of Knowledge in Software Development," *Communications of the ACM*, vol. 42, iss. 1, 1999, pp. 87-92.

(Rod97) A.G. Rodrigues and T.M. Williams, "System Dynamics in Software Project Management: Towards the Development of a Formal Integrated Framework," *European Journal of Information Systems*, vol. 6, 1997, pp. 51-66.

(Rop97) J. Ropponen and K. Lyytinen, "Can Software Risk Management Improve System Development: An Exploratory Study," *European Journal of Information Systems*, vol. 6, 1997, pp. 41-50.

(Sch99) C. Schmidt et al., "Disincentives for Communicating Risk: A Risk Paradox," *Information and Software Technology*, vol. 41, 1999, pp. 403-411.

(Sco92) R.L. v. Scoy, "Software Development Risk: Opportunity, Not Problem," *Software Engineering Institute*, Carnegie Mellon University CMU/SEI-92-TR-30, 1992.

(Sla98) S.A. Slaughter, D.E. Harter, and M.S. Krishnan, "Evaluating the Cost of Software Quality," *Communications of the ACM*, vol. 41, iss. 8, 1998, pp. 67-73.

(Sol98) R. v. Solingen, R. Berghout, and F. v. Latum, "Interrupts: Just a Minute Never Is," *IEEE Software*, September/October 1998, pp. 97-103.

(Whi95) N. Whitten, *Managing Software Development Projects: Formulas for Success*, Wiley, 1995.

(Wil99) B. Wiley, *Essential System Requirements: A Practical Guide to Event-Driven Methods*, Addison-Wesley, 1999.

(Zel98) M.V. Zelkowitz and D.R. Wallace, "Experimental Models for Validating Technology," *Computer*, vol. 31, iss. 5, 1998, pp. 23-31.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes, IEEE, 1996.

(ISO15939-02) ISO/IEC 15939:2002, Software Engineering-Software Measurement Process, ISO and IEC, 2002.

(PMI00) Project Management Institute Standards Committee, A Guide to the Project Management Body of Knowledge (PMBOK), Project Management Institute, 2000.

CAPÍTULO 9
PROCESO DE INGENIERÍA DEL SOFTWARE

ACRÓNIMOS

CMMI	Modelo de Capacidad de Madurez Integrado
EF	Creadora de Experiencia
FP	Punto Función
HRM	Gestión de Recursos Humanos
IDEAL	(Modelo de) Iniciación-Diagnóstico-Establecimiento-Actuación-Apoyo
OMG	Grupo de Gestión de Objetos
QIP	Paradigma de Mejoras de la Calidad
SCAMPI	Evaluación Basada en el MCM para Mejoras de los Procesos utilizando la CMMI
SCE	Evaluación de la Capacidad del Software
SEPG	Grupo de Proceso de la Ingeniería del Software

INTRODUCCIÓN

El KA del Proceso de Ingeniería del Software puede examinarse en dos niveles. El primer nivel engloba las actividades técnicas y de gestión dentro de los procesos del ciclo de vida del software realizadas durante la adquisición, desarrollo, mantenimiento y retirada del software. El segundo es un meta-nivel, que se refiere a la definición, implementación, valoración, medición, gestión, cambios y mejoras de los procesos mismos del ciclo de vida del software. El primer nivel lo cubren las otras KAs en la Guía. Este KA se ocupa del segundo nivel.

El término "proceso de ingeniería del software" puede interpretarse de diversas maneras, y esto puede llevar a confusiones.

- Un significado, donde se usa la palabra *el*, como en el caso de "*el*" proceso de ingeniería del software, podría implicar que existe sólo un modo correcto de realizar tareas de ingeniería del software. En la Guía se evita este significado porque no existe tal proceso. Los estándares como IEEE12207 hablan de *procesos* de ingeniería del software, lo que significa que hay muchos procesos involucrados, tales como Procesos de Desarrollo o Proceso de Configuración de Gestión.
- Un segundo significado se refiere a una discusión general sobre procesos relacionados con la ingeniería del software. Este es el significado que se pretende con el título de esta KA y el que se usa con más frecuencia en la descripción del KA.
- Finalmente, un tercer significado podría referirse al conjunto actual de actividades realizadas dentro de una organización, que podría verse como un solo proceso, especialmente desde dentro de la organización. Se utiliza este significado en el KA en muy pocos casos.

Esta KA se aplica a cualquier parte de la gestión de los procesos del ciclo de vida del software en la que se introduzcan cambios de procedimiento o tecnológicos para la mejora de procesos o productos.

Los procesos de ingeniería del software tienen importancia no sólo para las grandes organizaciones. Más aún, las actividades relacionadas con los procesos pueden ser, y han sido, realizadas con éxito por pequeñas organizaciones, equipos e individuos.

El objetivo de gestionar los procesos del ciclo de vida del software es implementar nuevos o mejores procesos en las prácticas actuales, sean éstos individuales, proyectos u organizacionales.

Esta KA no aborda explícitamente la Gestión de Recursos Humanos (HRM), por ejemplo, como lo recoge el MCM de la Gente (Cur02) y procesos de ingeniería de sistemas [ISO1528-028; IEEE 1220-98].

También debería reconocerse que muchos temas de procesos de ingeniería del software están relacionados de cerca con otras disciplinas, tales como la gestión, incluso a veces utilizando una terminología diferente.

DESCOMPOSICIÓN DE LOS TEMAS PARA EL PROCESO DE INGENIERÍA DEL SOFTWARE

La Figura 1 muestra la descomposición de los temas en este KA:

7. Proceso de Implementación y Cambios

Ésta subárea se centra en los cambios organizacionales. Describe la infraestructura, actividades, modelos y consideraciones prácticas de un proceso de implementación y cambios.

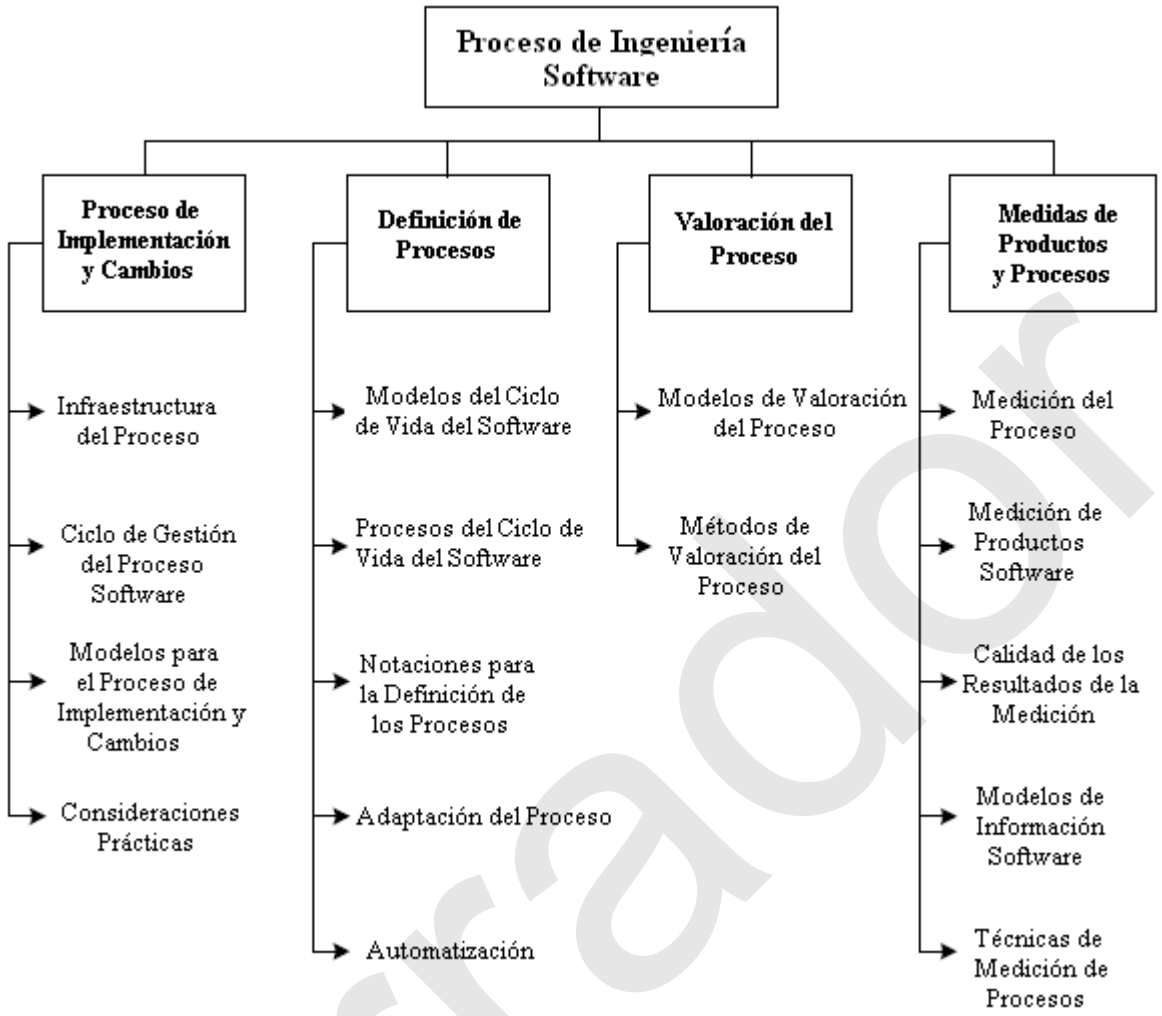


Figura 1 División de los temas para el KA del Proceso de Ingeniería Software

Aquí se describe la situación en la que los procesos se despliegan por primera vez (por ejemplo, introduciendo un proceso de inspección en un proyecto o un método que cubra todo el ciclo de vida), y donde se cambian los procesos actuales (por ejemplo, introduciendo una herramienta u optimizando un procedimiento). A esto también se le puede denominar *proceso de evolución.* En ambos casos hay que modificar las prácticas actuales. Si resulta que se extienden las modificaciones, puede que también sean necesarios cambios en la cultura organizacional.

1.2. Infraestructura del Proceso
[IEEE12207.0-96; ISO15504; SEL96]

Este tópico incluye el conocimiento relacionado con la infraestructura del proceso de ingeniería del software.

Para establecer procesos de ciclo de vida del software, es necesario que la adecuada infraestructura esté en su lugar, es decir que los recursos estén al alcance de la mano (personal competente, herramientas y financiación) y que se hayan asignado responsabilidades. El que se hayan completado estas tareas, indica el compromiso con la gestión y propiedad del esfuerzo del proceso de ingeniería del software. Puede que haya que establecer diversos comités, tales como un comité de dirección que supervise el esfuerzo del proceso de ingeniería del software.

En [McF96] se ofrece una descripción de la infraestructura de la mejora de los procesos en general. En la práctica se utilizan dos tipos principales de infraestructura: el Grupo de Proceso de Ingeniería del Software y la Creadora de Experiencia.

1.1.1 Grupo de Proceso de la Ingeniería del Software (SEPG)

Se pretende que el SEPG sea el foco central del proceso de mejoras de la ingeniería del software y tiene cierto número de responsabilidades en términos de inicialización y mantenimiento. Éstos se describen en [Fow90].

1.1.2 Creadora de Experiencia (EF)

El concepto de EF separa la organización del proyecto (la organización del desarrollo del software, por ejemplo) de la organización de las mejoras. La organización del proyecto se centra en el desarrollo y en

el mantenimiento del software, mientras que la EF se ocupa del proceso de mejoras de la ingeniería del software.

Se trata de que la EF institucionalice el aprendizaje colectivo de una organización, desarrollando, actualizando, y entregando a la organización del proyecto los *paquetes de experiencia* (por ejemplo, guías, modelos y cursos de entrenamiento), también conocidos como *validaciones de procesos*. La organización del proceso ofrece a la EF sus productos, los planes utilizados en su desarrollo, y los datos reunidos durante su desarrollo y operación. En [Bas92] se presentan ejemplos de paquetes de experiencia.

Ciclo de Gestión del Proceso del Software

[Bas92; Fow90; IEEE12207.0-96; ISO15504-98; McF96; SEL96]

La gestión de los procesos del software consiste en cuatro actividades secuenciadas en un ciclo iterativo permitiendo una retroalimentación continua y mejoras del proceso del software:

- La actividad del Establecimiento de la Infraestructura de un Proceso consiste en establecer un acuerdo con el proceso de implementación y cambios (que incluya la obtención de la gestión de compra buy-in) y levantar una adecuada infraestructura (recursos y responsabilidades) para que tenga lugar.
- El propósito de la actividad de Planificación es comprender los objetivos de las empresas actuales y las necesidades del proceso del individuo, proyecto u organización, para identificar sus fuerzas y flaquezas, y elaborar un plan para el proceso de implementación y cambios.
- El propósito del Proceso de Implementación y Cambios consiste en llevar a cabo el plan, desplegar nuevos procesos (que pueden implicar, por ejemplo, el desarrollo de herramientas y el entrenamiento del personal) y/o cambiar procesos ya existentes.
- La Evaluación del Proceso se encarga de descubrir lo bien que se ha llevado a cabo la implementación y cambios, y si se materializaron o no los beneficios esperados. Los resultados se utilizarán más adelante como entradas para ciclos subsiguientes.

Modelos Para el Proceso de Implementación y Cambios

Han surgido dos modelos generalizados para llevar a cabo el proceso de implementación y cambios que son el *Paradigma de Mejoras de la Calidad* (QIP) [SEL96] y el modelo IDEAL [McF96]. En [SEL96] se comparan los dos paradigmas. La evaluación del proceso de implementación y de los resultados de los cambios pueden ser cualitativos o cuantitativos.

Consideraciones Prácticas

El proceso de implementación y cambios constituye una instancia del cambio organizacional. Los esfuerzos de más éxito en los cambios organizacionales tratan del cambio como un proyecto en toda regla, con planes adecuados, monitoreo y revisiones.

En [Moi98; San98; Sti99] se encuentran las directrices sobre el proceso de implementación y cambios dentro de las organizaciones de ingeniería del software, incluyendo la planificación de las acciones, entrenamientos, gestión de patrocinadores, compromisos, y la selección de proyectos piloto, y abarcan tanto los procesos como las herramientas. En [EIE99a] se señalan los estudios empíricos sobre factores de éxito para los cambios en los procesos.

El papel de los agentes de cambio en esta actividad se discute en (Hut94). El proceso de implementación y cambios puede verse asimismo como una instancia de consultoría (sea interna o externa).

Uno también puede ver cambios organizacionales desde la perspectiva de la transferencia de tecnología (Rog83). Los archivos de ingeniería del software que se ocupan de la transferencia de tecnología y de las características de los recipientes de nuevas tecnologías (que podrían incluir tecnologías relacionadas con los procesos) son (Pfl99; Rag89).

Hay dos formas de acercarse la evaluación de un proceso de implementación y cambios, sea en términos de cambios al proceso mismo o en términos de cambios en las salidas de los procesos (por ejemplo, midiendo lo que te devuelve una inversión tras realizar un cambio). Una visión pragmática de lo que se puede lograr con estas evaluaciones se nos da en (Her98).

Las investigaciones sobre cómo evaluar el proceso de implementación y cambios, y los ejemplos de estudios dedicados a ello, se encuentran en [Gol99], (Kit98; Kra99; McG94).

8. Definición de Procesos

Una definición de un proceso puede ser un procedimiento, una política, o un estándar. Los procesos de ciclo de vida del software se definen por muchas razones, que incluiría el incrementar la calidad del producto, el facilitar el entendimiento y la comunicación humana, apoyar las mejoras de los procesos, apoyar la gestión de los procesos, suministrar una guía automatizada para los procesos, y suministrar un apoyo para ejecuciones automatizadas. Los tipos de definiciones de procesos requeridos dependerán, al menos parcialmente, de las razones para la definición.

Habría que señalar también que el contexto del proyecto y de la organización determinará el tipo de definición del proceso que resulte más útil. Algunas variables importantes que hay que considerar incluyen la naturaleza del trabajo (por ejemplo, mantenimiento o desarrollo), el dominio de la aplicación, el modelo de ciclo de vida, y la madurez de la organización.

Modelos de Ciclo de Vida del Software

[Pf101:c2; IEEE12207.0-96]

Los modelos del ciclo de vida del software sirven como definiciones de alto nivel de las fases que tienen lugar durante el desarrollo. No están enfocadas a ofrecer definiciones detalladas sino más bien a sobresaltar las actividades clave y sus interdependencias. Algunos ejemplos de modelos de ciclo de vida del software son el modelo de cascada, el modelo de prototipado de usar y tirar lo desechable, desarrollo evolutivo, entrega incremental/iterativa, el modelo en espiral, el modelo de software reutilizable, y la síntesis de software automatizado. Las comparaciones entre estos modelos se encuentran en [Como97], (Dav88), y hay un método para seleccionar entre muchos de ellos en (Ale91).

Procesos del Ciclo de Vida del Software

Las definiciones de los procesos de ciclo de vida del software tienden a ser más detalladas que los modelos de ciclo de vida del software. Sin embargo, los procesos del ciclo de vida del software no pretenden ordenar sus procesos en el tiempo. Esto significa que, en línea de principio, los procesos del ciclo de vida del software pueden ordenarse para tener cabida en cualquiera de los modelos del ciclo de vida del software. La principal referencia sobre esta área se encuentra en IEEE/EIA 12207.0: *Información Tecnológica – Procesos del Ciclo de Vida del Software* [IEEE 12207.0-96].

El estándar IEEE 1074:1997 para desarrollar procesos de ciclo de vida ofrece también una lista de procesos y actividades para el desarrollo y el mantenimiento del software [IEEE1074-97], además de ofrecer una lista de actividades del ciclo de vida que pueden mapearse hacia procesos y organizarse del mismo modo que cualquiera de los modelos de ciclo de vida del software. Además, identifica y une a estas actividades otros estándares de software IEEE. En línea de principio, el estándar IEEE 1074 puede utilizarse para construir procesos de acuerdo a cualquiera de los modelos de ciclo de vida. Los estándares enfocados al mantenimiento de procesos son el estándar IEEE 1219-1998 y la ISO 14764:1998 [IEEE 1219-98].

Otros estándares importantes que ofrecen definiciones de procesos son:

- Estándar IEEE 1540: Gestión de Riesgos del Software.
- Estándar IEEE 1517: Procesos de Reutilización del Software (IEEE 1517-99).
- ISO/IEC 15939: Proceso de Medición del Software [IEEE 15939-02]. Ver también el KA de Gestión de Ingeniería del Software para una descripción detallada de este proceso.

En algunas ocasiones se han de definir los procesos de ingeniería del software tomando en cuenta los procesos organizacionales para la gestión de la calidad. La ISO 9001 ofrece los requisitos para los procesos de gestión de la calidad y la ISO/IEC 90003 interpreta esos requerimientos para organizaciones que desarrollan software (ISO90003-04).

Algunos procesos del ciclo de vida del software ponen énfasis en entregas rápidas y en una fuerte participación de los usuarios, como por ejemplo métodos ágiles tales como la Programación Extrema [Bec99]. Un tipo de problema de selección tiene que ver con la elección realizable a lo largo del eje del método basado en planificación. Un acercamiento basado en riesgos para tomar tal decisión se describe en (Boe03a).

Notaciones para las Definiciones de los Procesos

Se pueden describir los procesos en diferentes niveles de abstracción (por ejemplo, definiciones genéricas contrapuestas a definiciones adaptadas, descriptivas contrapuestas a prescriptivas contrapuestas a proscriptivas [Pf101].

Varios elementos de un proceso pueden definirse, por ejemplo, actividades, productos (artefactos), y recursos. Los marcos detallados que estructuran los tipos de información requeridos para definir los procesos están descritos en (Mad94).

Existen algunas notaciones que se utilizan para definir procesos (SPC92). Una diferencia clave entre ellas reside en el tipo de información que definen, capturan y utilizan los marcos mencionados anteriormente. El ingeniero del software debería ser consciente de las siguientes aproximaciones al asunto: diagramas de flujo de datos, en términos de la finalidad del proceso y de las salidas [ISO15504-98], como una lista de procesos descompuestos en actividades constituyentes y tareas definidas en lenguaje natural [IEEE12207.0-96], Gráficos de Estados (Har98), EVTX (Rad85), modelado de Dependencia del Actor (Yu94), notación SADT (Mcg93), redes Petri (Ban95); IDEF0 (IEEE 1320.1-98), y los basados en reglas (Bar95). Más recientemente, un estándar de modelado del proceso ha sido publicado por el OMG que tiene como fin armonizar las notaciones de modelado. A esto se le llama la especificación MPIS (Meta-Modelo del Proceso de Ingeniería del Software). [OMG02]

Adaptación del Proceso

[IEEE 12007.0-96; ISO15504-98; Joh99]

Es importante señalar que los procesos predefinidos – incluso los estandarizados– deben adaptarse a las necesidades locales, por ejemplo, el contexto organizacional, el tamaño del proyecto, los requisitos reguladores, las prácticas industriales y las culturas corporativas. Algunos estándares, tales como IEEE/EIA 12207, contienen mecanismos y recomendaciones para lograr la adaptación.

Automatización

[Pf101:c2]

Las herramientas automatizadas o apoyan la ejecución de las definiciones del proceso o aportan una guía a los

humanos que desarrollan los procesos definidos. En los casos en los que se realiza el análisis de un proceso, algunas herramientas permiten distintos tipos de simulaciones (por ejemplo, la simulación de un evento discreto).

Además, existen herramientas que apoyan cada una de las notaciones de la definición del proceso citados anteriormente. Más aún, estas herramientas pueden ejecutar las definiciones de procesos para otorgar una ayuda automatizada a los procesos actuales, o en algunos casos para automatizarlos plenamente. Una visión general de las herramientas de modelado de procesos puede encontrarse en [Fin94] y de los entornos centrados en procesos en (Gar96). Los trabajos sobre cómo aplicar Internet al suministro de una guía de un proceso en tiempo real está descrita en (Kel98).

9. Valoración del Proceso

La valoración del proceso se lleva a cabo utilizando tanto un modelo de valoración como un método de valoración. En algunas instancias, el término "apreciación" se utiliza en vez de valoración, y el término "evaluación de la capabilidad" se utiliza cuando la apreciación tiene como propósito la adjudicación de un contrato.

Modelos de Valoración del Proceso

Un modelo de valoración recoge lo que se reconoce como buenas prácticas. Estas prácticas pueden referirse sólo a las actividades técnicas de ingeniería del software, o puede que se refieran también, por ejemplo, a actividades de gestión, de ingeniería de sistemas, y también de gestión de recursos humanos.

La ISO/IEC 15504 [ISO155'04-98] define un modelo ejemplar de valoración y de requisitos de conformidad con otros modelos de valoración. Los modelos de valoración específicos disponibles y en uso son SW-CMM (SEI95), CMMI [SEI01], y Bootstrap [Sti99]. Se han definido muchos otros modelos de capacidad y madurez –por ejemplo, para diseño, documentación y métodos formales, por nombrar algunos. La ISO 9001 es otro modelo común de validación que ha sido aplicado por organizaciones de software (ISO9001-00).

Se ha desarrollado asimismo un modelo de madurez para sistemas de ingeniería, que puede resultar útil cuando un proyecto u organización esté implicado en el desarrollo y mantenimiento de sistemas, incluido el software (EIA/IS731-99).

En [Joh99; San98] se examina la aplicabilidad de los modelos de valoración a pequeñas organizaciones.

Existen dos arquitecturas generales para un modelo de valoración que ofrecen diversas conjeturas sobre el orden en el que los procesos han de ser valorados: continua y escalonadamente (Pau94). Son muy diferentes entre sí y la organización debería evaluarlos sopesándolos para determinar cuáles son los más pertinentes para sus necesidades y objetivos.

Métodos de Valoración del Proceso
[Go199]

Para poder realizar una valoración, se necesita seguir un método específico de valoración para producir un resultado cuantitativo que caracterizaría la capacidad del proceso (o madurez de la organización).

El método de valoración CBA-IPI, por ejemplo, se centra en la mejora de proceso (Dun96), y el método SCE se centra en evaluar la capacidad de los proveedores (Bar95). Ambos métodos fueron desarrollados para el SW-CMM. En [ISO15504-98], (Mas95) se ofrecen los requisitos de ambos tipos de métodos que reflejan lo que se cree que serían buenas prácticas de valoración. Los métodos SCAMPI giran en torno a las valoraciones CMMI [SEI01]. Las actividades realizadas durante una valoración, la distribución de esfuerzos en estas actividades, así como la atmósfera durante una valoración son muy diferentes dependiendo de que sean para una mejora o para la adjudicación de un contrato.

Se han criticado los modelos y métodos de las valoraciones de los procesos, por ejemplo (Fay97; Gra98). La mayoría de estas críticas se refieren a la evidencia empírica que apoya el uso de modelos y métodos de valoración. Sin embargo, desde la publicación de estos artículos, ha existido alguna evidencia sistemática que apoyaba la eficacia de las valoraciones de los procesos (Cla97; Ele00; Ele00a; Kri99).

10. Medición de los Procesos y Productos

Mientras que la aplicación de mediciones a la ingeniería del software puede resultar compleja, particularmente en términos de métodos de modelado y análisis, existen varios aspectos de las mediciones en la ingeniería del software que resultan fundamentales y que están a la base de muchos de los procesos de medición y análisis más avanzados. Más aún, los esfuerzos para mejorar el logro de procesos y productos sólo pueden valorarse si se ha establecido un conjunto de medidas de base.

La medición puede realizarse para apoyar la iniciación de un procesos de implementación y cambio o para evaluar las consecuencias de un proceso de implementación y cambio, o puede realizarse en el producto mismo.

La medición puede realizarse para apoyar la iniciación de unos procesos de implementación y cambio o para evaluar las consecuencias de un proceso de implementación y cambio, o puede realizarse en el producto mismo.

Los términos clave de medición del software y de métodos de medición han sido definidos en la ISO/IEC 15939 basados en el vocabulario ISO internacional de metrología. La ISO/IEC 15359 también ofrece un proceso estándar para medir tanto los procesos como las características de los productos [VIM93].

A pesar de todo, los lectores encontrarán diferencias terminológicas en la literatura; por ejemplo, el término "métrica" se utiliza a veces en vez de "medida".

4.1 Medición del Proceso [ISO15539-02]

El término "medición del proceso", tal y como se utiliza aquí, significa que se recoge, analiza e interpreta información cuantitativa sobre el proceso. Se utiliza la medición para identificar las fuerzas y las debilidades de los procesos y para evaluar los procesos después de que hayan sido implementados y/o cambiados.

La medición del proceso también puede servir para otros propósitos. Por ejemplo, la medición del proceso es útil para gestionar un proyecto de ingeniería del software. Aquí, el enfoque está en la medición del proceso con el propósito de la implementación y cambio del proceso.

El diagrama de caminos de la Figura 2 ilustra algo que se dar por supuesto en la mayoría de los proyectos de ingeniería del software, que indica que normalmente el proceso tiene un impacto en los resultados de un proyecto.

No todo proceso va a tener un impacto positivo en todas sus salidas. Por ejemplo, la introducción de inspecciones del software puede reducir esfuerzos y costes de las pruebas, pero puede incrementar el tiempo de espera si cada inspección introduce largas esperas a causa de haber calendarizado reuniones de larga inspección (Vot93). Por tanto, es preferible utilizar medidas para salidas de múltiples procesos que resultan importantes para la organización de la empresa.

Mientras que se pueden hacer algunos esfuerzos para valorar la utilización de herramientas y de hardware, el recurso principal que necesita ser gestionado en la ingeniería del software es el personal. Como resultado de esto, las principales mediciones del interés son aquéllas relacionadas con la productividad de los equipos o procesos (por ejemplo, utilizando una medida de puntos función producidos por unidad de persona-esfuerzo) y sus niveles asociados de experiencia en la ingeniería del software en general, y quizás en particulares tecnologías [Fen98: c3, c11; Som05: c25].

Las salidas de los procesos pueden ser, por ejemplo, calidad del producto (errores por KLOC (Kilo Líneas de Código) o por Punto Función (FP)), mantenibilidad (el esfuerzo para hacer un cierto tipo de cambio), productividad (LDC (Líneas de Código)) o Puntos Función por persona-mes), tiempo-de-mercado, o satisfacción del cliente (como medidos por medio de una encuesta a clientes). Esta relación depende del contexto particular (por ejemplo, el tamaño de la organización o el tamaño del proyecto).

En general, estamos mucho más preocupados acerca del proceso de salidas. Sin embargo, con el objetivo de conseguir las salidas del proceso que deseamos (por ejemplo, mayor facilidad de mantenimiento, mayor satisfacción del cliente), debemos haber implementado los procesos adecuados.

Por supuesto que no son únicamente los procesos lo que tiene incidencia en las salidas. Otros factores como la capacidad del equipo y las herramientas que utilizan juegan un importante papel. Por ejemplo, cuando se evalúa el impacto de cambio en un proceso, es importante poner de relieve esas otras influencias. Además es importante considerar el grado en el que el proceso ha sido institucionalizado (que fidelidad hay al proceso) para poder explicar porqué "buenos procesos" no siempre proporcionan las salidas deseadas en una situación dada.

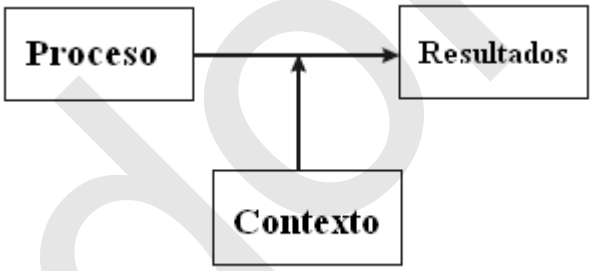


Figura 2 Diagrama que muestra la relación entre un proceso y los resultados obtenidos

Medida del Producto Software [ISO9126-01]

La medición de un producto software incluye, principalmente, la medición del tamaño del producto, la estructura del producto y la calidad del producto.

4.1.1 Medición del Tamaño

El tamaño de un producto software es evaluado a menudo mediante medidas de longitud (por ejemplo, líneas de código fuente en un módulo, páginas en documento de especificación de los requisitos del software), o funcionalidad (por ejemplo, puntos de función en una especificación). El Estándar IEEE Std 14143.1 proporciona los principios de medición funcional del software. Los estándares internacionales para la medición funcional del software incluyen el ISO/IEC 19761, 20926, y el 20968 [IEEE 14143.1-00; ISO19761-03; ISO20926-03; ISO20968-02].

4.1.2 Medición de la Estructura

Un rango diverso de medidas de la estructura de un producto software puede ser aplicado a un nivel bajo y alto de diseño y código del artefacto para así reflejar el control del flujo (por ejemplo el número ciclomático, código de nudo), flujo de la información (por ejemplo, medidas de porción), anidación (por ejemplo, la medida polinomial de anidación, la medida BAND), estructuras de control (por ejemplo, la medida del vector, la medida NPATH), y la estructura e interacción modular (por

ejemplo, el flujo de la información, medidas basadas en árboles, acoplamiento y cohesión).

[Fen98: c8; Pre04: c15]

4.1.3 Medición de la Calidad

Como un atributo multidimensional, la medición de la calidad es menos sencilla de definir que los anteriores. Además, algunas de las dimensiones de la calidad son probables que requieran medidas cualitativas más que cuantitativas. Una discusión más detallada de las medidas de calidad del software es ofrecido en la KA de Calidad del Software, tema 3.4. Los modelos ISO de la calidad del software y las medidas relacionadas son descritos en la ISO 9126, de la parte 1 a la parte 4 [ISO9126-01]. [Fen98: c9,c10; Pre04: c15; Som05: c24]

4.2. Calidad de los Resultados de Medición

Los resultados de la medición de la calidad (precisión, reproducibilidad, repitibilidad, convertibilidad, medición aleatoria de errores) son primordiales para la medida de los programas para proveer resultados efectivos y estables. Las características clave de los resultados de la medición y la calidad relacionada con los instrumentos de medición definidos en vocabulario internacional de metrología [VIM93] de la ISO.

La teoría de la medición establece la base en qué medidas significativas pueden ser creadas. La teoría de la medición y los tipos de escalas son discutidas en [Kan02]. La medición es definida en la teoría como "la asignación de números a los objetos de una forma sistemática para presentar las propiedades de los objetos."

Una apreciación de las escalas para la medición del software y la implicación de cada tipo de escala en relación a la posterior selección de métodos de análisis de información es especialmente importante.

[Abr96; Fen98: c2; Pfl01: c11] Las escalas significativas son mencionadas en una clasificación de escalas. Para aquellas, la teoría de medición ofrece una sucesión de más y más caminos obligatorios de asignación de medidas. Si los números asignados son simplemente para ofrecer etiquetas para clasificar los objetos, entonces son llamados nominales. Si son asignados de tal forma que clasifique los objetos (por ejemplo, bueno, mejor, el mejor), son llamados ordinales. Si son tratados con magnitudes de la propiedad relativa a la unidad de medición definida, son llamados intervalos (y los intervalos son uniformes entre los números; si no, entonces serán aditivos). Las medidas están el nivel del ratio si tienen un punto de cero absoluto, por lo que las distancias de los ratios al punto cero son significativas.

4.3 Modelos de Información del Software

Tal como la información es recogida y el repositorio de medición es completado, nosotros podemos hacer posible la construcción de modelos usando la información y el conocimiento.

Esos modelos existen para analizar, clasificar y predecir. Tales modelos necesitan ser evaluados para asegurar que los niveles de precisión son suficientes y que sus limitaciones son conocidas y entendidas. El refinamiento de los modelos, que tiene lugar durante y después de que los proyectos sean completados, es otra actividad importante.

4.3.1 Creación de Modelos

La creación de modelos incluye la calibración y la evaluación del modelo. El objetivo aproximado al que nos dirigimos son informes de medidas del proceso de construcción de un modelo hasta que los modelos son construidos para así resolver las cuestiones importantes y conseguir las mejoras del software propuestas. Este proceso es está también influenciado por las limitaciones de unas escalas particulares de medición en relación con el método de análisis seleccionado. Los modelos son calibrados (mediante el uso de ciertas observaciones relevantes, como por ejemplo, proyectos recientes, proyectos en los cuales se han utilizado tecnologías similares) y su efectividad es evaluada (por ejemplo, mediante pruebas de su rendimiento en algunas muestras). [Fen98: c4,c6,c13;Pfl01: c3,c11,c12;Som05: c25]

4.3.2 Implementación de Modelos

La implementación de modelos incluye interpretación y refinamiento de modelos- los modelos calibrados son usados en los procesos, sus resultados son interpretados y evaluados en el contexto del proceso/proyecto, y los modelos son luego redefinidos donde sea apropiado. [Fen98: c6; Pfl01: c3,c11, c12; Pre04: c22; Som05: c25]

4.4 Técnicas de Medición del Proceso

Las técnicas de medición deben ser usadas para analizar los procesos de ingeniería del software y para la identificación de las fortalezas y debilidades. Esto puede ser desempeñado para iniciar la implementación y cambio del proceso, o para evaluar las consecuencias de la implementación y el cambio en el proceso.

La calidad de los resultados medidos, como la exactitud, repetitividad, y la reproductibilidad, son asuntos de medición en el proceso de ingeniería del software, ya que están basados en instrumentos y mediciones críticas, como por ejemplo, cuando un asesor asigna una puntuación a un proceso en particular.

Un método y una forma de conseguir calidad en las mediciones se puede encontrar en [Gol99].

Las técnicas de medición de procesos han sido clasificadas dentro de dos tipos generales: analíticas y puntos de referencia. Los dos tipos de técnicas puedes ser usados conjuntamente ya que están basados en distinto tipo de información. (Car91)

4.4.1 Técnicas Analíticas

Las técnicas analíticas se caracterizan por depender de "la evidencia cuantitativa para determinar dónde son necesarias unas mejoras y si una iniciativa de mejora ha tenido éxito". El tipo analítico es ejemplificado por el Paradigma de la Mejora de la Calidad (QIP), que consiste en un círculo de entendimiento, evaluación y empaquetado [SEL96]. Las técnicas presentadas a continuación son pretendidas como otros ejemplos de técnicas analíticas, y reflejan que está hecho en la práctica. [Fen98; Mus99], (Lyu96; Wei93; Zel98) Si una organización específica usa o no todas estas técnicas dependerá, al menos parcialmente, en su madurez.

- *Estudios Experimentales:* la experimentación implica el establecimiento controlado o cuasi experimentos en la organización para evaluar procesos. (McG94). De forma usual, un Nuevo proceso es comparado con el proceso actual para determinar si el primero tuvo mejores resultados o no.

  Otro tipo de estudios experimentales es la simulación del proceso. Este tipo de estudio puede ser usado para analizar el comportamiento del proceso, para explorar las mejoras exponenciales del proceso, predecir los resultados del proceso si el proceso actual es cambiado de cierta manera, controlar la ejecución del proceso. La información inicial sobre el rendimiento del proceso actual necesita ser recogida como base a la simulación.

- *El Informe de Definición del* Proceso es un medio por el cual la definición de un proceso (si es descriptivo, o preceptivo, o ambos) es revisado, e identificadas las deficiencias y las mejoras potenciales del proceso. Ejemplos típicos de esto son presentados en (Ban95; Kel98). Un camino operacional sencillo para analizar el proceso es compararlo con un estándar existente (nacional, internacional, o entidad profesional), como IEEE/EIA 12207.0 [IEEE12207.0-96]. Con esta aproximación, la información cuantitativa no es recogida del proceso, o, si hay, juegan un rol de apoyo. La definición de los análisis de rendimiento individual de los procesos utilizan su conocimiento y capacidades para decidir qué cambios en los procesos deberían conducir a resultados deseables. Estudios observacionales pueden también proveer una útil retroalimentación para identificar las mejoras de los procesos. (Agr99)

- La *Clasificación del Defecto Ortogonal* es una técnica que puede ser usada para enlazar los errores encontrados con causas potenciales. Depende de las vinculaciones entre tipos y disparadores de errores. (Chi92; Chi96) En la clasificación de errores (o anomalías), el Estándar IEEE puede ser útil en este contexto (Estándar IEEE para la Clasificación de Anomalías del Software (IEEE1044-93).

- El *Análisis de Causas desde la Raíz* es otra técnica analítica común que se utiliza en la práctica. Ésta tiene su origen desde los problemas detectados (por ejemplo, errores) para detectar las causas del proceso, con el propósito de cambiar el proceso para evitar estos problemas en el futuro. Se pueden encontrar ejemplos para distintos tipos de procesos en (Col93; Ele97; Nak91).

- La técnica de *Clasificación del Defecto Ortogonal* descrita arriba puede ser usada para encontrar categorías en las que pueden existir muchos problemas, hasta el punto en el que puedan ser analizados. La Clasificación del Defecto Ortogonal es así de este modo usada para hacer una selección cuantitativa para saber dónde aplicar el Análisis de Causas desde la Raíz.

- El *Proceso de Control Estadístico* es un modo eficaz de identificar estabilidad, o la falta de ella, en el proceso a través del uso de trazas de control y sus interpretaciones. Una buena introducción a SPC en el contexto de la ingeniería del software es presentada en (Flo99).

- El *Proceso Personal de Software* define una serie de mejoras a una práctica de desarrollo individualizada en un orden específico [Hum95]. Es un proceso que va desde abajo arriba en el sentido que estipula una colección de información personal y las mejoras basadas en la interpretación de la información.

4.4.2 Técnicas de Bancos de Pruebas

La segunda técnica, bancos de prueba, "depende de la identificar una organización 'excelente' en un campo y en la documentación de sus prácticas y herramientas." El banco de pruebas asume que si una organización menos competente adopta las prácticas de la

organización excelente, ella también llegará a ser excelente. Los bancos de pruebas engloban la evaluación de la madurez de una organización o de la capacidad de sus procesos. Eso se simplifica por el trabajo de evaluación del proceso software. Se puede ver una introducción general de evaluaciones de procesos y su aplicación en (Zah98).

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Abr96]	[Bas92]	[Bec99]	[Boe03]	[Com97]	[Fen98]	[Fin94]	[Fow90]	Gol99]	[Joh99]	[McF96]	[Moi98]	[Mus99]	[OMG02]	[Pfl01]	[Pre04]	[San98]	[SEI01]	[SEL96]	[Som05]	[Sti99]
1.Proceso de Implementación y Cambios																					
1.1Infraestructura del Proceso		*																	*		
1.2 Ciclo de Gestión del Proceso Software		*							*		*								*		
1.3 Modelos para el Proceso de Implementación y Cambios									*		*								*		
1.4 Consideraciones Prácticas											*						*				*
2.Definición de Procesos									*			*									
2.1Modelos de Ciclo de Vida del Software					*																
2.2Procesos del Ciclo de Vida del Software			*	*											c2						
2.3Notaciones para la Definición de Procesos																					
2.4 Adaptación del Proceso														*	c2						
2.5Automatización										*											
3.Valoración del Proceso							*								c2						
3.1Modelos de Valoración de Procesos																	*	*			*
3.2Métodos de Valoración de Procesos										*								*			
4.Medición de Productos y Procesos									*												
4.1Medición de Procesos						c3, c11														c25	
4.2Medición de Productos Software						c8-c10										c15				c24	
4.3Caliddad de los Resultados de Medición	*					c2															
4.4Modelos de Información Software															c11						
Construcción del Modelo						c4, c6,c13														c25	
Implementación del Modelo						c6									c3, c11,c12	c22			*	c25	

4.5 Técnicas de Medición de Procesos						*			*											

	ISO9001	**ISO9000-3**	**ISO9126**	**ISO14764**	**ISO15504**	**ISO15288**	**ISO15939**	**ISO19761**	**ISO20926**	**ISO20969**	**ISO VIM**	**IEEE 1044**	**IEEE 1061**	**IEEE 1074**	**IEEE 1219**	**IEEE 1517**	**IEEE 1540**	**IEEE12207**	**IEEE14143 .1**
1.Proceso de Implementación y Cambios																			
1.1Infraestructura del Proceso					*													*	
1.2 Ciclo de Gestión del Proceso Software					*													*	
1.3 Modelos para el Proceso de Implementación y Cambios																			
1.4 Consideraciones Prácticas																			
2.Definición de Procesos																			
2.1Modelos de Ciclo de Vida del Software																			
2.2Procesos del Ciclo de Vida del Software	*	*		*			*							*	*	*	*	*	
2.3Notaciones para la Definición de Procesos					*														
2.4 Adaptación del Proceso																		*	
2.5Automatización																			
3.Valoración del Proceso																			
3.1Modelos de Valoración de Procesos	*				*														
3.2Métodos de Valoración de Procesos					*														
4.Medición de Productos y Procesos																			
4.1Medición de Procesos							*				*								
4.2Medición de Productos Software			*				*	*	*										*
4.3Caliddad de los Resultados de Medición											*								
4.4Modelos de Información Software																			
Construcción del Modelo																			
Implementación del Modelo																			
4.5 Técnicas de Medición de Procesos												*						*	

REFERENCIAS RECOMENDADAS

[Abr96] A. Abran and P.N. Robillard, "Function Points Analysis: An Empirical Study of its Measurement Processes," *IEEE Transactions on Software Engineering*, vol. 22, 1996, pp. 895-909.

[Bas92] V. Basili et al., "The Software Engineering Laboratory — An Operational Software Experience Factory," presented at the International Conference on Software Engineering, 1992.

[Bec99] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[Boe03] B. Boehm and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *Computer*, June 2003, pp. 57-66.

[Com97] E. Comer, "Alternative Software Life Cycle Models," presented at International Conference on Software Engineering, 1997.

[ElE99] K. El-Emam and N. Madhavji, *Elements of Software Process Assessment and Improvement*, IEEE Computer Society Press, 1999.

[Fen98] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, second ed., International Thomson Computer Press, 1998.

[Fin94] A. Finkelstein, J. Kramer, and B. Nuseibeh, "Software Process Modeling and Technology," *Research Studies Press Ltd.*, 1994.

[Fow90] P. Fowler and S. Rifkin, Software Engineering *Process Group Guide*, Software Engineering Institute, Technical Report CMU/SEI-90-TR-24, 1990, available at http://www.sei.cmu.edu/pub/documents/90.reports/pdf/tr24.90.pdf.

[Gol99] D. Goldenson et al., "Empirical Studies of Software Process Assessment Methods," presented at Elements of Software Process Assessment and Improvement, 1999.

[IEEE1074-97] IEEE Std 1074-1997, *IEEE Standard for Developing Software Life Cycle Processes*, IEEE, 1997.

[IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

[VIM93] ISO VIM, *International Vocabulary of Basic and General Terms in Metrology*, ISO, 1993.
[ISO9126-01] ISO/IEC 9126-1:2001, *Software Engineering - Product Quality-Part 1: Quality Model*, ISO and IEC, 2001.

[ISO15504-98] ISO/IEC TR 15504:1998, *Information Technology - Software Process Assessment (parts 1-9)*, ISO and IEC, 1998. [ISO15939-02] ISO/IEC 15939:2002, *Software Engineering — Software Measurement Process*, ISO and IEC, 2002.

[Joh99] D. Johnson and J. Brodman, "Tailoring the CMM for Small Businesses, Small Organizations, and Small Projects," presented at Elements of Software Process Assessment and Improvement, 1999.

[McF96] B. McFeeley, *IDEAL: A User's Guide for Software Process Improvement*, Software Engineering Institute CMU/SEI-96-HB-001, 1996, available at http://www.sei.cmu.edu/pub/documents/96.reports/pdf/hb001.96.pdf.

[Moi98] D. Moitra, "Managing Change for Software Process Improvement Initiatives: A Practical Experience Based Approach," *Software Process — Improvement and Practice*, vol. 4, iss. 4, 1998, pp. 199-207.

[Mus99] J. Musa, *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*, McGraw-Hill, 1999.

[OMG02] Object Management Group, "Software Process Engineering Metamodel Specification," 2002, available at http://www.omg.org/docs/formal/02-11-14.pdf.

[Pfl01] S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001.

[Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, sixth ed., McGraw-Hill, 2004.

[San98] M. Sanders, "The SPIRE Handbook: Better, Faster, Cheaper Software Development in Small Organisations," *European Commission*, 1998.

[SEI01] Software Engineering Institute, "Capability Maturity Model Integration, v1.1," 2001, available at http://www.sei.cmu.edu/cmmi/cmmi.html.

[SEL96] Software Engineering Laboratory, *Software Process Improvement Guidebook*, NASA/GSFC, Technical Report SEL-95-102, April 1996, available at http://sel.gsfc.nasa.gov/website/documents/online-doc/95-102.pdf.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005.

[Sti99] H. Stienen, "Software Process Assessment and Improvement: 5 Years of Experiences with Bootstrap," *Elements of Software Process Assessment and Improvement*, K. El-Emam and N. Madhavji, eds., IEEE Computer Society Press, 1999.

APÉNDICE A. LISTA DE LECTURAS ADICIONALES

(Agr99) W. Agresti, "The Role of Design and Analysis in Process Improvement," presented at Elements of Software Process Assessment and Improvement, 1999.

(Ale91) L. Alexander and A. Davis, "Criteria for Selecting Software Process Models," presented at COMPSAC '91, 1991.

(Ban95) S. Bandinelli et al., "Modeling and Improving an Industrial Software Process," *IEEE Transactions on Software Engineering*, vol. 21, iss. 5, 1995, pp. 440-454.

(Bar95) N. Barghouti et al., "Two Case Studies in Modeling Real, Corporate Processes," *Software Process — Improvement and Practice*, Pilot Issue, 1995, pp. 17-32.

(Boe03a) B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison-Wesley, 2003.

(Bur99) I. Burnstein et al., "A Testing Maturity Model for Software Test Process Assessment and Improvement," *Software Quality Professional*, vol. 1, iss. 4, 1999, pp. 8-21.

(Chi92) R. Chillarege et al., "Orthogonal Defect Classification - A Concept for In-Process Measurement," *IEEE Transactions on Software Engineering*, vol. 18, iss. 11, 1992, pp. 943-956.

(Chi96) R. Chillarege, "Orthogonal Defect Classification," *Handbook of Software Reliability Engineering*, M. Lyu, ed., IEEE Computer Society Press, 1996.

(Col93) J. Collofello and B. Gosalia, "An Application of Causal Analysis to the Software Production Process," *Software Practice and Experience*, vol. 23, iss. 10, 1993, pp. 1095-1105.

(Cur02) B. Curtis, W. Hefley, and S. Miller, *The People Capability Maturity Model: Guidelines for Improving the Workforce*, Addison-Wesley, 2002.

(Dav88) A. Davis, E. Bersoff, and E. Comer, "A Strategy for Comparing Alternative Software Development Life Cycle Models," *IEEE Transactions on Software Engineering*, vol. 14, iss. 10, 1988, pp. 1453-1461.

(Dun96) D. Dunnaway and S. Masters, "CMM-Based Appraisal for Internal Process Improvement

(CBA IPI): Method Description," Software Engineering Institute CMU/SEI-96-TR-007, 1996, available at http://www.sei.cmu.edu/pub/documents/96.reports/pdf/tr007. 96.pdf.

(EIA/IS731-99) EIA, "EIA/IS 731 Systems Engineering Capability Model," 1999, available at http://www.geia.org/eoc/G47/index.html.

(ElE-97) K. El-Emam, D. Holtje, and N. Madhavji, "Causal Analysis of the Requirements Change Process for a Large System," presented at Proceedings of the International Conference on Software Maintenance, 1997.

(ElE-99a) K. El-Emam, B. Smith, and P. Fusaro, "Success Factors and Barriers in Software Process Improvement: An Empirical Study," *Better Software Practice for Business Benefit: Principles and Experiences*, R. Messnarz and C. Tully, eds., IEEE Computer Society Press, 1999.

(ElE-00a) K. El-Emam and A. Birk, "Validating the ISO/IEC 15504 Measures of Software Development Process Capability," *Journal of Systems and Software*, vol. 51, iss. 2, 2000, pp. 119-149.

(ElE-00b) K. El-Emam and A. Birk, "Validating the ISO/IEC 15504 Measures of Software Requirements Analysis Process Capability," *IEEE Transactions on Software Engineering*, vol. 26, iss. 6, June 2000, pp. 541-566

(Fay97) M. Fayad and M. Laitinen, "Process Assessment: Considered Wasteful," *Communications of the ACM*, vol. 40, iss. 11, November 1997.

(Flo99) W. Florac and A. Carleton, *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison-Wesley, 1999.

(Gar96) P. Garg and M. Jazayeri, "Process-Centered Software Engineering Environments: A Grand Tour," *Software Process*, A. Fuggetta and A. Wolf, eds., John Wiley & Sons, 1996.

(Gra97) R. Grady, *Successful Software Process Improvement*, Prentice Hall, 1997.

(Gra88) E. Gray and W. Smith, "On the Limitations of Software Process Assessment and the Recognition of a Required Re-Orientation for Global Process Improvement," *Software Quality Journal*, vol. 7, 1998, pp. 21-34.

(Har98) D. Harel and M. Politi, *Modeling Reactive Systems with Statecharts: The Statemate Approach*, McGraw-Hill, 1998.

(Her98) J. Herbsleb, "Hard Problems and Hard Science: On the Practical Limits of Experimentation," *IEEE TCSE Software Process Newsletter*, vol. 11, 1998, pp. 18-21, available at http://www.seg.iit.nrc.ca/SPN.

(Hum95) W. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, 1995.

(Hum99) W. Humphrey, *An Introduction to the Team Software Process*, Addison-Wesley, 1999.

(Hut94) D. Hutton, *The Change Agent's Handbook: A Survival Guide for Quality Improvement Champions*, Irwin, 1994.

(Kan02) S.H. Kan, *Metrics and Models in Software Quality Engineering*, second ed., Addison-Wesley, 2002.

(Kel98) M. Kellner et al., "Process Guides: Effective Guidance for Process Participants," presented at the 5th International Conference on the Software Process, 1998.

(Kit98) B. Kitchenham, "Selecting Projects for Technology Evaluation," *IEEE TCSE Software Process Newsletter*, iss. 11, 1998, pp. 3-6, available at http://www.seg.iit.nrc.ca/SPN.

(Kra99) H. Krasner, "The Payoff for Software Process Improvement: What It Is and How to Get It," presented at Elements of Software Process Assessment and Improvement, 1999.

(Kri99) M.S. Krishnan and M. Kellner, "Measuring Process Consistency: Implications for Reducing Software Defects," *IEEE Transactions on Software Engineering*, vol. 25, iss. 6, November/December 1999, pp. 800-815.

(Lyu96) M.R. Lyu, *Handbook of Software Reliability Engineering*, Mc-Graw-Hill/IEEE, 1996.

(Mad94) N. Madhavji et al., "Elicit: A Method for Eliciting Process Models," presented at Proceedings of the Third International Conference on the Software Process, 1994.

(Mas95) S. Masters and C. Bothwell, "CMM Appraisal Framework - Version 1.0," Software Engineering Institute CMU/SEI-TR-95-001, 1995, available at http://www.sei.cmu.edu/pub/documents/95.reports/pdf/tr001.95.pdf.

(McG94) F. McGarry et al., "Software Process Improvement in the NASA Software Engineering Laboratory," Software Engineering Institute CMU/SEI-94-R-22, 1994, available at http://www.sei.cmu.edu/pub/documents/94.reports/pdf/tr22.94.pdf.

(McG01) J. McGarry et al., *Practical Software Measurement: Objective Information for Decision Makers*, ddison-Wesley, 2001.

(Mcg93) C. McGowan and S. Bohner, "Model Based rocess Assessments," presented at International Conference on Software Engineering, 1993.

(Nak91) T. Nakajo and H. Kume, "A Case History Analysis of Software Error Cause-Effect Relationship," *IEEE Transactions on Software Engineering*, vol. 17, iss. 8, 1991.

(Pau94) M. Paulk and M. Konrad, "Measuring Process Capability Versus Organizational Process Maturity," presented at 4th International Conference on Software Quality, 1994.

(Pfl99) S.L. Pfleeger, "Understanding and Improving Technology Transfer in Software Engineering," *Journal of Systems and Software*, vol. 47, 1999, pp. 111-124.
(Pfl01) S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001.

(Rad85) R. Radice et al., "A Programming Process Architecture," *IBM Systems Journal*, vol. 24, iss. 2, 1985, pp. 79-90.

(Rag89) S. Raghavan and D. Chand, "Diffusing Software-Engineering Methods," *IEEE Software*, July 1989, pp. 81-90.

(Rog83) E. Rogers, *Diffusion of Innovations*, Free Press, 1983.

(Sch99) E. Schein, *Process Consultation Revisited: Building the Helping Relationship*, Addison-Wesley, 1999.

(SEI95) Software Engineering Institute, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995.

(SEL96) Software Engineering Laboratory, *Software Process Improvement Guidebook*, Software Engineering Laboratory, NASA/GSFC, Technical Report SEL-95-102, April 1996, available at http://sel.gsfc.nasa.gov/website/documents/online-doc/95-102.pdf

(SPC92) Software Productivity Consortium, *Process Definition and Modeling Guidebook*, Software Productivity Consortium, SPC-92041-CMC, 1992.

(Som97) I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, 1997.

(Vot93) L. Votta, "Does Every Inspection Need a Meeting?" *ACM Software Engineering Notes*, vol. 18, iss. 5, 1993, pp. 107-114.

(Wei93) G.M. Weinberg, "Quality Software Management," *First-Order Measurement (Ch. 8, Measuring Cost and Value)*, vol. 2, 1993.

(Yu94) E. Yu and J. Mylopolous, "Understanding 'Why' in Software Process Modeling, Analysis, and Design," presented at 16th International Conference on Software Engineering, 1994

(Zah98) S. Zahran, *Software Process Improvement: Practical Guidelines for Business Success*, Addison-Wesley, 1998.

(Zel98) M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," *Computer*, vol. 31, iss. 5, 1998, pp. 23-31.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE1044-93) IEEE Std 1044-1993 (R2002), *IEEE Standard for the Classification of Software Anomalies*, IEEE, 1993.

(IEEE1061-98) IEEE Std 1061-1998, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE, 1998.

(IEEE1074-97) IEEE Std 1074-1997, *IEEE Standard for Developing Software Life Cycle Processes*, IEEE, 1997.

(IEEE1219-98) IEEE Std 1219-1998, *IEEE Standard for Software Maintenance*, IEEE, 1998.

(IEEE1220-98) IEEE Std 1220-1998, *IEEE Standard for the Application and Management of the Systems Engineering Process*, IEEE, 1998.

(IEEE1517-99) IEEE Std 1517-1999, *IEEE Standard for Information Technology-Software Life Cycle Processes-Reuse Processes*, IEEE, 1999.

(IEEE1540-01) IEEE Std 1540-2001//ISO/IEC16085:2003,*IEEE Standard for Software Life Cycle Processes-Risk Management*, IEEE, 2001.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/ IEC12207:1995, *Industry Implementation of Int. Std ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

(IEEE12207.1-96) IEEE/EIA 12207.1-1996, *Industry Implementation of Int. Std ISO/IEC 12207:95, Standard forInformation Technology-Software Life Cycle Processes -Life Cycle Data*, IEEE, 1996.

(IEEE12207.2-97) IEEE/EIA 12207.2-1997, *Industry Implementation of Int. Std ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes -Implementation Considerations*, IEEE, 1997.

(IEEE14143.1-00) IEEE Std 14143.1-2000//ISO/ IEC14143-1:1998, *Information Technology-Software Measurement-Functional Size Measurement-Part 1: Definitions of Concepts*, IEEE, 2000.

(ISO9001-00) ISO 9001:2000, *Quality Management Systems-Requirements*, ISO, 1994.

(ISO9126-01) ISO/IEC 9126-1:2001, *Software Engineering-Product Quality-Part 1: Quality Model*, ISO and IEC, 2001.

(ISO14674-99) ISO/IEC 14674:1999, *Information Technology - Software Maintenance*, ISO and IEC, 1999.

(ISO15288-02) ISO/IEC 15288:2002, *Systems Engineering-System Life Cycle Process*, ISO and IEC, 2002.

(ISO15504-98) ISO/IEC TR 15504:1998, *Information Technology - Software Process Assessment (parts 1-9)*, ISO and IEC, 1998.

(ISO15939-02) ISO/IEC 15939:2002, *Software Engineering-Software Measurement Process*, ISO and IEC, 2002.

(ISO19761-03) ISO/IEC 19761:2003, *Software Engineering-Cosmic FPP-A Functional Size Measurement Method*, ISO and IEC, 2003.

(ISO20926-03) ISO/IEC 20926:2003, *Software Engineering-IFPUG 4.1 Unadjusted Functional Size Measurement Method-Counting Practices Manual*, ISO and IEC, 2003.

(ISO20968-02) ISO/IEC 20968:2002, *Software Engineering-MK II Function Point Analysis – Counting Practices Manual*, ISO and IEC, 2002.

(ISO90003-04) ISO/IEC 90003:2004, *Software and Systems Engineering - Guidelines for the Application of ISO9001:2000 to Computer Software*, ISO and IEC, 2004.

(VIM93) ISO VIM, *International Vocabulary of Basic and General Terms in Metrology*, ISO, 1993.

CAPÍTULO 10
INSTRUMENTOS Y MÉTODOS DE LA INGENIERÍA DE SOFTWARE

ACRÓNIMOS

CASE	Computer Assisted Software Engineering

INTRODUCCIÓN

Los instrumentos de desarrollo de software son los instrumentos asistidos por ordenador que son requeridos para ayudar a los procesos de ciclo de vida de software. Los instrumentos permiten a acciones repetidas, bien definidas para ser automatizadas, reduciendo la carga cognoscitiva sobre el ingeniero de software que es entonces libre de concentrarse en los aspectos creativos del proceso. Los instrumentos a menudo son diseñados para apoyar el software particular métodos de la ingeniería, reduciendo cualquier carga administrativa asociada con la aplicación del método a mano. Como los métodos de la ingeniería de software, ellos son queridos para hacer el software que trama más sistemático, varían en el alcance de apoyar tareas individuales que abarcan el ciclo de vida completo.

Los métodos de la ingeniería de software imponen la estructura a la actividad de la ingeniería de software con el objetivo de hacer la actividad sistemática y en última instancia más probablemente de ser acertado. Los métodos por lo general proporcionan la notación y el vocabulario, procedimientos para realizar tareas identificables, y directrices para comprobar tanto el proceso como el producto. Ellos varían extensamente en el alcance, de una fase única del ciclo de vida al ciclo de vida completo. El énfasis en esta Área de Conocimiento está sobre los métodos de la ingeniería de software que abarcan múltiples fases de ciclo de vida, ya que métodos específicos de fase son cubiertos por otras áreas de conocimiento.

Mientras hay manuales detallados sobre instrumentos específicos y numerosos papeles de investigación sobre instrumentos innovadores, escrituras genéricas técnicas sobre instrumentos de la ingeniería de software son relativamente escasas. Una dificultad es la alta tarifa de cambio de instrumentos de software en general. Detalles específicos cambian con regularidad, haciendo difícil de proporcionar ejemplos concretos y actualizados.

Los Instrumentos de Ingeniería de Software y los Métodos del Área de Conocimiento cubren los procesos de ciclo de vida completos, y por lo tanto son relacionados con cada área de conocimiento en la Guía.

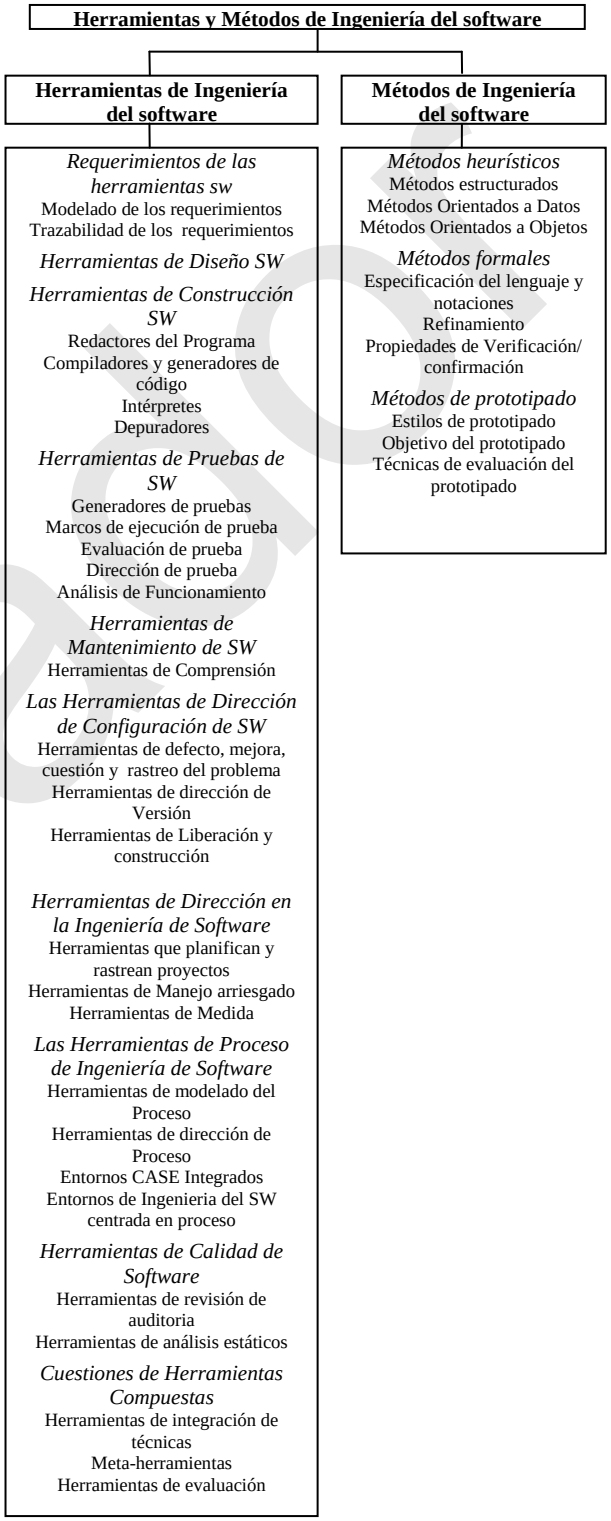


Figura 1 Desglose de tópicos de Instrumentos de Ingeniería del software y los Métodos del Área de Conocimiento.

ESTUDIO DE LAS HERRAMIENTAS Y MÉTODOS DE LA INGENIERÍA DE SOFTWARE

1. Las Herramientas de Ingeniería de Software

Los cinco primeros asuntos del subárea de los Instrumentos de Ingeniería de Software corresponden a las cinco primeras áreas del conocimiento de la Guía (Exigencias de Software, el Diseño de Software, la Construcción de Software, Pruebas de Software, y el Mantenimiento de Software). Los cuatro siguientes asuntos corresponden a las áreas de conocimiento restantes (la Dirección de Configuración de Software, la Dirección de la Ingeniería de Software, el Proceso de Ingeniería de Software, y la Calidad de Software). Proporcionan un asunto adicional, dirigiendo áreas como las técnicas de integración de instrumento que son potencialmente aplicables a todas las clases de instrumentos

1.1 *Las herramientas de Exigencias de Software* [Dor97, Dor02]

Los instrumentos para tratar con exigencias de software han sido clasificados en dos categorías: modelado e instrumentos de capacidad de rastreo.

- Exigencias de los Instrumentos de modelado. Estos instrumentos son usados para la obtención, el análisis, la especificación, y validez de las exigencias de software.
- Exigencias de los Instrumentos de capacidad de rastreo. [Dor02] Estos instrumentos se hacen cada vez más importante debido a que la complejidad de software crece. Ya que ellos son también relevantes en otros procesos de ciclo de vida, son presentados separadamente de los instrumentos de modelado.

1.2. Las herramientas Diseño de Software [Dor02]

Este asunto cubre instrumentos para crear y comprobar diseños de software. Hay una variedad de tales instrumentos, con la mayor parte de esta variedad siendo una consecuencia de la diversidad de notaciones de diseño de software y métodos. A pesar de esta variedad, ninguna división convincente para este asunto ha sido encontrada.

1.3. Las Herramientas de Construcción de Software [Dor02, Rei96]

Este asunto cubre instrumentos de construcción de software. Estos instrumentos son usados para producir y traducir la representación de programa (por ejemplo, el código original) que suficientemente es detallado y explícito para permitir la ejecución de máquina.

- Redactores del Programa. Estos instrumentos son usados para la creación y la modificación de programas, y posiblemente los documentos asociados con ellos. Pueden ser el texto de uso general o redactores de documento, o pueden ser especializado para un idioma de llegada.
- Compiladores y generadores de código. Tradicionalmente, los compiladores han sido los traductores no interactivos de código original, pero hubo una tendencia para integrar compiladores y redactores de programa para proporcionar ambientes de programa integrados. Este asunto también cubre preprocesadores, enlazadores/cargadores, y generadores de código.
- Intérpretes. Estos instrumentos proporcionan la ejecución de software por la emulación. Pueden apoyar actividades de construcción de software proporcionando un ambiente más controlable y observable para la ejecución de programa.
- Depuradores. Estos instrumentos son considerados en una categoría separada ya que ellos apoyan el proceso de construcción de software, pero son diferentes de redactores de programa y recopiladores.

1.4. Herramientas de Pruebas de Software [Dor02, Pfl01, Rei96]

- Generadores de pruebas. Estos instrumentos ayudan en el desarrollo de casos de prueba.
- Marcos de ejecución de prueba. Estos instrumentos permiten la ejecución de casos de prueba en un ambiente controlado donde el comportamiento del objeto bajo prueba es observado.
- Herramientas de evaluación de prueba. Estos instrumentos apoyan la evaluación de los resultados de ejecución de prueba, ayudando a determinar si realmente el comportamiento observado se conforma al comportamiento esperado.
- Herramientas de dirección de prueba. Estos instrumentos proporcionan el apoyo a todos los aspectos del proceso de pruebas de software.
- Herramientas de análisis de Funcionamiento. [Rei96] Estos instrumentos son usado para medir y analizar el funcionamiento de software, que es una forma especializada de pruebas donde el objetivo es de evaluar el comportamiento de funcionamiento más bien que el comportamiento funcional (la corrección).

1.5. Herramientas de Mantenimiento de Software [Dor02, Pfl01]

Este asunto abarca los instrumentos que son en particular importantes en el mantenimiento de software donde el software existente está siendo modificado. Dos categorías son identificadas: instrumentos de comprensión e instrumentos de reingeniería.

- Herramientas de Comprensión. [Re196] Estos instrumentos ayudan en la comprensión humana de programas. Los ejemplos incluyen instrumentos de visualización como rebanadores de programa y animadores.

- Herramientas de reingeniería. En el Mantenimiento de las áreas de conocimiento de Software, reingeniería es definido como el examen y la alteración del software sustancial para reconstituirlo en una nueva forma, e incluye la puesta en práctica subsiguiente de la nueva forma. Los instrumentos de reingeniería apoyan aquella actividad.

Al revés herramientas de la ingeniería ayudan al proceso trabajando hacia atrás de un producto existente a crear artefactos como la especificación y descripciones de diseño, que entonces pueden ser transformadas para generar un nuevo producto de uno anterior.

1.6. Las herramientas de Dirección de Configuración de Software

[Dor02, Rei96, Som05]

Las herramientas para la dirección de configuración han sido divididos en tres categorías: rastreo, dirección de versión, e instrumentos de liberación.

- Defecto, mejora, cuestión, e instrumentos que rastrean problema. Estos instrumentos son usados en la conexión con las cuestiones que rastrean problema asociadas con un producto de software particular.
- Herramientas de dirección de Versión. Estos instrumentos están implicados en la dirección de múltiples versiones de un producto.
- Herramientas de liberación y construcción. Estos instrumentos son usados para las tareas de liberación y construcción de software. La categoría incluye los instrumentos de instalación que se han hecho extensamente usados para configurar la instalación de productos de software.

Más información adicional en Software Configuration Management KA, topic 1.3 *Planning for SCM*.

1.7. Herramientas de Dirección en la Ingeniería de Software

[Dor02]

Herramientas de Dirección en la Ingeniería de Software esta subdividido en tres categorías: planificación de proyecto y rastreo, manejo arriesgado, y medida.

- Herramientas que planifican y rastrean proyectos. Estos instrumentos son usados en la medida de esfuerzo de proyecto de software y cuentan la valoración, así como la planificación de proyecto.
- Herramientas de Manejo arriesgado. Estos instrumentos son usados en la identificación, la estimación, y riesgos de supervisión.
- Herramientas de Medida. Los instrumentos de medida asisten en la realización de las actividades relacionadas con el programa de medida de software.

1.8. Las Herramientas de Proceso de Ingeniería de Software

[Dor02, Som05]

Las herramientas de proceso de ingeniería de Software están divididos en instrumentos que modelan, instrumentos de dirección, y ambientes de desarrollo de software.

- Herramientas de modelado del Proceso. [Pfl01] Estos instrumentos son usados para modelar e investigar los procesos de la ingeniería de software.
- Herramientas de dirección de Proceso. Estos instrumentos proporcionan el apoyo a la dirección de la ingeniería de software.
- Entornos CASE Integrados. [Rei96, Som05] (ECMA55-93, ECMA69-94, IEEE1209-92, IEEE1348-95, Mul96) el software Integrado automatiza instrumentos de la ingeniería o ambientes que cubren múltiples fases del software el ciclo de vida de la ingeniería pertenece a este subtema. Tales instrumentos realizan múltiples funciones y de ahí potencialmente actúan recíprocamente con el proceso de ciclo de vida de software siendo ejecutado.
- Entornos de Ingeniería del SW centrada en proceso. [Rei96] (Gar96) Estos ambientes explícitamente incorporan la información sobre los procesos de ciclo de vida de software y dirigen y supervisan al usuario según el proceso definido.

1.9. Las Herramientas de Calidad de Software

[Dor02]

Las herramientas de Calidad son divididas en dos categorías: inspección e instrumentos de análisis.

- Herramientas de revisión de auditoría. Estos instrumentos son usados para apoyar revisiones y revisiones de cuentas.
- Herramientas de análisis estáticos. [Cla96, Pfl01, Rei96] Estos instrumentos son usados para analizar artefactos de software, como analizadores sintácticos y semánticos, así como datos, el flujo de control, y analizadores de dependencia. Tales instrumentos son queridos para comprobar artefactos de software para la conformidad o para verificar propiedades deseadas.

1.10. Cuestiones de Instrumento Compuestas

[Dor02]

Este asunto cubre el tema aplicable a todas las clases de instrumentos. Tres categorías han sido identificadas: técnicas de integración de instrumento, meta-instrumentos, y evaluación de instrumento.

- Herramientas de integración de técnicas [Pfl01, Rei96, Som01] (Bro94) la integración de Instrumento es importante para hacer a instrumentos individuales cooperar. Esta categoría potencialmente se solapa con la categoría de ambientes de CASO integrada donde las técnicas de integración son

aplicadas; sin embargo, es suficientemente distinto para merecer una categoría de su propiedad. Las clases típicas de integración de instrumento son la plataforma, la presentación, el proceso, datos, y el control.

- Meta-herramientas. Los Meta-instrumentos generan otros instrumentos; recopilador de recopiladores son el ejemplo clásico.
- Herramientas de evaluación. [Pfl01] (IEEE1209-92, IEEE1348-95, Mos92, Val97) A causa de la evolución continua de los instrumentos de la ingeniería de software, la evaluación de instrumento son un tema esencial.

2. Los Métodos de la Ingeniería de Software

Los Métodos de la Ingeniería de Software están dividido en tres temas: métodos heurísticos que tratan con accesos informales, métodos formales que tratan con accesos matemáticamente basados, y métodos de prototipado que tratan con software que trama accesos basados en varias formas de prototipado. Estos tres temas no son inconexos; más bien representan preocupaciones distintas. Por ejemplo, un método orientado por objeto puede incorporar técnicas formales y confiar en prototipado para la verificación y la validación. Como los instrumentos de la Ingeniería de Software, las metodologías continuamente se desarrollan. Por consiguiente, la descripción del área de conocimiento evita en la medida de lo posible llamar metodologías particulares.

2.1. Métodos heurísticos
[Was96]

Este tema contienen cuatro categorías: estructurado, orientado a datos, orientado a objetos, y específico de dominio. La categoría específica de dominio incluye métodos especializados para desarrollar los sistemas que implican en tiempo real, de seguridad, o aspectos de seguridad.

- Métodos Estructurados. [Dor02, Pfl01, Pre04, Som05] el sistema es construido de un punto de vista funcional, que comienza con una vista de alto nivel y cada vez más la refinación de esto en un diseño más detallado.
- Métodos Orientados a datos. [Dor02, Pre04] Aquí, los puntos de partida son las estructuras de datos que un programa manipula más que la función que esto realiza.
- Métodos Orientados a objetos. [Dor02, Pfl01, Pre04, Som05] el sistema es visto como una colección de objetos más que de funciones.

2.2. Métodos Formales
[Dor02, Pre04, Som05]

Esta subdivisión trata con el software matemáticamente basado métodos de la ingeniería, y es subdividida según varios aspectos de métodos formales.

- Especificación del lenguaje y notaciones. [Cla96, Pfl01, Pre01] Este tema concierne la notación de especificación o la lengua usada. Las lenguas de especificación pueden ser clasificadas como orientado por modelo, orientado por característica, u orientado por comportamiento.
- Refinamiento. [Pre04] Este tema trata como el método refina (o transforma) la especificación en una forma que es más cercana a la forma deseada final de un programa ejecutable.
- Propiedades de Verificación/confirmación. [Cla96, Pfl01, Som05] Este tema cubre las propiedades de verificación que son específicas al acercamiento formal, incluyendo tanto confirmación de teorema como la comprobación del modelo.

2.3. Métodos de prototipado
[Pre04, Som05, Was96]

Esta subdivisión cubre métodos que implican el prototipado de software y es subdividida en estilos de prototipado, objetivos, y técnicas de evaluación.

- Estilos de prototipado. [Dor02, Pfl01, Pre04] (Pom96) el tema de estilos de prototipado identifica varios accesos: especificación desechable, evolutiva, y ejecutable.
- Objetivo del prototipado. [Dor97] (Pom96) los Ejemplos de los objetivos de un método prototipado puede ser exigencias, el diseño arquitectónico, o el interfaz de usuario.
- Técnicas de evaluación del prototipado. Este tema cubre las razones por las cuales los resultados de un ejercicio de prototipo son usados.

MATRIZ DE TEMAS VS. REFERENCIAS

	[Cla96]	[Dor02] {Dor97}	[Pfl01] {PFL98}	[Pre04]	[Rei96]	[Som05]	[Was96]
1.Las Herramientas de Ingeniería de Software							
1.1Las Herramientas de Exigencias de Software		{c4s1} ,v2c8s4					
Exigencias de los Herramientas de modelado							
Exigencias de los Herramientas de capacidad de rastreo.		v1c4s2					
1.2 Los Herramientas de Diseño de Software		v2c8s4					
1.3. Los Herramientas de Construcción de Software		v2c8s4			c112s2		
Redactores del Programa							
Compiladores y generadores de código							
Intérpretes.							
Depuradores							
1.4. Herramientas *de Pruebas de Software*		v2c8s4	C8s7,c9s7		c112s3		
Generadores de pruebas							
Marcos de ejecución de prueba							
Herramientas de evaluación de prueba							
Herramientas de dirección de prueba.							
Herramientas de análisis de Funcionamiento					c112s5		
1.5. Herramientas *de Mantenimiento de Software*		v2c8s4	c11s5				
Herramientas de Comprensión					c112s5		
Herramientas de reingeniería							
1.6.Las Herramientas *de Dirección de Configuración de Software*		v2c8s4	c11s5		c112s3	c29	
Herramientas de defecto, mejora, cuestión y rastreo del problema							
Herramientas de dirección de Versión							
Herramientas de Liberación y construcción							

	[Cla96]	[Dor02]{Dor97}	[Pfl01]{PFL98}	[Pre04]	[Rei96]	[Som05]	[Was96]
1.7. Herramientas de Dirección en la Ingeniería de Software		v2c8s4					
Herramientas que planifican y rastrean proyectos							
Herramientas de Manejo arriesgado							
Herramientas de Medida							
1.8. Las Herramientas de Proceso de Ingeniería de Software		v2c8s4					
Herramientas de modelado del Proceso			c2s3, 2s4				
Herramientas de dirección de Proceso							
Entornos CASE Integrados					c112s3, c112s4	c3	
Entornos de Ingeniería del SW centrada en proceso					c112s5		
1.9. Las Herramientas de Calidad de Software		v2c8s4					
Herramientas de revisión de auditoria							
Herramientas de análisis estáticos	*		C8s7		c112s5		
1.10. Cuestiones de Herramientas Compuestas		v2c8s4					
Herramientas de integración de técnicas			c1s8		c112s4		*
Meta-herramientas							
Herramientas de evaluación			C9s10				
2. Los Métodos de la Ingeniería de Software							
2.1. Métodos heurísticos							*
Métodos Estructurados		v1c5s1, v1c6s3	c4s5	c7-c9		c15	
Métodos Orientados a datos		v1c5s1, v1c6s3		c7-c9			
Métodos Orientados a objetos		v1c6s2, v1c6s3	c4s4, c6, c8s5	c7-c9		c12	
2.2. Métodos Formales		v1c6s5		c28		c9	
Especificación del lenguaje y notaciones	*		c4s5				
Refinamiento							
Propiedades de Verificación/ confirmación	*		c5s7, c8s3				
2.3. Métodos de prototipado						c8	*
Estilos de prototipado		v1c4s4	c4s6, c5s6				
Objetivo del prototipado		v1c4s4					
Técnicas de evaluación del prototipado							

REFERENCIAS RECOMENDADAS PARA HERRAMIENTAS Y MÉTODOS DE INGENIERÍA DEL SOFTWARE

[Cla96] E.M. Clarke et al., "Formal Methods: State of the Art and Future Directions," *ACM Computer Surveys*, vol. 28, iss. 4, 1996, pp. 626-643.

[Dor97] M. Christensen, M. Dorfman and R.H. Thayer, eds., *Software Engineering*, IEEE Computer Society Press, 1997.

[Dor02] M. Christensen, M. Dorfman and R.H. Thayer, eds., *Software Engineering*, Vol. 1 & Vol. 2, IEEE Computer Society Press, 2002.

[Pfl01] S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001.

[Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, sixth ed., McGraw-Hill, 2004.

[Rei96] S.P. Reiss, *Software Tools and Environments in The Computer Science and Engineering Handbook*, CRC Press, 1996.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005.

[Was96] A.I. Wasserman, "Toward a Discipline of Software Engineering," *IEEE Software*, vol. 13, iss. 6, November 1996, pp. 23-31.

APÉNDICE A. LISTA DE LECTURAS COMPLEMENTARIAS

(Ber93) E.V. Berard, *Essays on Object-Oriented Software Engineering*, Prentice Hall, 1993.

(Bis92) W. Bischofberger and G. Pomberger, *Prototyping-Oriented Software Development: Concepts and Tools*, Springer-Verlag, 1992.

(Bro94) A.W. Brown et al., *Principles of CASE Tool Integration*, Oxford University Press, 1994.

(Car95) D.J. Carney and A.W. Brown, "On the Necessary Conditions for the Composition of Integrated Software Engineering Environments," presented at Advances in Computers, 1995.

(Col94) D. Coleman et al., *Object-Oriented Development: The Fusion Method*, Prentice Hall, 1994.

(Cra95) D. Craigen, S. Gerhart, and T. Ralston, "Formal Methods Reality Check: Industrial Usage," *IEEE Transactions on Software Engineering*, vol. 21, iss. 2, February 1995, pp. 90-98.

(Fin00) A. Finkelstein, ed., *The Future of Software Engineering*, ACM, 2000.

(Gar96) P.K. Garg and M. Jazayeri, *Process-Centered Software Engineering Environments*, IEEE Computer Society Press, 1996.

(Har00) W. Harrison, H. Ossher, and P. Tarr, "Software Engineering Tools and Environments: A Roadmap," 2000.

(Jar98) S. Jarzabek and R. Huang, "The Case for User-Centered CASE Tools," *Communications of the ACM*, vol. 41, iss. 8, August 1998, pp. 93-99.

(Kit95) B. Kitchenham, L. Pickard, and S.L. Pfleeger, "Case Studies for Method and Tool Evaluation," *IEEE Software*, vol. 12, iss. 4, July 1995, pp. 52-62.

(Lam00) A. v. Lamsweerde, "Formal Specification: A Roadmap," *The Future of Software Engineering*, A. Finkelstein, ed., ACM, 2000, pp. 149-159.

(Mey97) B. Meyer, *Object-Oriented Software Construction*, second ed., Prentice Hall, 1997.

(Moo98) J.W. Moore, *Software Engineering Standards, A User's Roadmap*, IEEE Computer Society Press, 1998.

(Mos92) V. Mosley, "How to Assess Tools Efficiently and Quantitatively," *IEEE Software*, vol. 9, iss. 3, May 1992, pp. 29-32.

(Mül96) H.A. Muller, R.J. Norman, and J. Slonim, eds., "Computer Aided Software Engineering," special issue of *Automated Software Engineering*, vol. 3, iss. 3/4, Kluwer, 1996.

(Mül00) H. Müller et al., "Reverse Engineering: A Roadmap," *The Future of Software Engineering*, A. Finkelstein, ed., ACM, 2000, pp. 49-60.

(Pom96) G. Pomberger and G. Blaschek, *Object-Orientation and Prototyping in Software Engineering*: Prentice Hall, 1996.

(Pos96) R.M. Poston, *Automating Specification-based Software Testing*, IEEE Press, 1996.

(Ric92) C. Rich and R.C. Waters, "Knowledge Intensive Software Engineering Tools," *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, iss. 5, October 1992, pp. 424-430.

(Son92) X. Song and L.J. Osterweil, "Towards Objective, Systematic Design-Method Comparisons," *IEEE Software*, vol. 9, iss. 3, May 1992, pp. 43-53.

(Tuc96) A.B. Tucker, *The Computer Science and Engineering Handbook*, CRC Press, 1996.

(Val97) L.A. Valaer and R.C.B. II, "Choosing a User Interface Development Tool," *IEEE Software*, vol. 14, iss. 4, 1997, pp. 29-39.

(Vin90) W.G. Vincenti, *What Engineers Know and How They Know It — Analytical Studies from Aeronautical History*, John Hopkins University Press, 1990.

(Wie98) R. Wieringa, "A Survey of Structured and Object-Oriented Software Specification Methods and Techniques," *ACM Computing Surveys*, vol. 30, iss. 4, 1998, pp. 459-527.

APÉNDICE B. LISTA DE ESTÁNDARES

(ECMA55-93) ECMA, *TR/55 Reference Model for Frameworks of Software Engineering Environments*, third ed., 1993.
(ECMA69-94) ECMA, *TR/69 Reference Model for Project Support Environments*, 1994.
(IEEE1175.1-02) IEEE Std 1175.1-2002, *IEEE Guide for CASE Tool Interconnections—Classification and Description*, IEEE Press, 2002.
(IEEE1209-92) IEEE Std 1209-1992, *Recommended Practice for the Evaluation and Selection of CASE Tools, (ISO/IEC 14102, 1995)*, IEEE Press, 1992.
(IEEE1348-95) IEEE Std 1348-1995, *Recommended Practice for the Adoption of CASE Tools, (ISO/IEC 14471)*, IEEE Press, 1995.
(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology— Software Life Cycle Processes*, IEEE Press, 1996.

CAPÍTULO 11
CALIDAD DEL SOFTWARE

ACRÓNIMOS

CMMI	Capability Maturity Model Integrated
COTS	Commercial Off-the-Shelf Software
PDCA	Plan, Do, Check, Act
SQA	Software Quality Assurance
SQM	Software Quality Management
TQM	Total Quality Management
V&V	Verificación y Validación

INTRODUCCIÓN

¿Que es la calidad de software, y por qué es tan importante como para estar omnipresente en la Guía SWEBOK? Durante años, los autores y organizaciones han definido el término "calidad" de manera diferente. Para A Phil Crosby (Cro79), fue " la conformidad a las exigencias de usuario. " Watts Humphrey (Hum89) se refiere a calidad como " el alcanzar los niveles excelentes de salud para el empleo" mientras IBM acuñó la frase " la calidad conducida por el mercado, " frase basada en el objetivo de alcanzar la satisfacción de cliente total.

Los criterios Bladridge para la calidad organizacional utilizan una frase similar "calidad conducida por el cliente," e incluye la satisfacción del cliente como una consideración mayor. Más recientemente, la calidad se ha definido en (ISO9001-00) como "el grado en que un conjunto de características inherentes cumple requisitos."

Este capítulo estudia los aspectos relativos a la calidad de software los cuales transcienden a los procesos del ciclo de vida. La calidad de software es un aspecto ubicuo en la ingeniería de software, y por lo tanto también es tratado en mucho de los KAS. En el sumario, la Guía SWEBOK describe un conjunto de modos de alcanzar la calidad del software. En particular, este KA tratará las técnicas estáticas, es decir, aquellas que no requieren la ejecución del software para su evaluación, mientras que las técnicas dinámicas son cubiertas en el KA referido a Pruebas del Software.

DESGLOSE DE LOS TEMAS EN CALIDAD DEL SOFTWARE

1. Fundamentos de Calidad de Software

Un acuerdo sobre exigencias de calidad, así como trasladar a la ingeniería del software qué constituye calidad, requiere que muchos de los aspectos del concepto calidad sean formalmente definidos y tratados.

Un ingeniero de software debería entender los significados subyacentes en los conceptos y características de calidad y su relevancia en el desarrollo o mantenimiento de software.

El concepto relevante es que los requerimientos del software definen las características de calidad requeridas de ese software e influyen en los métodos de medición y criterios de aceptación para evaluar estas características.

1.1. Ingeniería del Software Cultura y Ética.

Los ingenieros de software esperan compartir un compromiso sobre calidad de software como una parte de su cultura. Una cultura sana sobre ingeniería del software se describe en [Wie96].

La ética puede jugar un papel significativo en la calidad de software, la cultura, y las actitudes de ingenieros de software. La IEEE Computer society y el ACM [IEEE99] han desarrollado un código de ética y práctica profesional basada en ocho principios con el objetivo de ayudar a los ingenieros de software a reforzar actitudes relacionadas con la calidad y con la independencia de su trabajo.

1.2. Valor y coste de la calidad.

El concepto de calidad no es tan simple como parece, para un ingeniero de productos hay muchas calidades deseadas relevantes para una perspectiva determinada de un producto, para que esto pueda ser tratado y determinado en el tiempo las exigencias de producto son puestas por escrito. Las características de calidad pueden requerirse o no, o se pueden requerir en un mayor o menor grado, y pueden hacerse compensaciones entre ellas. [Pfl01] El coste de calidad puede segmentarse en el coste de prevención, el coste de apreciación, el coste de fracaso interno, y el coste de fracaso externo. [Hou99]

La motivación latente tras un proyecto de software es el deseo de crear un software que tiene valor, y este valor puede o no puede ser cuantificado como un coste. El cliente tendrá en mente algún coste máximo, a cambio del cual espera que se cumpla el objetivo básico del software. El cliente también puede tener alguna expectativa en cuanto a la calidad del software. En ocasiones los clientes pueden no haber estudiado detenidamente las cuestiones de calidad o sus gastos relacionados. ¿La calidad es meramente decorativa, o es consustancial al software? Si la respuesta se sitúa en un punto intermedio, como es casi siempre el caso, es cuestión de hacer del cliente parte del proceso de decisión y concienciarle totalmente tanto de los costes como de los beneficios. Idealmente, la mayor parte de estas decisiones serán adoptadas en el proceso de requerimientos de software (vd. El KA Requerimientos del software), sin embargo estas cuestiones pueden

surgir en todas las etapas del ciclo de vida del software. No hay ninguna regla definida en cuanto a como deben ser adoptadas estas decisiones, pero el ingeniero de software debería ser capaz de presentar alternativas de calidad y sus correspondientes costes. Una discusión acerca del coste y el valor de los requerimientos de calidad puede encontrarse en [Jon96:c5; Wei96:c11].

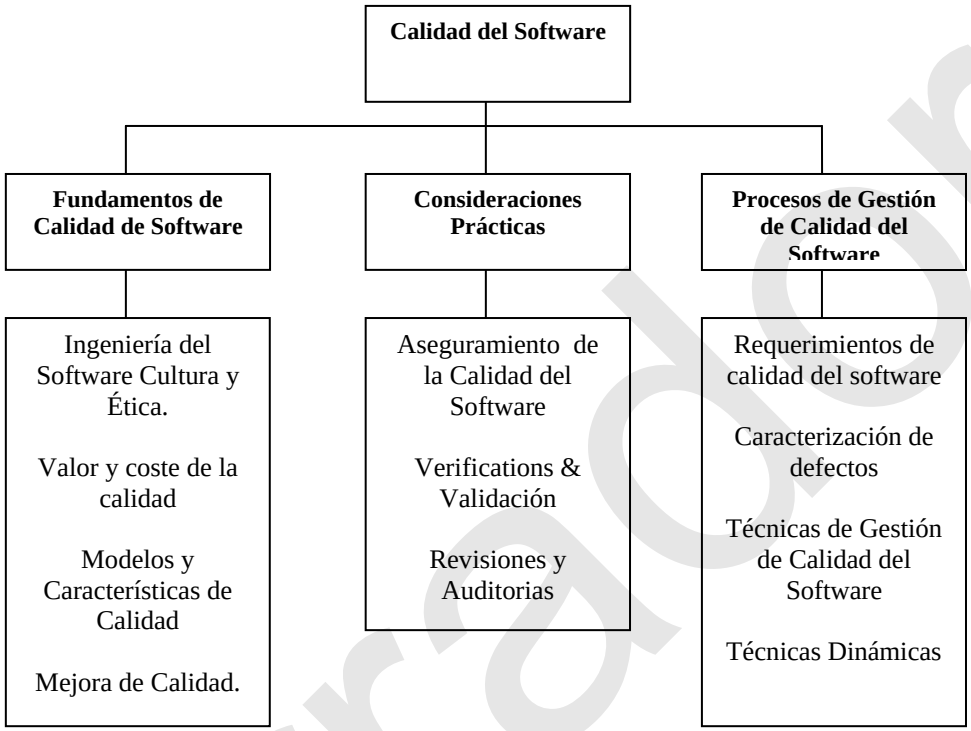


Figura 1 Desglose de los temas de Calidad del Software.

1.3 .Modelos y Características de Calidad.

[Dac01; Kia95; Lap91; Lew92; Mus99; NIST; Pre01; Rak97; Sei02; Wal96].

La terminología para las características de calidad del software difiere de una taxonomía (o modelo de calidad de software) a otra, cada modelo quizás tenga un número diferente de niveles jerárquicos y un número total diferente de características. Varios autores han enunciado distintos modelos de características de calidad de software o atributos que pueden ser útiles para la negociación, planificación, y tasación de la calidad de productos software. [Boe78; McC77] ISO/IEC ha definido tres modelos relacionados de calidad de productos software (la calidad interna, la calidad externa, y la calidad en el empleo) (ISO9126-01) y un conjunto de partes relacionadas (ISO14598-98).

1.3.1. La calidad del proceso en la ingeniería del software.

La gestión de la calidad de software y la calidad de proceso en la ingeniería de software guarda relación directa con la calidad del producto software.

Los modelos y los criterios que evalúan las capacidades organizacionales en software son esencialmente la organización de proyecto y consideraciones de gestión, y, como tales, son tratados en los KAs relativos a Gestión en Ingeniería del Software y el Proceso en Ingeniería de Software.

Desde luego, no es posible distinguir completamente la calidad del proceso de la calidad del producto.

La calidad de proceso, tratada en el KA, de esta Guía, el Proceso en Ingeniería de Software, afecta a las características de calidad de los productos software, que a su vez repercuten en la calidad-en-el-uso tal y como es percibido por el cliente.

Dos importantes estándares de calidad son TickIT [Llo03] y uno con impacto sobre la calidad de software, el estándar ISO9001-00, con sus directrices para su aplicación al software [ISO90003-04].

Otro estándar industrial en calidad del software es el CMMI [SEI02], también tratado en el KA Proceso en la Ingeniería de Software. CMMI pretende proporcionar directrices para mejorar procesos.

Específicamente las áreas de procesos relacionadas con la gestión de calidad son: a) Aseguramiento de la calidad en el proceso y el producto, (b) la verificación de proceso, y c) la validación de proceso. CMMI clasifica revisiones y auditorias como los métodos de verificación, y no como procesos específicos como (IEEE12207.0-96).

Hubo inicialmente algún debate sobre si ISO9001 O CMMI deberían ser usados por ingenieros de software para asegurar la calidad. Este debate ha sido profusamente publicado, y, como resultado, se ha concluido que los dos resultan complementarios y que tener la certificación ISO9001 puede ayudar enormemente para alcanzar los niveles de madurez más altos del CMMI. [Dac01].

1.3.2. Calidad de producto software.

El ingeniero de software, ante todo, necesita determinar el Objetivo verdadero del software. En cuanto a esto, es de capital importancia tener presente los requerimientos del cliente y aquellos que estos incluyen como requerimientos de calidad, no únicamente los requerimientos funcionales. Así, el ingeniero de software tiene como responsabilidad obtener los requerimientos de calidad, que pueden no estar explícitos en un principio, tratar su importancia así como el nivel dificultad para alcanzarlos. Todos los procesos asociados a la Calidad de software (como por ejemplo, construcción, pruebas, mejora de la calidad) serán diseñados con estas exigencias en mente, y ello conlleva gastos adicionales.

El estándar (ISO9126-01) define, para dos de sus tres modelos de calidad, las características de calidad mencionadas, las Sub-características, y las medidas que son útiles para Evaluación de calidad de producto de software. (Sur03)

El significado del término "producto" es ampliado para incluir cualquier artefacto que es la salida de cualquier proceso empleado para construir el producto de software final. Como ejemplos de un producto cabe incluir, aunque no con carácter limitativo, una completa especificación del sistema, una especificación de requerimientos de software para un componente de software de un sistema, un módulo de diseño, código, documentación de prueba, o los informes producidos como consecuencia de tareas de análisis de calidad. Mientras la mayor parte del tratamiento de la calidad es descrito en términos del software final y funcionamiento del sistema, una ingeniería práctica responsable requiere que los productos intermedios relevantes para la calidad sean evaluados a lo largo de todo el proceso de ingeniería de software.

1.4. Mejora de Calidad.

[NIST03; Pre04; Wei96]

La calidad de los productos software puede ser mejorada mediante un proceso iterativo de mejora continua que requiere control de dirección, coordinación, y retroalimentación de muchos procesos simultáneos: (1) los procesos de ciclo de vida de software, (2) El proceso de detección de error/defecto, retirada de los mismo y prevención, (y 3) el proceso de mejora de calidad. (Kin92) La teoría y conceptos presentes detrás de mejora de calidad, tales como la construcción en calidad, mediante la prevención y detección temprana de errores, mejora continua y enfoque en el cliente, son adecuados para la ingeniería de software. Estos conceptos están basados en el trabajo de expertos en calidad los cuales ha afirmado que la calidad de un producto está directamente conectada con la calidad del proceso empleado para crearlo.

Aproximaciones tales como Total Quality Management (TQM) process of Plan, Do, Check, and Act (PDCA) son Instrumentos mediante los cuales conocer los objetivos de calidad. El apoyo a la gestión sustenta el proceso y la evaluación del producto así como las conclusiones resultantes. Entonces se desarrolla un programa de mejora identificando acciones detalladas y proyectos de mejora para ser gestionados en un plazo de tiempo factible. El apoyo a la gestión implica que cada proyecto de mejora tiene suficientes recursos para alcanzar el objetivo definido. El apoyo a la gestión ha ser solicitado con frecuencia mediante la implementación proactiva de actividades de comunicación. La participación de los equipos de trabajo, así como el apoyo a la gerencia media y los recursos asignados en el nivel de proyecto, son tratados en el KA Proceso de Ingeniería de Software.

2. Procesos de Gestión de Calidad del Software

La gestión de calidad de software (SQM) resulta de aplicación a todas las perspectivas de procesos de software, productos, y recursos. Esto define procesos, propietarios de proceso, y requerimientos para aquellos procesos, medidas del Proceso y sus correspondientes salidas, y canales de retroalimentación. (Art93) Los procesos de gestión de calidad del software consisten en numerosas actividades. Algunos de ellos pueden encontrar defectos directamente, mientras otros indican donde pueden resultar valiosas más revisiones. Estos últimos también son conocidos como actividades de "direct-defect-finding". Muchas actividades a menudo sirven para ambos propósitos.

La planificación para la calidad de software implica:

(1) Definición del producto requerido en términos de sus características calidad (descrito más detalladamente en, por ejemplo, El KA Gestión en Ingeniería del Software).
(2) Planificación de los procesos para alcanzar el producto requerido (Descrito, por ejemplo, en los Kas, Diseño de Software y Construcción de Software).

Estos aspectos difieren de, por ejemplo, los procesos mismos de planificación SQM, que evalúan las características de calidad planificadas versus la implementación actual de esa planificación. **Los procesos de gestión de calidad de software deben**

dirigirse a como los buenos productos software van a satisfacer o satisfacen al cliente y las exigencias del personal implicado, a como proporcionan valor a los clientes y demás personal implicado, y proveen la calidad de software precisa para conocer los requerimientos del software.

El SQM puede ser utilizado para evaluar productos intermedios así como el producto final.

Algunos de los procesos específicos SQM están definidos en el estándar (IEEE12207.0-96):

- Procesos de aseguramiento de calidad
- Procesos de verificación
- Procesos de validación
- Procesos de revisión
- Procesos de auditoría

Estos procesos incentivan la calidad y también permiten encontrar posibles problemas. Sin embargo presentan diferencias en cuanto a su énfasis.

Los procesos SQM ayudan a asegurar una calidad de software óptima en un proyecto dado. Además proveen, como un subproducto, información general sobre gestión, incluyendo directrices de calidad para todo el proceso de ingeniería del software. Las Kas Proceso de la Ingeniería del Software y Gestión en Ingeniería del Software tratan sobre la calidad de los programas para la organización que desarrolla el software. El SQM puede proporcionar retroalimentación relevante para estas áreas.

Los procesos SQM consisten en tareas y técnicas para indicar como los proyectos de software (por ejemplo, la gestión, el desarrollo, la gestión de configuración) están siendo puestos en práctica y la mejor manera para que los productos intermedios y los finales encuentren sus requerimientos especificados. Los resultados de estas tareas son recopilados en informes para la dirección antes de que sea tomada la acción correctiva. La gestión de un proceso SQM se desarrolla con la certeza de que los resultados de estos informes son exactos.

Como se describe en este KA, los procesos SQM están estrechamente relacionados; pueden solaparse y hasta, en ocasiones, estar combinados. En su mayor parte parecen de naturaleza reactiva ya que toman los procesos como practicados y los productos como producidos; sin embargo tienen un papel principal en la fase de planificación, con carácter proactivo en términos de los procesos y los procedimientos precisos para alcanzar las características y grados de calidad requerida por los sujetos implicados en el software.

La gestión del riesgo también puede jugar un papel importante en la entrega de software de calidad. La incorporación de un análisis de riesgo disciplinado y técnicas de gestión en los procesos de ciclo de vida de software puede incrementar el potencial para producir un producto de calidad (Cha89). Refiérase al KA Gestión en la Ingeniería del Software para el material relacionado sobre gestión de riesgos.

2.1. Aseguramiento de la Calidad del Software
[Ack02; Ebe94; Fre98; Gra92; Hor03; Pfl01; Pre04; Rak97; Sch99; Som05; Voa99; Wal89; Wal96]

Los procesos SQA proporcionan la garantía de que los productos software y los procesos en el ciclo de vida de proyecto son conformes a los requerimientos especificados por medio de la planificación, emitiendo, y realizando un conjunto de actividades para generar la confianza adecuada en que se está construyendo calidad dentro del software.

Ello significa asegurar que el problema está clara y suficientemente identificado y que los requerimientos de la solución están correctamente definidos y expresados. El SQA procura mantener la calidad a lo largo de todo el desarrollo y mantenimiento del producto mediante la ejecución de una variedad de actividades en cada etapa que pueden permitir identificación temprana de problemas, un rasgo casi inevitable de cualquier actividad compleja. El papel del SQA en lo que concierne al proceso es asegurar que procesos planificados son apropiados y posteriormente implementados de acuerdo con la planificación, y se proveen procesos de medición relevantes para una adecuada organización.

El plan SQA define el medio que será usado para asegurar que el software desarrollado para un producto específico satisface las exigencias del usuario y es de la máxima calidad posible dentro de las restricciones del proyecto. Con el objetivo de llevar esto acabo, primero debe asegurarse que el objetivo de calidad es claramente definido y entendido. En ello deben considerarse los planes de gestión, desarrollo, y mantenimiento para el software. Ver el estándar (IEEE730-98) para detalles.

Las actividades y tareas específicas de calidad se elaboran, con sus gastos y exigencias de recursos, sus objetivos totales de gestión, y su programa en relación con aquellos objetivos de gestión en la ingeniería, el desarrollo, o planes de mantenimiento. El plan SQA debería ser compatible con el plan de gestión de configuración de software (refiérase al KA Gestión de Configuración de Software). El plan SQA identifica documentos, normas, prácticas, y convenciones que guían el proyecto y de qué manera serán comprobados y supervisados para asegurar adecuación y conformidad. El plan SQA también identifica medidas, técnicas estadísticas, procedimientos para el reporte de problemas así como la correspondiente acción correctiva, recursos tales como herramientas, técnicas, y metodologías, seguridad para el medio físico, formación, además de reportes y documentación SQA. Por otro lado, el plan SQA considera las actividades de garantía de calidad de software como cualquier otro tipo de actividad descrita en los proyectos de software, tales como la consecución de proveedor de software para el proyecto o el software de instalación comercial disponible (COTS), así como el servicio tras la entrega del software. También puede incluir criterios de aceptación así como reportes y actividades de gestión críticas para la calidad de software.

2.2. Verificaciones y Validación
[Fre98; Hor03; Pfl01; Pre04; Som05; Wal89; Wal96]

Con el propósito de ser breve, Verificación y Validación (V&V) son tratadas como un único asunto en esta Guía más que como dos asuntos separados tal y como se hace en el estándar (IEEE12207.0-96). La V&V del software es un acercamiento disciplinado a la evaluación de productos de software a lo largo de todo el ciclo de vida de producto. Un esfuerzo en V&V es esforzarse en asegurar que la calidad es construida dentro del software y que el software satisface exigencias de usuario " (IEEE1059-93).

La V&V trata directamente la calidad de producto software y emplea técnicas de prueba que pueden localizar defectos de tal manera que estos puedan ser tratados. También evalúa los productos intermedios, como, y, en esta capacidad, los pasos intermedios de los procesos de ciclo de vida de software.

El proceso V&V determina si productos de una actividad dada de desarrollo o mantenimiento se adecuan o no al correspondiente requisito de esa actividad, y si el producto final de software cumple o no cumple con su propósito fijado y converge o no con los requisitos del usuario. La Verificación es un intento para asegurar que el producto se construya correctamente, en el sentido que los productos resultantes de una actividad converjan con las especificaciones fijadas para los mismos en actividades previas. La validación es un intento por asegurar que se construye el producto correcto, es decir, que el producto satisface su propósito específico fijado. Tanto el proceso de comprobación como el proceso de validación empiezan temprano en la fase de desarrollo o mantenimiento. Proporcionan un examen de características claves del producto en relación con el producto inmediato predecesor y a las especificaciones con las que debe converger.

El propósito de la planificación V&V es asegurar que cada recurso, papel y responsabilidad está claramente asignada. Los documentos del proyecto V&V resultantes describen varios recursos y sus papeles y actividades, así como técnicas y herramientas para ser usados. La comprensión de los objetivos diferentes de cada actividad V&V ayudará en la planificación cuidadosa de las técnicas y los recursos precisos para alcanzar sus respectivos objetivos. Estándares específicos (IEEE1012-98:s7 y IEEE1059-93: El apéndice A) que generalmente incluye un plan de V&V.

El plan también considera la gestión, la comunicación, la política, y los procedimientos de las actividades V&V y su interacción, así como el reporte de defectos y exigencias de documentación.

2.3. Revisiones y Auditorias

Con el propósito de ser breve, revisiones y auditorías son tratadas como un solo tema en esta Guía, más que como dos temas separados tal y como se hace en (IEEE12207.0-96). La revisión y el proceso de auditoría son ampliamente definidos en (IEEE12207.0-96) y más detalladamente en (IEEE1028-97). Cinco tipos de revisiones o auditorias se presentan en el estándar IEEE1028-97:

- Revisiones de gestión
- Revisiones técnicas
- Inspecciones
- Walk-throughs
- Auditorias

2.3.1. Revisiones de gestión

El objetivo de una revisión de gestión es supervisar el progreso, determinando el estado de planes y programas, requerimientos confirmados y su sistema de localización, o evaluar la efectividad de los enfoques de gestión empleados para lograr la idoneidad del objetivo. [IEEE1028-97]. Ello apoya decisiones sobre cambios y las acciones correctivas precisas durante un proyecto de software. Las revisiones de gestión determinan la idoneidad de los proyectos, programas, y requerimientos y supervisan su progreso o inconsistencias. Estas revisiones pueden ser realizadas sobre productos tales como informes de auditoría, informes de progreso, informes V&V, y proyectos de muchos tipos, incluyendo la gestión de riesgo, gestión del proyecto, gestión de configuración del software, seguridad del software, y la evaluación de riesgo, entre otros. Refiérase para el material relacionado a los Kas Gestión en Ingeniería del Software y Gestión de Configuración de Software.

2.3.2. Revisiones técnicas
[Fre98; Hor03; Lew92; Pfl01; Pre04; Som05; Voa99; Wal89; Wal96]

"El propósito de una revisión técnica es evaluar el producto software para determinar si es idóneo para su correspondiente uso. El objetivo es identificar discrepancias con especificaciones aprobadas y estándares. El resultado debería proporcionar gestión con evidencias (o no) de que el producto converge con sus especificaciones y se adhiere a los estándares, y que los cambios están controlados. (IEEE1028- 97).

En una revisión técnica deben estar establecidos los roles específicos: el que adopta las decisiones, un líder revisor, un registrador, y un personal técnico para apoyar las actividades de revisión. Una revisión técnica requiere que las entradas obligatorias estén en su lugar con el objeto de proceder a:

- Exposición de objetivos
- Un producto software específico
- El plan específico de gestión del proyecto
- La lista de cuestiones claves asociadas al producto
- El procedimiento de revisión técnica

El equipo sigue el procedimiento de revisión. Un individuo técnicamente calificado presenta una descripción del producto, y el examen se lleva a cabo durante una o varias reuniones. La revisión técnica es

completada una vez que todas las actividades catalogadas en el examen han sido completadas.

2.3.3. Inspecciones
[Ack02; Fre98; Gil93; Rad02; Rak97]

El propósito de una inspección es detectar e identificar anomalías en los productos software" (IEEE1028-97).Existen dos importantes elementos diferenciadores entre inspección y revisión, son los siguientes:

1. Un individuo que mantiene una posición de dirección sobre cualquier miembro del equipo de inspección no participará en la inspección.

2. La inspección ha de ser llevada por un inspector con formación en inspecciones técnicas

Las inspecciones de software siempre implican al autor de un producto intermedio o final, mientras otras revisiones puede que no. Las inspecciones también incluyen a un líder de inspección, un registrador, un lector, y un grupo (2 a 5) de inspectores. Los miembros de un equipo de inspección pueden poseer diferentes especializaciones, como especialización en el dominio, especialización en el método de diseño, o especialización en el lenguaje. Las inspecciones por lo general son llevadas a cabo sobre una relativamente pequeña sección del producto a la vez. Cada miembro del equipo debe examinar el producto software así como practicar otras revisiones de imputs con anterioridad a la reunión de revisión, quizás aplicando una técnica analítica (referida en la sección 3.3.3) en una pequeña porción del producto, o en todo el producto enfocando solo un aspecto, por ejemplo, interfaces. Cualquier anomalía encontrada se documenta y se envía al responsable de la inspección. Durante la inspección, el responsable de la inspección conduce la sesión y verifica que todos están preparados para la misma.

Un herramienta comúnmente usada en las inspecciones es una lista de comprobación, con anomalías y preguntas pertinentes sobre las cuestiones de interés. La lista resultante a menudo clasifica las anomalías (refiérase a IEEE1044-93 para detalles) y se revisa por parte del equipo su exactitud e integridad. La decisión sobre el final de la inspección se corresponde con uno de los tres criterios siguientes:

1. No aceptar re-trabajo o como mucho un re-trabajo menor
2. Aceptar con verificación del re-trabajo
3. Re-inspección

Típicamente las reuniones de inspección duran por lo general algunas horas mientras que las revisiones técnicas y auditorias son por lo general de mayor alcance y requieren más tiempo.

2.3.4. Walk-throughs
[Fre98; Hor03; Pfl01; Pre04; Som05; Wal89; Wal96]

"El objetivo de un Walk-throughs es evaluar un producto de software. Un Walk-throughs puede ser conducido para el objetivo de formar a una audiencia en cuanto a un producto de software." (IEEE1028-97) los objetivos principales son [IEEE1028-97]:

- Encontrara anomalías
- Mejorar el producto software
- Considerar implementaciones alternativas
- Evaluar la conformidad con estándares y especificaciones

El Walk-throughs es similar a una inspección, sin embargo, su desarrollo, por lo general, es menos formal. El Walk-throughs es organizado fundamentalmente por el ingeniero de software para dar a sus compañeros de equipo la oportunidad de repasar su trabajo, como una técnica de aseguramiento.

2.3.5. Auditorias
[Fre98; Hor03; Pfl01; Pre01; Som05;Voa99; Wal89; Wal96]

"El objetivo de una auditoría de software es proporcionar una evaluación independiente de la conformidad de productos software y procesos a regulaciones aplicables, estándares, directrices, planes, y procedimientos " [IEEE1028-97]. La auditoría es una actividad formalmente organizada, con participantes que tienen papeles específicos, como el auditor jefe, otro auditor, un registrador, o un iniciador, e incluye a un representante de la organización auditada.

La auditoría identificará los casos de no conformidad y producirá un informe requiriendo al equipo que adopte la acción correctiva correspondiente.

Mientras puede haber muchos nombres formales para revisiones y auditorias como los identificados en el estándar (IEEE1028-97), La clave es qué revisiones y auditorias pueden practicarse sobre casi cualquier producto en cualquier etapa del proceso de mantenimiento o el desarrollo.

3. Consideraciones Prácticas

3.1. Requerimientos de calidad del software
[Hor03; Lew92; Rak97; Sch99; Wal89; Wal96]

3.1.1. Factores de influencia

Varios factores influyen en la planificación, gestión, y selección de actividades de SQM y técnicas, incluyendo:

- El dominio del sistema en el cual el software residirá (seguridad crítica, misión crítica, negocio crítico)
- Requerimientos del Sistema y del software
- Los componentes comerciales (externos) o estándar (internos) que serán usados por el sistema

- Los estándares específicos de ingeniería del software específico aplicables
- Los métodos y herramientas de software para ser usados en el desarrollo y el mantenimiento así como para la evaluación de calidad y mejora
- El presupuesto, el personal, planes de organización, proyectos, y la planificación de todos los procesos
- Los usuarios implicados y el empleo del sistema
- El nivel de integridad del sistema

La Información sobre estos factores de influencia como los procesos SQM son organizados y documentados, son seleccionadas como actividades SQM específicas, que recursos son necesarios, y que impondrá límites a los esfuerzos.

3.1.2. Confiabilidad

En casos en los que el fracaso del sistema puede tener consecuencias sumamente severas, la confiabilidad total (en hardware, el software, y humanos) son la exigencia de calidad principal además de la funcionalidad básica. La confiabilidad del software incluye características tales como tolerancia al defecto, fiabilidad, seguridad, y usabilidad. La fiabilidad es también un criterio que puede ser definido en términos de confiabilidad (ISO9126).

El cuerpo de conocimiento para sistemas debe ser sumamente confiable (" alta confianza " " o alta integridad en sistemas"). La terminología para los sistemas tradicionales mecánicos y eléctricos que no incluyen software ha sido importada para tratar amenazas o peligros, riesgos, integridad del sistema, y conceptos relacionados, y puede ser encontrada en las referencias citadas para esta sección.

3.1.3. Niveles de integridad del software

El nivel de integridad se determina en base a las consecuencias posibles del fracaso del software y a la probabilidad de fracaso. Para el software en el que la seguridad o la fiabilidad son importantes, técnicas como el análisis de riesgo para la seguridad o el análisis de amenazas para la fiabilidad pueden ser usadas para desarrollar una actividad de planificación que se identificaría en donde se encuentren conflictos potenciales. Históricos de fracaso de software similar también puede ayudar en la identificación de qué técnicas serán las más útiles en el descubrimiento de defectos y evaluación de calidad. Se proponen niveles de integridad (por ejemplo, la gradación de integridad) en (IEEE1012-98).

3.2. Caracterización de defectos

[Fri95; Hor03; Lew92; Rub94; Wak99; Wal89]

Los procesos SQM encuentran defectos. Caracterizar estos defectos conduce a un entendimiento del producto, facilita correcciones al proceso o al producto, e informa al gestor del proyecto o al cliente del estado del proceso o el producto. Existes muchas taxonomías defectuosas, y, mientras se ha intentado obtener un consenso en una taxonomía de defectos o fallos, la literatura indica que hay bastantes en uso [Bei90, Chi96, Gra92], IEEE1044-93). La caracterización del Defecto (la anomalía) también es usada en auditorias y revisiones, el responsable de revisión a menudo presenta una lista de anomalías proporcionadas por miembros de equipo para su consideración en una reunión de revisión.

Nuevos métodos de diseño y lenguajes se desarrollan, junto con avances en todas las tecnologías de software, las nuevas clases de defectos aparecen, y requieren mucho esfuerzo para interpretar clases previamente definidas. Rastreando defectos, el ingeniero de software está interesado tanto en el número como en el tipo de defectos. La Información sola, sin ninguna clasificación, no es realmente de ninguna utilidad en la identificación de las causas subyacentes de los defectos, pues los tipos específicos de problemas tienen que ser agrupados juntos para determinar como se gestionan. El asunto es establecer una taxonomía de defectos que sea significativa para la organización y los ingenieros de software.

SQM descubre la información en todas las etapas de desarrollo de software y mantenimiento. Típicamente donde se usa la palabra "defecto" se refiere "a un fallo" tal y como se detalla más abajo.

Sin embargo, diferentes culturas y normas pueden usar significados algo diferentes para estos términos, lo cual ha llevado a tentativas para definirlos. Definiciones parciales tomadas del estándar (IEEE610.12-90) son:

- Error: “Una diferencia entre un resultado calculado y un resultado concreto”
- Faultt: “Un paso, proceso, o definición de datos incorrecto en programa de computadora.”
- Failure: “El resultado [incorrecto] de un fault”
- Mistake: “Un error humano que produce un resultado incorrecto” fallo falla

Los fracasos encontrados en pruebas como consecuencia de fallos de software están incluidos como defectos en esta sección.

Los modelos de fiabilidad son construidos en base a los fallos recogidos durante pruebas de software o procedentes de software en servicio, y de esta manera pueden ser usados para predecir futuros fracasos y asistir en decisiones sobre cuando detener las pruebas. [Mus89]

Una probable acción resultante de las conclusiones SQM es eliminar los defectos del producto durante su inspección. Otras acciones permiten alcanzar el valor completo de las conclusiones SQM. Estas acciones incluyen el análisis y el resumen de las conclusiones, y técnicas de medida de utilización para mejorar el producto y el proceso así como para rastrear los defectos y su eliminación. La mejora de proceso principalmente es tratada en el KA Proceso de Ingeniería de Software,

junto con el proceso SQM como una fuente de información.

Los datos sobre las insuficiencias y defectos encontrados durante la implementación de técnicas SQM pueden ser perdidos a no ser que sean registrados. Para algunas técnicas (por ejemplo, revisiones técnicas, auditorias, inspecciones), los registradores están presentes para poner por escrito tal información, incluyendo cuestiones y decisiones. Cuando se utilizan herramientas automatizadas, las salidas de la herramienta pueden proporcionar la información sobre defectos. Los datos sobre defectos pueden ser recogidos y registrados sobre un formulario SCR (software change request) y posteriormente puede ser trasladado a algún tipo de base de datos, a mano o automáticamente, desde una herramienta de análisis. Proporcionan informes sobre defectos a la dirección de la organización.

3.3. Técnicas de Gestión de Calidad del Software

[Bas94; Bei90; Con86; Chi96; Fen97; Fri95; Lev95; Mus89; Pen93; Sch99; Wak99; Wei93; Zel98]

Las técnicas SQM pueden categorizarse de diferentes formas: estáticas,
Personal-intensivas, analíticas, dinámicas.

3.3.1. Técnicas Estáticas

Las técnicas estáticas implican el examen de la documentación del proyecto y el software, y otra información sobre los productos de software, sin ejecutarlos. Estas técnicas pueden incluir actividades intensivas de personal (como se define en 3.3.2) o actividades analíticas (como se define en 3.3.3) conducidas por individuos, con o sin la ayuda de herramientas automatizadas.

3.3.2. Técnicas intensivas de personal.

La puesta en marcha de técnicas intensivas de personal, incluyendo revisiones y auditorias, puede variar de una reunión formal a una reunión informal o una situación de comprobación de escritorio, pero (por lo general, al menos) dos o más personas están implicadas. Puede resultar necesaria una preparación anticipada. Tanto los recursos como los ítems bajo examen pueden incluir listas de comprobación y son resultado de técnicas analíticas y pruebas. Estas actividades son tratadas en (IEEE1028-97) sobre revisiones y auditorias. [Fre98, Hor03] [y Jon96, Rak97]

3.3.3. Técnicas Analíticas

Un ingeniero de software generalmente aplica técnicas analíticas. A veces varios ingenieros de software usan la misma técnica, pero cada uno lo aplica a partes diferentes del producto. Algunas técnicas son llevadas a cabo por herramientas; las otras son manuales. Algunas pueden encontrar defectos directamente, pero generalmente son usadas para apoyar otras técnicas. Algunas técnicas también incluyen varias evaluaciones como la parte de análisis de calidad total. Ejemplos de tales técnicas incluyen el análisis de complejidad, controlan el análisis de flujo, y el análisis algorítmico.

Cada tipo de análisis tiene un objetivo específico, y no se aplican todos ellos a cada proyecto. Un ejemplo de una técnica de apoyo es el análisis de complejidad, que es útil para determinar si realmente el diseño o la implementación resultan demasiado complejos para desarrollar correctamente, realizar las pruebas, o el mantenimiento. Los resultados de un análisis de complejidad también pueden ser usados en test de desarrollo. Las técnicas para encontrar defectos como el análisis de flujo de control, también pueden utilizarse para apoyar otra actividad.

Para software con muchos algoritmos, el análisis algorítmico es importante, especialmente cuando un algoritmo incorrecto podría causar un resultado catastrófico. Hay demasiadas técnicas analíticas para listarlas todas aquí. La lista y referencias proporcionadas pueden ofrecer ideas en la selección de una técnica, así como sugerencias para posteriores lecturas.

Otros tipos, más formales, de técnicas analíticas se conocen como métodos formales. Son usados para verificar requerimiento de software y diseños. Las pruebas de corrección corresponden a las partes críticas de software. Han sido usadas, sobre todo, en la verificación de las partes cruciales de sistemas críticos, tales como la seguridad y exigencias de fiabilidad. (Nas97)

3.3.4. Técnicas Dinámicas

Las diferentes clases de técnicas dinámicas son ejecutadas durante todo el desarrollo y el mantenimiento de software.

Generalmente, son técnicas de testeo, pero técnicas tales como la simulación, la comprobación de modelo, y la ejecución simbólica pueden ser consideradas dinámicas. La lectura de código es considerada una técnica estática, pero ingenieros de software experimentados puede ejecutar el código tal y como lo leen. En este sentido, la lectura de código también puede considerarse una técnica dinámica. Esta discrepancia en la categoría indica que personal con papeles diferentes en la organización puede considerar y aplicar estas técnicas de manera diferente.

Algunas pruebas, así mismo, pueden ejecutarse en el proceso de desarrollo, el proceso SQA, o el proceso de V&V, de nuevo dependen de la organización de proyecto. Debido a que la planificación SQM incluye testeo, esta sección incluye algunos comentarios sobre pruebas.
El KA Pruebas de Software proporciona el tratamiento y referencias técnicas a la teoría, técnicas para pruebas, y automatización.

3.3.5. Pruebas

Los procesos de aseguramiento descritos en SQA y V&V examinan todos los output de la especificación de requerimientos del software con el objeto de asegurar su trazabilidad, consistencia, completitud, corrección y ejecución. Esta comprobación también incluye los output de los procesos de desarrollo y mantenimiento, recopilando, analizando y midiendo los resultados. SQA asegura la planificación, desarrollo e implementación de determinados tipos de pruebas, y V&V desarrolla planes de prueba, estrategias, casos y procedimientos.

Las pruebas son tratadas detalladamente en el KA. testeo de Software. Dos tipos de pruebas pueden incluirse en el ámbito de SQA y V*V, por razón de su responsabilidad por la calidad de los materiales usados en el proyecto:

- Evaluación y prueba de herramientas que serán usadas en el proyecto (IEEE1462-98).
- Prueba de Conformidad (o revisión de prueba de conformidad) de componentes y productos COTS que se usaran en el producto; allí ahora existe un estándar para paquetes de software (IEEE1465-98).

En ocasiones una organización independiente V&V puede ser requerida para monitorear el proceso de pruebas y a veces para atestiguar la ejecución actual con el objeto de asegurar que este es llevado a cabo conforme a los procedimientos especificados. También se puede apelar a V&V para evaluar las pruebas en sí mismas: adecuación a los proyectos y procedimientos, y suficiencia y exactitud de resultados

Otro tipo de pruebas que puede incluirse en el ámbito de la organización V&V es el de pruebas de terceros.

Este tercero no es el desarrollador, tampoco está relacionado en modo alguno con el desarrollo del producto. En cambio, el tercero es un "Facility" independiente, por lo general acreditado por alguna autoridad. Su objetivo es el de probar un producto respecto de su conformidad con un conjunto específico de exigencias.

3.4. Medición de calidad del software

Los modelos de calidad del software a menudo incluyen métricas para determinar el grado de calidad de cada característica alcanzada por el producto.

Si estos modelos se seleccionan correctamente, las métricas pueden apoyar la calidad del software de muchas maneras (entre otros aspectos de los procesos de ciclo de vida de software).

Pueden ayudar en procesos de toma de decisiones de gestión. Pueden encontrar áreas problemáticas y cuellos de botella en procesos de software y pueden ayudar a los ingenieros de software a evaluar la calidad de su trabajo respecto de objetivos SQA y respecto de procesos a largo plazo de mejora de calidad.

Con el incremento de la complejidad del software las cuestiones sobre calidad van más allá de si el software es capaz de alcanzar objetivos de calidad mensurables. Existen algunas áreas en las que las métricas soportan SQM directamente. Esto incluye asistencias en la decisión de cuando detener las pruebas. Para ello los modelos de fiabilidad y pruebas, resultan útiles usando datos de fallos y fracasos.

El coste de los procesos SQM es una cuestión que casi siempre se suscita cuando se adopta la decisión de cómo debería organizarse un proyecto. A menudo, los modelos genéricos de coste usados, están basados en cuando se encuentra y cuanto esfuerzo se precisa para fijar el defecto en relación con el hallazgo del defecto temprano en el proceso de desarrollo. Los datos de proyecto pueden ofrecer una mejor idea del coste. El tratamiento de este tema puede ser encontrada en [Rak97: pp. 39-50]. La información relacionada puede ser encontrada en los KAs el Proceso de Ingeniería de Software y Gestión en Ingeniería del Software.

Finalmente los informes SQM en sí mismos proveen información valiosa no sólo sobre estos procesos, sino también sobre como pueden perfeccionarse todos los procesos de ciclo de vida. El tratamiento de estos asuntos puede encontrarse en [McC04] (y IEEE1012-98).

Mientras las métricas para características de calidad y rasgos de producto pueden resultar útiles en sí mismas (por ejemplo, el número de requerimientos defectuosos o la proporción de requerimientos defectuosos), pueden aplicarse técnicas matemáticas y gráficas para ayudar en la interpretación de esa métricas. Ello se encuentra en las categorías siguientes y son tratados en [Fen97, Jon96, Kan02, Lyu96, Mus99].

- Basadas estadísticamente basado (por ejemplo, Pareto analysis, run charts, scatter plots, normal distribution.
- Pruebas Estadísticas (por ejemplo, binomial test, chisquared test).
- El análisis de Tendencia.
- Predicción (por ejemplo, modelos de fiabilidad).

Las técnicas basadas en estadísticas y las pruebas a menudo proporcionan una imagen de las áreas más problemáticas del producto software examinado. Los cuadros y gráficos resultantes resultan ayudas de visualización que los gestores con poder de decisión pueden utilizar para concentrar recursos donde resulten más necesarios. Los resultados del análisis de tendencia pueden indicar que una programación no ha sido respetada, tales como las pruebas, o que ciertas clases de fallos se intensificarán a no ser que se adopte alguna acción correctiva en el desarrollo. Las técnicas de predicción asisten en la planificación del periodo de prueba y en la predicción del fracaso. Más información sobre medición en general pude encontrarse en los Kas

Proceso de Ingeniería de Software y Gestión en Ingeniería del Software. Información más específica sobre medición en pruebas se presenta en el Ka Testeo de Software.

Las referencias [Fen97, Jon96, Kan02, Pfl01] proporcionan un tratamiento del análisis del defecto, consistente en medir la aparición del defecto y posteriormente aplicar métodos estadísticos para entender los tipos de defectos que aparecen más frecuentemente, es decir, determinar su densidad. También ayudan a comprender las tendencias en cómo están trabajando las técnicas de detección, y cómo están progresando los procesos de desarrollo y mantenimiento. La medición de la cobertura de la prueba ayuda a estimar cuanto esfuerzo en términos de prueba queda por hacer, y predecir los posibles defectos restantes. De estos métodos de medición, pueden desarrollarse los perfiles del defecto para un dominio específico de la aplicación.

Así, para el siguiente sistema de software dentro de esa organización, los perfiles podrán utilizarse para guiar los procesos SQM, es decir, para focalizar esfuerzo allí donde sea más probable que puedan surgir problemas. Semejantemente, los puntos de referencia, o los defectos típicos de ese dominio, pueden servir como ayuda en la determinación de cuándo estará el producto listo para su entrega.

El estudio de como usar datos de SQM para mejorar procesos de desarrollo y mantenimiento puede encontrarse en los Kas Gestión en Ingeniería del Software y Procesos en Ingeniería del Software.

	[Boe78]	[Dac01]	[Hou99]	[IEEE99]	ISO9001-00]	[ISO900 03-04]	[Jon96]	[Kia95]	[Lap91]	[Lew92]	[Llo03]	[McC77]	[Mus99]	[NIST03]	[Pfl01]	[Pre04]	[Rak97]	[Sei02]	[Wal96]	[Wei93]	[Wei96]
1. Fundamentos de Calidad de Software																					
1.1. Ingeniería del Software Cultura y Ética.				*																	*
1.2. Valor y coste de la calidad.	*		*				*							*	*	*				*	*
1.3 .Modelos y Características de Calidad.	*	*			*	*		*	*	*	*	*	*	*		*	*	*	*		
1.4. Mejora de Calidad.														*		*					*

MATRIZ DE TEMAS VS REFERENCIAS

	[Ack02]	[Ebe94]	[Fre98]	[Gil93]	[Gra92]	[Hor03]	[Lew92]	[Pfl01]	[Pre04]	[Rad02]	[Rak97]	[Sch99]	[Som05]	[Voa99]	[Wal89]	[Wal96]
2. Procesos de Gestión de Calidad del Software																
2.1. Aseguramiento de la Calidad del Software	*	*	*		*	*		*	*		*	*	*	*	*	*
2.2. Verificaciones y Validación			*			*		*	*				*		*	*
2.3. Revisiones y Auditorias	*		*	*		*	*	*	*	*	*		*	*	*	*

	[Bas84]	[Bei90]	[Con86]	[Chi96]	[Fen97]	[Fre98]	[Fri95]	[Gra92]	[Hor03]	[Jon96]	[Kan02]	[Lev95]	[Lew92]	[Lyu96]	[McC04]	[Mus89]	[Mus99]	[Pen93]	[Pfl01]	[Rak97]	[Rub94]	[Sch99]	[Wak99]	[Wal89]	[Wal96]	[Wei93]	[Zel98]
3. Consideraciones Prácticas																											
3.1. Requerimientos de calidad del software									*				*							*		*		*	*		
3.2. Caracterización de defectos		*		*			*	*	*				*			*					*		*	*			
3.3. Técnicas de Gestión de Calidad del Software	*	*	*	*	*	*	*		*	*		*				*		*		*		*	*			*	*
3.4. Medición de calidad SW					*			*		*	*			*	*		*		*	*							

REFERENCIAS RECOMENDADAS PARA CALIDAD SW

[Ack02] F.A. Ackerman, "Software Inspections and the Cost Effective Production of Reliable Software," *Software Engineering, Volume 2: The Supporting Processes*, Richard H. Thayer and Mark Christensen, eds., Wiley-IEEE Computer Society Press, 2002.

[Bas84] V.R. Basili and D.M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, vol. SE-10, iss. 6, November 1984, pp. 728-738.

[Bei90] B. Beizer, *Software Testing Techniques*, International Thomson Press, 1990. [Boe78] B.W. Boehm et al., "Characteristics of Software Quality," *TRW Series on Software Technologies*, vol. 1, 1978.

[Chi96] R. Chillarege, "Orthogonal Defect Classification," *Handbook of Software Reliability Engineering*, M. Lyu, ed., IEEE Computer Society Press, 1996.

[Con86] S.D. Conte, H.E. Dunsmore, and V.Y. Shen, *Software Engineering Metrics and Models*: The Benjamin Cummings Publishing Company, 1986.

[Dac01] G. Dache, "IT Companies will gain competitive advantage by integrating CMM with ISO9001," *Quality System Update*, vol. 11, iss. 11, November 2001.

[Ebe94] R.G. Ebenau and S. Strauss, *Software Inspection Process*, McGraw-Hill, 1994.

[Fen98] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, second ed., International Thomson Computer Press, 1998.

[Fre98] D.P. Freedman and G.M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Little, Brown and Company, 1998.

[Fri95] M.A. Friedman and J.M. Voas, *Software Assessment: Reliability, Safety Testability*, John Wiley & Sons, 1995.

[Gil93] T. Gilb and D. Graham, *Software Inspections*, Addison-Wesley, 1993.

[Gra92] R.B. Grady, *Practical Software Metrics for Project Management and Process Management*, Prentice Hall, 1992.

[Hor03] J. W. Horch, *Practical Guide to Software Quality Management*, Artech House Publishers, 2003.

[Hou99] D. Houston, "Software Quality Professional," *ASQC*, vol. 1, iss. 2, 1999.

[IEEE-CS-99] IEEE-CS-1999, "Software Engineering Code of Ethics and Professional Practice," IEEE-CS/ACM, 1999, available at http://www.computer.org/certification/ ethics.htm. [ISO9001-00] ISO 9001:2000, *Quality Management Systems — Requirements*, ISO, 2000.

[ISO90003-04] ISO/IEC 90003:2004, *Software and Systems Engineering-Guidelines for the Application of ISO9001:2000 to Computer Software*, ISO and IEC, 2004.

[Jon96] C. Jones and J. Capers, *Applied Software Measurement: Assuring Productivity and Quality*, second ed., McGraw-Hill, 1996.

[Kan02] S.H. Kan, *Metrics and Models in Software Quality Engineering*, second ed., Addison-Wesley, 2002.

[Kia95] D. Kiang, "Harmonization of International Software Standards on Integrity and Dependability," *Proc. IEEE International Software Engineering Standards Symposium*, IEEE Computer Society Press, 1995.

[Lap91] J.C. Laprie, *Dependability: Basic Concepts and Terminology in English, French, German, Italian and Japanese, IFIP WG 10.4*, Springer-Verlag, 1991.

[Lev95] N.G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

[Lew92] R.O. Lewis, *Independent Verification and Validation: A Life Cycle Engineering Process for Quality Software*, John Wiley & Sons, 1992.

[Llo03] Lloyd's Register, "TickIT Guide," iss. 5, 2003, available at http://www.tickit.org.

[Lyu96] M.R. Lyu, *Handbook of Software Reliability Engineering*: McGraw-Hill/IEEE, 1996.

[Mcc77] J.A. McCall, "Factors in Software Quality — General Electric," n77C1502, June 1977.

[McC04] S. McConnell, *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press, second ed., 2004.

[Mus89] J.D. Musa and A.F. Ackerman, "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, vol. 6, iss. 3, May 1989, pp. 19-27.

[Mus99] J. Musa, *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*: McGraw Hill, 1999.

[NIST03] National Institute of Standards and Technology, "Baldrige National Quality Program," available at http://www.quality.nist.gov.

[Pen93] W.W. Peng and D.R. Wallace, "Software Error Analysis," National Institute of Standards and Technology, Gaithersburg, NIST SP 500-209, December 1993, available at http://hissa.nist.gov/SWERROR/.

[Pfl01] S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001.

[Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, sixth ed., McGraw-Hill, 2004.

[Rad02] R. Radice, *High Quality Low Cost Software Inspections*, Paradoxicon, 2002, p. 479.

[Rak97] S.R. Rakitin, *Software Verification and Validation: A Practitioner's Guide*, Artech House, 1997.

[Rub94] J. Rubin, *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*, John Wiley & Sons, 1994.

[Sch99] G.C. Schulmeyer and J.I. McManus, *Handbook of Software Quality Assurance*, third ed., Prentice Hall, 1999.

[SEI02] "Capability Maturity Model Integration for Software Engineering (CMMI)," CMU/SEI-2002-TR-028, ESC-TR-2002-028, Software Engineering Institute, Carnegie Mellon University, 2002.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005.

[Voa99] J. Voas, "Certifying Software For High Assurance Environments," *IEEE Software*, vol. 16, iss. 4, July-August 1999, pp. 48-54.

[Wak99] S. Wakid, D.R. Kuhn, and D.R. Wallace, "Toward Credible IT Testing and Certification," *IEEE Software*, July/August 1999, pp. 39-47.

[Wal89] D.R. Wallace and R.U. Fujii, "Software Verification and Validation: An Overview," *IEEE Software*, vol. 6, iss. 3, May 1989, pp. 10-17.

[Wal96] D.R. Wallace, L. Ippolito, and B. Cuthill, "Reference Information for the Software Verification and Validation Process," NIST SP 500-234, NIST, April 1996, available at http://hissa.nist.gov/VV234/.

[Wei93] G.M. Weinberg, “Measuring Cost and Value,” *Quality Software Management: First-Order Measurement*, vol. 2, chap. 8, Dorset House, 1993.
[Wie96] K. Wiegers, *Creating a Software Engineering Culture*, Dorset House, 1996.
[Zel98] M.V. Zelkowitz and D.R. Wallace, “Experimental Models for Validating Technology,” *Computer*, vol. 31, iss. 5, 1998, pp. 23-31.

APÉNDICE A. LISTA DE LECTURAS COMPLEMENTARIAS

(Abr96) A. Abran and P.N. Robillard, "Function Points Analysis: An Empirical Study of Its Measurement Processes," presented at IEEE Transactions on Software Engineering, 1996. **//journal or conference?//**

(Art93) L.J. Arthur, *Improving Software Quality: An Insider's Guide to TQM*, John Wiley & Sons, 1993.

(Bev97) N. Bevan, "Quality and Usability: A New Framework," *Achieving Software Product Quality*, E. v. Veenendaal and J. McMullan, eds., Uitgeverij Tutein Nolthenius, 1997.

(Cha89) R.N. Charette, *Software Engineering Risk Analysis and Management*, McGraw-Hill, 1989.

(Cro79) P.B. Crosby, *Quality Is Free*, McGraw-Hill, 1979.

(Dem86) W.E. Deming, *Out of the Crisis*: MIT Press, 1986.

(Dod00) Department of Defense and US Army, "Practical Software and Systems Measurement: A Foundation for Objective Project Management, Version 4.0b," October 2000, available at http://www.psmsc.com.

(Hum89) W. Humphrey, "Managing the Software Process," Chap. 8, 10, 16, Addison-Wesley, 1989.

(Hya96) L.E. Hyatt and L. Rosenberg, "A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality," presented at 8th Annual Software Technology Conference, 1996.

(Inc94) D. Ince, *ISO 9001 and Software Quality Assurance*, McGraw-Hill, 1994.

(Jur89) J.M. Juran, *Juran on Leadership for Quality*, The Free Press, 1989.

(Kin92) M.R. Kindl, "Software Quality and Testing: What DoD Can Learn from Commercial Practices," U.S. Army Institute for Research in Management Information, Communications and Computer Sciences, Georgia Institute of Technology, August 1992.

(NAS97) NASA, "Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Volume II: A Practitioner's Companion," 1997, available at http://eis.jpl.nasa.gov/quality/Formal_Methods/.

(Pal97) J.D. Palmer, "Traceability," *Software Engineering*, M. Dorfman and R. Thayer, eds., 1997, pp. 266-276.

(Ros98) L. Rosenberg, "Applying and Interpreting Object-Oriented Metrics," presented at Software Technology Conference, 1998, available at http://satc.gsfc.nasa.gov/support/index. html.

(Sur03) W. Suryn, A. Abran, and A. April, "ISO/IEC SQuaRE. The Second Generation of Standards for Software Product Quality," presented at IASTED 2003, 2003.

(Vin90) W.G. Vincenti, *What Engineers Know and How They Know It — Analytical Studies from Aeronautical History*, John Hopkins University Press, 1990.

APÉNDICE B. LISTA DE ESTÁNDARES

(FIPS140.1-94) FIPS 140-1, *Security Requirements for Cryptographic Modules*, 1994.

(IEC61508-98) IEC 61508, *Functional Safety — Safety-Related Systems Parts 1, 2, 3*, IEEE, 1998.

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

(IEEE730-02) IEEE Std 730-2002, *IEEE Standard for Software Quality Assurance Plans*, IEEE, 2002.

(IEEE982.1-88) IEEE Std 982.1-1988, *IEEE Standard Dictionary of Measures to Produce Reliable Software*, 1988.

(IEEE1008-87) IEEE Std 1008-1987 (R2003), *IEEE Standard for Software Unit Testing*, IEEE, 1987.

(IEEE1012-98) IEEE Std 1012-1998, *Software Verification and Validation*, IEEE, 1998.

(IEEE1028-97) IEEE Std 1028-1997 (R2002), *IEEE Standard for Software Reviews*, IEEE, 1997.

(IEEE1044-93) IEEE Std 1044-1993 (R2002), *IEEE Standard for the Classification of Software Anomalies*, IEEE, 1993.

(IEEE1059-93) IEEE Std 1059-1993, *IEEE Guide for Software Verification and Validation Plans*, IEEE, 1993.

(IEEE1061-98) IEEE Std 1061-1998, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE, 1998.

(IEEE1228-94) IEEE Std 1228-1994, *Software Safety Plans*, IEEE, 1994.

(IEEE1462-98) IEEE Std 1462-1998//ISO/IEC14102, *Information Technology — Guideline for the Evaluation and Selection of CASE Tools*.

(IEEE1465-98) IEEE Std 1465-1998//ISO/IEC12119:1994, *IEEE Standard Adoption of International Standard IDO/IEC12119:1994(E), Information Technology-Software Packages — Quality Requirements and Testing*, IEEE, 1998.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

(ISO9001-00) ISO 9001:2000, *Quality Management Systems — Requirements*, ISO, 2000.

(ISO9126-01) ISO/IEC 9126-1:2001, *Software Engineering — Product Quality, Part 1: Quality Model*, ISO and IEC, 2001.

(ISO14598-98) ISO/IEC 14598:1998, *Software Product Evaluation*, ISO and IEC, 1998.

(ISO15026-98) ISO/IEC 15026:1998, *Information Technology — System and Software Integrity Levels*, ISO and IEC, 1998.

(ISO15504-98) ISO/IEC TR 15504-1998, *Information Technology — Software Process Assessment (parts 1-9)*, ISO and IEC, 1998.

(ISO15939-00) ISO/IEC 15939:2000, *Information Technology — Software Measurement Process*, ISO and IEC, 2000.

(ISO90003-04) ISO/IEC 90003:2004, *Software and Systems Engineering — Guidelines for the Application of ISO9001:2000 to Computer Software*, ISO and IEC, 2004.

CAPÍTULO 12
DISCIPLINAS RELACIONADAS CON LA INGENIERÍA DEL SOFTWARE

INTRODUCCIÓN

Para circunscribir la Ingeniería del Software, es necesario identificar las disciplinas con las que la Ingeniería del Software comparte un límite común. Este capítulo identifica, en orden alfabético, esas Disciplinas Relacionadas. Por supuesto, las Disciplinas Relacionadas también comparten varios límites en común entre ellas mismas.

Usado como fuente reconocida y consensuada, este capítulo identifica para cada Disciplina Relacionada:

- Una definición informativa (cuando sea factible).
- Un listado de áreas de conocimiento.

La figura 1 da una representación gráfica de estas disciplinas relacionadas.

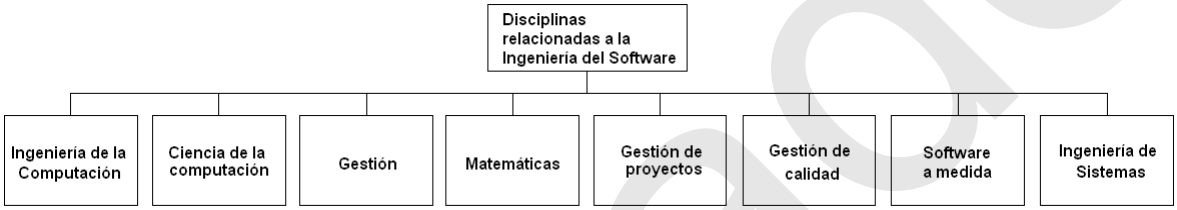


Figura 1 Disciplinas relativas a la Ingeniería del Software

LISTADO DE DISCIPLINAS RELACIONADAS Y SUS ÁREAS DE CONOCIMIENTO

Ingeniería de la computación

El informe borrador del volumen en la Ingeniería de la Computación de *Computing Curricula 2001 project (CC2001)*[1] establece que "La Ingeniería de la Computación incorpora la ciencia y la tecnología del diseño, construcción, implementación y mantenimiento de los componentes software y hardware de los sistemas de cálculo moderno y del equipo controlado por ordenador."

Este informe identifica las siguientes Áreas de Conocimiento (conocidas como áreas en el informe) para la ingeniería de la computación:

- Algoritmos y Complejidad
- Arquitectura de ordenadores y Organización
- Ingeniería de Sistemas Informáticos
- Circuitos y Sistemas
- Lógica Digital
- Estructuras discretas
- Procesamiento de señales digitales
- Sistemas distribuidos
- Electronica
- Sistemas embebidos
- Interacción Hombre-Máquina
- Gestión de la información
- Sistemas inteligentes
- Redes
- Sistemas operativos
- Fundamentos de la programación
- Probabilidad y Estadística
- Problemas sociales y profesionales
- Ingeniería del Software
- Verificación y Prueba
- Diseño VLSI/ASIC

Ciencia de la computación

El informe final del volumen en Ciencia de la computación de *Computing Curricula 2001 project (CC2001)*[2] identifica la siguiente lista de áreas de conocimiento (identificadas como áreas en el informe) para la Ciencia de la computación:

[1] hhtp://www.eng.auburn.edu/ece/CCCE/Iron_Man_Draft_October_2003.pdf

- Estructuras discretas
- Fundamentos de la programación
- Algoritmos y Complejidad
- Arquitectura y Organización
- Sistemas operativos
- Sistemas centralizados
- Lenguajes de programación
- Interacción Hombre-Máquina
- Gráficos y computación visual
- Sistemas inteligentes
- Gestión de la información
- Problemas sociales y profesionales
- Ingeniería del Software
- Ciencia computacional y Cálculo numérico

Gestión

La *European MBA Guidelines* definida por la asociación europea de cuerpos de acreditación nacional (EQUAL)[3] establece que el título *Master en Administración de negocios* debe incluir cobertura e instrucción en

1) Contabilidad
- Finanzas
- Marketing y ventas
- Gestión de operaciones
- Gestión de Sistemas de Información
- Derecho
- Gestión de Recursos Humanos
- Economía
- Analisis cuantitativo
- Política y estrategia de negocio

Matemáticas

Dos fuentes son seleccionadas para identificar la lista de áreas de conocimiento para matemáticas. El informe titulado "Criterios de acreditación y procedimientos"[4] de la *Canadian Engineering Accreditation Board* identifica que los elementos apropiados de las áreas siguientes deben estar presentes en un plan de estudios de la ingeniería del estudiante:

- Algebra lineal
- Cálculo Integral y Diferencial
- Ecuaciones diferenciales
- Probabilidad
- Estadística
- Análisis numérico
- Matemática discreta

Un listado más enfocado en asuntos matemáticos (llamados unidades y asuntos en el informe) que sostiene la ingeniería del software puede ser encontrado en el informe borrador del volumen en Ingeniería del Software en el *Computing Curricula 2001 project* (CC2001)[5].

Gestión de Proyectos

La gestión de proyectos es definida en la edición del 2000 de *A Guide to the Project Management Body of Knowledge* (PMBOK® Guide [6]) publicado por el Project Management Institute y adoptado como IEEE Std 1490-2003, como "el uso del conocimiento, de las habilidades, de las herramientas, y de las técnicas para planificar actividades para satisfacer los requisitos del proyecto." Las áreas de conocimiento identificadas en la guía de PMBOK para la Gestión de Proyectos son

- Gestión de Integración del proyecto
- Gestión de Alcance del proyecto
- Gestión del tiempo del proyecto
- Gestión del coste del proyecto
- Gestión de la calidad del proyecto
- Gestión de los recursos humanos del proyecto
- Gestión de comunicaciones del proyecto
- Gestión de riesgos del proyecto
- Gestión de consecución del proyecto

Gestión de calidad

La gestión de la calidad es definida en el ISO 9000-2000 como "actividades coordinadas para dirigir y controlar una organización con respecto a la calidad." Las tres referencias seleccionadas en gestión de calidad son

- ISO 9000:2000 Sistemas de gestión de calidad – Fundamentos y vocabulario
- ISO 9001:2000 Sistemas de gestión de calidad – Requerimientos
- ISO 9004:2000 Sistemas de gestión de calidad - Pautas para la mejora del funcionamiento

[2] ttp://www.computer.org/education/cc2001/final/cc2001.pdf
[3] http://www.efmd.be/
[4] http://www.ccpe.ca/e/files/report_ceab.pdf
5 http://sites.computer.org/ccse/volume/FirstDraft.pdf
6 PMBOK es una marca registrada de los Estados Unidos y otras naciones.
La Sociedad Americana para la Calidad identifica las siguientes áreas de conocimiento en su Cuerpo de Conocimiento para la certificación como Ingeniero de Calidad[7]:

2) Gestión y liderazgo en ingeniería de calidad

- Sistemas de calidad de desarrollo, implementación y verificación
- Planificación, control y fiabilidad en la calidad del producto y del proceso
- Gestión de la fiabilidad y el riesgo
- Solución de problemas y mejora de la calidad
- Métodos cuantitativos

Ergonomía del software

El campo de la ergonomía está definido por la ISO Technical Committee 159 sobre la ergonomía y dice: Ergonomía o (factores humanos) es la disciplina científica referida a comprensión de las interacciones entre un ser humano y otros elementos de un sistema, y la profesión que conlleva teoría, principios, datos y métodos para diseñar con objeto de optimizar el bienestar humano y el funcionamiento del sistema global.[8"]

Una lista de las áreas del conocimiento para la ergonomía n relación con el software se detalla a continuación[9]:

- Cognición
- I.A. cognoscitiva I: Razonamiento
- Aprendizaje de las máquinas e inducción de la gramática
- Métodos formales en ciencia cognoscitiva: Lenguaje
- Ellos debían indicar cuáles de estos tópicos debían ser conocidos como Ingeniería del Software. Los que fueron rechazados por dos de los tres integrantes fue eliminados de la lista original.
- Métodos formales en ciencia cognoscitiva: Razonamiento
- Métodos formales en ciencia cognoscitiva:
- Arquitectura cognoscitiva
- I.A. cognoscitiva II: Aprendizaje

7.http://isotc.iso.ch/livelink/livelink.exe/fetch/2000/2122/687806/ISO_TC_159__Ergonomics_.pdf?nodeid=1162319&vernum=0http://www.asq.org/cert/types/cqe/bok.html

8.http://isotc.iso.ch/livelink/livelink.exe/fetch/2000/2122/687806/ISO_TC_159_Ergonomics_.pdf?nodeid=1162319&vernum=0

9 Esta lista fue compilada para la edición del 2001 del SWEBOK Guide para la lista de cursos ofrecidos por el Departamento de Ciencia Cognitiva de la Universidad de John Hopkins University y el ACM SIGCHI Curricula for Human-Computer.La lista fue entonces refinada por tres expertos en el campo: dos de la Universidad de Québec en Montréal y por W. W. McMillan, de la Eastern Michigan University.

- Fundaciones de la ciencia cognoscitiva
- Extracción de la información del habla y del texto
- Procesado léxico
- Adquisición de cómputo de la lengua
- Naturaleza del HCI
  - (Meta-) Modelos de HCI
- Uso y contexto de computadoras
  - Organización y trabajo sociales humanos
  - Áreas de aplicación
- Ajuste y adaptación Hombre-Máquina
- Características humanas
  - Tratamiento de la información humana
  - Lengua, comunicación, interacción
  - Ergonómica
- Sistema informático y arquitectura del interfaz
  - Dispositivos de la entrada y de salida
  - Técnicas del diálogo
  - Género del diálogo
  - Gráficos de computadora
- Arquitectura del diálogo
- Proceso del desarrollo
  - Borradores del diseño
  - Técnicas de implementación
  - Técnicas de la evaluación
  - Sistemas de ejemplo y casos de estudio

Se puede encontrar una lista más detallada de asuntos de interfaz hombre-computadora diseñada (llamado unidades y asuntos en el informe) para la ingeniería del software, en el borrador del informe del volumen en la ingeniería del software Computing Curricula 2001 project (CC2001)[10].

Ingeniería de sistemas

El consejo internacional sobre la ingeniería de sistemas (INCOSE)[11] dice que "la ingeniería de sistemas es interdisciplinar y capaz de activar la realización de sistemas eficientes. Se centra en definir los requisitos de usuario y funcionalidad requerida en fases iniciales del ciclo de desarrollo, documentando requisitos, luego procediendo con la síntesis del diseño y la validación del sistema mientras se considera el problema globalmente: operaciones de funcionamiento, pruebas, desarrollo, coste y planificación, entrenamiento, ayuda y disposición."

La ingeniería de sistemas integra todas las disciplinas y grupos de especialidades en un esfuerzo del equipo que forma procesos estructurados de desarrollo a el cual procede de concepto, a producción y a operación. La

10 http://sites.computer.org/ccse/volume/FirstDraft.pdf
11 www.incose.org

ingeniería de sistemas considera ambos el negocio y las necesidades técnicas de todos los clientes con el objetivo de proporcionar un producto de calidad que resuelve las necesidades del usuario.

El consejo internacional sobre la ingeniería de sistemas (INCOSE, www.incose.org) está trabajando en una guía del Cuerpo de Conocimiento de la ingeniería de sistemas. Versiones iniciales incluyen las siguientes áreas de competencias de primer nivel:

1. Procesos de negocio y gravamen operacional (BPOA)
2. Sistema/Solución/Prueba de la arquitectura (SSTA)
3. Coste del ciclo vital y análisis de costes y beneficios (LCC y CBA)
4. Utilidad/logística (S/L)
5. Modelado, simulación, y análisis (MS&A)
6. Gestión: Riesgo, configuración, línea de fondo (Mgt)

APÉNDICE A
ESPECIFICACIONES DE LA DESCRIPCIÓN DE ÁREAS DE CONOCIMIENTO PARA LA GUÍA HOMBRE DE HIERRO DE LA GUÍA DEL CUERPO DE CONOCIMIENTO DE LA INGENIERÍA DEL SOFTWARE

INTRODUCCIÓN

Este documento presenta la versión 1.9 de las especificaciones proporcionadas al equipo editorial a los expertos de las AC en relación a las Descripciones de las Áreas de Conocimiento de la Guía del Cuerpo de Conocimiento de la Ingeniería del software (Versión Hombre de Hierro).

Este documento comienza presentando especificaciones sobre los contenidos de las áreas de las Descripciones de las Áreas de Conocimiento. Los criterios y requerimientos son definidos: para las descripciones de los temas propuestos, las bases para dichas descomposiciones y una breve descripción de los temas, para la selección de las referencias, y para la identificación de las Áreas de Conocimiento de las Disciplinas Relacionadas. Se explican documentos y sus roles de dentro de los proyectos software. Además se tratan asuntos no relacionados con el contenido como formatos de entrega y guías de estilo.

CRITERIOS Y REQUERIMIENTOS PARA LA DESCOMPOSICIÓN DE TÓPICOS INTERRUPCIÓN DE LOS ASUNTOS PARA LOS REQUISITOS DEL SOFTWARE

Los siguientes requerimientos y criterios que deberían utilizarse para proponer una descomposición de temas en un Área de Conocimiento:

a) Se espera que los editores asociados propongan una o posiblemente dos, descomposiciones complementarias a sus especificas Áreas de Conocimiento. Los temas encontrados en todas las descomposiciones dada un Área de Conocimiento deben ser iguales.

b) Se espera que la descomposición de temas sea "razonable", no "perfecta". La Guía al Cuerpo de Conocimiento de la Ingeniería del Software es un esfuerzo a realizarse en fases, con muchas interacciones dentro de cada fase y múltiples fases que serán necesarias para mejorar las descomposiciones.

c) Las descomposiciones de los temas propuestas dentro de un Área de Conocimiento debe descomponer el subconjunto del Cuerpo de Conocimiento que es "globalmente aceptado". Este punto se detalla más adelante.

d) La descomposición de temas propuesta dentro de un Área de Conocimiento no debe asumir dominios de aplicaciones específicos, necesidades de negocio, tamaños de organizaciones, filosofías de gestión, ciclos de vida del software, tecnologías software o métodos de desarrollo del software.

e) La descomposición de temas propuesta debe, en todo lo posible, ser compatible con las varias escuelas de pensamiento dentro de la ingeniería del software.

f) La descomposición de temas propuesta dentro de las Áreas de Conocimiento, debe ser compatible con la descomposición de la ingeniería del software encontrada en la industria y en los estándares y literatura de la ingeniería del software.

g) La descomposición de temas propuesta debe ser tan extensa como sea posible. Es mejor sugerir muchos temas y que luego algunos sean abandonados que lo contrario.

h) Se espera de los editores asociados del Área de Conocimiento, que se posicionen los temas de calidad (en general) y medición, comunes y parte integral de todas las Áreas de Conocimiento.

   Nótese que como tratar temas comunes u ortogonales, y si estos deberían de tratarse de otra manera no ha sido completamente resulto todavía.

i) La descomposición propuesta debería de tener a lo sumo tener dos o tres niveles de profundidad. Aunque no se ha puesto ningún límite ni superior ni inferior al número de temas dentro de cada Área de Conocimiento, se espera que los Editores Asociados de cada Área de Conocimiento propongan un número razonable y manejable de temas por Áreas de Conocimiento. Además el énfasis debe de ponerse en la selección de temas en lugar de su organización en una jerarquía apropiada.

Los temas propuestos deben ser lo suficientemente significantes incluso cuando se citan fuera de la Guía al Cuerpo de Conocimiento del Software.

j) La descripción de un Área de Conocimiento incluirá una figura (en forma de árbol) describiendo la descomposición del conocimiento.

CRITERIOS Y REQUERIMIENTOS PARA LA DESCRIPCIÓN DE TEMAS

a) Los temas necesitan solo ser descritos hasta un nivel apropiado para que el lector pueda seleccionar el material referenciado acorde a sus necesidades.

CRITERIOS Y REQUERIMIENTOS PARA LA SELECCIÓN DE MATERIAL DE REFERENCIA

a) Debe identificarse material de referencia específico a cada tema. Los materiales de referencia pueden cubrir múltiples temas.

b) El material de referencia propuesto pueden ser capítulos de libro, artículos de revista revisados, artículos de referencia revisados, reportes técnicos o de empresa revisados, o cualquier otro tipo de artefactos como documentos Web. Deben de estar disponibles y no ser confidenciales por naturaleza. Las referencias deberían de ser lo más precisas posibles especificando los capítulos o secciones relevantes.

c) El material de referencia propuesto debe de estar en Ingles.

d) Para cada Área de Conocimiento debe seleccionarse una un número de referencias apropiado. Las siguientes guías deben usarse para determinar el numero apropiado:

- Si el material referenciado estuviese escrito de manera coherente siguiendo la descomposición propuesta y en un estilo uniforme (por ejemplo, en un nuevo libro basado en la descripción de las Áreas de Conocimiento), el objetivo debería de ser unas 500 páginas. Sin embargo, este objetivo puede no ser alcanzable en la selección de referencias debido a diferencias en estilo, sobreposición y redundancia entre los materiales seleccionados.
- La cantidad de material de referencia de ser razonable si constituyese el material de estudio para el Área de Conocimiento de un examen de licencia de ingeniería del software que un graduado debería de pasar tras cuatro años de experiencia laboral.
- La Guía al cuerpo de Conocimiento de Ingeniería del Software busca por definición ser selectivo en su selección de temas y material de referencia asociado. La lista de material de referencia para cada Área de Conocimiento debería de ser visto y será presentado como una "selección razonable y bien fundada" y no como una lista definitiva.
- Material de referencia adicional puede ser incluido en la lista de "Lecturas Complementarias". Estas lecturas complementarias deben relacionarse con los temas en la descomposición. Además deben comentarse en el conocimiento globalmente aceptado. No debería de haber una matriz entre el material de referencia contenido en las Lecturas Complementarias y los temas individuales.

e) Si se considera posible y rentable por el IEEE Computer Society, el material de referencia seleccionado será publicado en la Web de la Guía al Cuerpo de Conocimiento de la Ingeniería del Software. Para facilitar esta tarea, se debería referenciar materiales cuyo copyright ya pertenecen al IEEE Computer Society. Sin embargo, esto no debería considerarse como una restricción o una obligación.

f) Debe proporcionarse una matriz con los materiales de referencia versus temas

CRITERIOS Y REQUERIMIENTOS PARA LA IDENTIFICACIÓN DE LAS ÁREAS DE CONOCIMIENTO DE LAS DISCIPLINAS RELACIONADAS

Se espera de los editores asociados de las Áreas de Conocimiento identifiquen en una sección aparte, que Áreas de Conocimiento de las Disciplinas Relacionadas son suficientemente relevantes a las Áreas de Conocimiento de la Ingeniería del Software para graduados con cuatro años de experiencia.

Esta información será particularmente útil y generara dialogo entre la iniciativa de la Guía al cuerpo de Conocimiento de la Ingeniería del Software e iniciativas hermanas responsables de definir el currículum de la ingeniería del software y normas de comportamiento estándar.

La lista de Áreas de Conocimiento de Disciplinas Relacionadas se puede encontrar en la Lista Base de Disciplinas Relacionadas propuestas. Si se considera

necesario y va acompañado de una justificación, los especialistas de las Áreas de Conocimiento pueden proponer Disciplinas Relacionadas adicionales que no han sido incluidas en la Lista Base de Disciplinas Relacionadas (nótese que una clasificación de los temas de las Disciplinas Relacionadas ha sido producido pero será publicado en la Web más adelante como documento en progreso. Contáctese con el equipo editorial para más información).

ÍNDICE DE CONTENIDOS COMUNES

Las descripciones de las Áreas de Conocimiento deberían usar el siguiente índice de contenidos:

- Introducción
- Descomposición de temas del Área de Conocimiento (por claridad, creemos que esta sección debería de estar situarse al principio y no en un apéndice al final del documento. Además, debería de una figura describiendo la descomposición).
- Matriz de temas vs. Material de referencia
- Referencias recomendadas para el Área de Conocimiento descrita (por favor, no mezclarlas con las referencias usadas al escribir la descripción del Área de Conocimiento)
- Lista de Lecturas Complementarias

¿QUÉ QUEREMOS DECIR CON "CONOCIMIENTO GENERALMENTE ACEPTADO"?

El Cuerpo de Conocimiento de la Ingeniería del Software es un término que describe la suma de conocimiento dentro de la profesión de la ingeniería del software. Sin embargo, la Guía al Cuerpo de Conocimiento de la Ingeniería del Software busca identificar y describir que subconjunto de la ingeniería del software es generalmente aceptado o, en otras palabras, el núcleo del cuerpo del conocimiento. Para ilustrar mejor que "conocimiento generalmente aceptado" relativo a otros tipos de conocimiento, la Figura 1 propone un borrador para clasificar el conocimiento en un esquema con 3 categorías.

El *Project Management Institute* en su Guía al Cuerpo de Conocimiento de la Gestión de Proyectos[5] define conocimiento "generalmente aceptado" como:

"'Generalmente aceptado' significa que el conocimiento y practicas descritas son aplicables a la mayoría de los proyectos la mayor parte del tiempo, y que existe un consenso extendido sobre su valor y utilidad. 'Generalmente aceptado' no significa que el conocimiento y practicas descritas son o deberían de ser aplicadas uniformemente a todos los proyectos; el equipo de gestión de proyectos es siempre responsable de determinar que es apropiado a un proyecto dado".

La Guía al Cuerpo de Conocimiento de la Ingeniería del Software es un estándar IEEE.

En la reunión inicial en Mont Tremblant en 1998, el comité ejecutivo profesional definió "generalmente aceptado" como conocimiento a ser incluido en el material de estudio de un examen de licencia de ingeniería del software que un graduado con cuatro años de licencia debería de superar. Estas dos definiciones deben verse como complementarias.

También se espera de los editores asociados de las Áreas de Conocimiento miren al futuro en su interpretación de lo que es "generalmente aceptado" a día de hoy, pero lo que se espera que sea "generalmente aceptado" en 5 años.

Especializado Práticas utilizadas solamente en ciertos tipos de software	**Generalmente aceptado** Prácticas tradicionales establecidas que son recomendadas por muchas organizaciones.
	Avanzados y de investigación Practicas innovadoras probadas y usadas solo por algunas organizaciones y conceptos que están todavía siendo desarrollados y probados en organizaciones de investigación

LONGITUD EN LA DESCRIPCIÓN DE LAS ÁREA DE CONOCIMIENTO

Actualmente, se espera que la longitud de las Áreas de Conocimiento este sobre las 10 páginas usando el formato de la Conferencia Internacional de la Ingeniería del Software. Esto incluye texto, referencias, apéndices, tablas, etc. Esto, por supuesto, no incluye los materiales de referencia. Este límite no debería de ser visto como una restricción u obligación.

EL ROL DEL EQUIPO EDITORIAL

Alain Abran y James W. Moore son editores ejecutivos y responsables de mantener buenas relaciones con el *IEEE Computer Society*, el comité ejecutivo profesional, el panel de control de cambios, el panel de expertos y la estrategia general, aproximaciones a tomar, organización y financiación del proyecto.

Pierre Bourque y Robert Dupuis son los editores y responsables de la coordinación, operación y logística del proyecto. Más específicamente, los editores son los responsables del desarrollo del plan del proyecto y la especificación de la descripción del Área de conocimiento,

[5] Ver "A Guide to the Project Management Body of Knowledge," Project Management Institute, Newton Square, PA 1996, 2000; Disponible en: http://www.pmi.org/

coordinación de los editores asociados de las Áreas de Conocimiento, reclutamiento de revisores y de capitanear las revisiones, y también de los varios ciclos de revisión.

Los Editores son por tanto, los responsables de la coherencia de la Guía y de la identificación y establecimiento de los enlaces entre las Áreas de Conocimiento. Los editores y editores asociados de las Áreas de Conocimiento negociaran la resolución de vacíos y solapamientos entre las Áreas de Conocimiento.

IMPORTANTES DOCUMENTOS RELACIONADOS (EN ORDEN ALFABÉTICO DEL PRIMER AUTOR)

P. Bourque, R. Dupuis, A. Abran, J. W. Moore, L. Tripp, and D. Frailey, "A Baseline List of Knowledge Areas for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge," Université du Québec à Montréal, Montréal, February 1999.

Basado en la versión "Hombre de Paja", en las discusiones y en las expectativas marcadas en la reunión inicial del Comité Ejecutivo Profesional, en otros cuerpos del conocimiento, y en los criterios definidos en este documento, propone una línea base de 10 Áreas de Conocimiento para la versión de prueba de la Guía al Cuerpo de Conocimiento de la Ingeniería del Software. Por supuesto, esta línea base puede evolucionar al tratarse de un trabajo en progreso y según se van identificando temas durante el transcurso del proyecto.

Este documento está disponible en http://www.swebok.org/

P. Bourque, R. Dupuis, A. Abran, J. W. Moore, y L. Tripp, "A Proposed Baseline List of Related Disciplines for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge," Université du Québec à Montréal, Montréal, February 1999

Basado en la versión "Hombre de Paja", en las discusiones y en las expectativas marcadas en la reunión inicial del Comité Ejecutivo Profesional, en trabajos posteriores, este documento propone la línea base de las Disciplinas Relacionadas y Áreas de Conocimiento dentro de estas Disciplinas Relacionadas. Este documento ha sido envido y discutido en el Comité Ejecutivo Profesional, y una lista de Áreas de Conocimiento todavía tienen que ser identificadas para ciertas Disciplinas Relacionadas. Los editores asociados serán informados de la evolución de este documento.

La versión actual se encuentra disponible en http://www.swebok.org/

P. Bourque, R. Dupuis, A. Abran, J. W. Moore, L. Tripp, K. Shyne, B. Pflug, M. Maya, and G. Tremblay, Guide to the Software Engineering Body of Knowledge - A Straw Man Version, technical report, Université du Québec à Montréal, Montréal, September 1998.

Este informe es la base de todo el proyecto. Define la estrategia general del proyecto, su razón de ser, y los principios y procesos que fundan una lista inicial de Áreas de Conocimiento y Disciplinas Relacionadas.

La versión actual se encuentra disponible en http://www.swebok.org/

J. W. Moore, Software Engineering Standards, A User's Road Map, IEEE Computer Society Press, 1998.

Este libro describe el ámbito, roles, usos y las tendencias de los estándares más usados en la ingeniería del software. Se concentra en actividades importantes de la ingeniería del software – calidad y gestión de proyectos, ingeniería de sistemas, fiabilidad y seguridad (prevención de riesgos). El análisis y reagrupamiento de colecciones de estándares le muestran al lector relaciones clave entre estándares.

Aunque la Guía al Cuerpo de Conocimiento de la Ingeniería del Software no se trata del desarrollo de estándares en sí mismo, debe considerarse muy especialmente en todo el proyecto la compatibilidad de Guía con el conjunto de estándares de IEEE e ISO.

IEEE Standard Glossary of Software Engineering Terminology, Std 610.12-1990, IEEE, 1990.

La lista de referencias para la terminología es Merriam Webster's Collegiate Dictionary (10th ed.), IEEE Std 610.12, y nuevas propuestas de definiciones si son necesarias.

Information Technology – Software Life Cycle Processes, International std ISO/IEC 12207:1995(E), 1995.

Este estándar es considerado clave en relación a la definición de los procesos del ciclo de vida y ha sido adoptado por los dos principales organismos de estandarización en la ingeniería del software: ISO/IEC JTC1 SC7 y el comité de estándares de ingeniería del software de IEEE Computer Society. Además, ha sido designado como un estándar fundamental sobre el cual el Software Engineering Standards Committee (SESC) está actualmente armonizando toda su colección de estándares. Este estándar una entrada fundamental para la versión "Hombre de Paja".

Aunque no se persigue que la Guía al Cuerpo de Conocimiento de la Ingeniería de Software sea totalmente compatible con el estándar 12207, este estándar se mantiene como uno de los documentos clave para la versión Hombre de Piedra, y debe prestarse un cuidado especial en todo el proyecto el relación a la compatibilidad de la guiad con el estándar 12207.

Merriam Webster's Collegiate Dictionary (10th ed.).

Ver nota para de Std IEEE 610.12.

ESTILO Y GUÍAS TÉCNICAS

Las Descripciones de las Áreas de Conocimiento debería ajustarse al formato de las actas de la conferencia internacional de la ingeniería del software (International Conference in Software Engineering). Las plantillas están disponibles en:
`http://sunset.usc.edu/icse99/cfp /technical_papers.html`

Se espera que las Descripciones de las Áreas de Conocimiento sigan la guía de estilo del IEEE Computer Society. Ver:
http://www.computer.org/author/style/cs-style.htm

El formato preferido para la presentación del trabajo es Microsoft Word. Por favor, contáctese al equipo editorial si esto no es posible.

OTRAS GUÍAS DETALLADAS

Cuando se referencie la guía, recomendamos que se use el título completo "Guía al SWEBOK" en lugar de "SWEBOK".

Por simplicidad, recomendamos a los editores asociados de las Áreas de Conocimiento que no se usen pies de página. Deberían de intentar incluir su contenido en el cuerpo principal del documento.

Recomendamos usar en el texto referencias explicitas a estándares en lugar de insertar los números que referencian a la bibliografía. Creemos que permiten al lector una mejor exposición a la fuente y ámbito del estándar.

El texto acompañando las figuras y tablas debería de ser autocontenido o tener suficiente texto. Esto permite al lector saber lo que significan las lecturas y tablas.

Asegúrese de usar información correcta (actual) en las referencias (versiones, títulos, etc.)

Para asegurarse que la información contenida en la guía al SWEBOK no se queda rápidamente obsoleta, por favor, evítese nombrar directamente nombres de herramientas y productos. En su lugar, intente nombrar sus funciones. La lista de productos y herramientas siempre puede ponerse en un apéndice.

Se espera que se detallen los acrónimos usados, se haga un uso apropiado de los acrónimos, marcas, etc.

Las Descripciones de las Áreas de Conocimiento deberían de ser escritas siempre en tercera persona.

EDICIÓN

El Equipo Editorial y **ESTILO Y GUÍAS TÉCNICAS**

editores profesionales editaran las Descripciones de las Áreas de Conocimiento. La edición incluye corrección (gramática, puntuación y capitalización), estilo (adaptación al estilo de las revistas de la *Computer Society*), y edición de contenido (flujo, significado, claridad, lenguaje directo y organización). La edición final será un proceso colaborativo entre el Equipo Editorial y los autores de los trabajos para alcanzar unas Descripciones de las Áreas de Conocimientos concisas, bien escritas y útiles.

PUBLICACIÓN DEL COPYRIGHT

Toda propiedad intelectual asociada a la Guía al Cuerpo de Conocimiento de la Ingeniería del Software pertenecerá al IEEE Computer Society. Los Editores Asociados a las Áreas de Conocimiento firmaran un formulario sobre la publicación del copyright.

Además, la Guía al Cuerpo de Conocimiento de la Ingeniería del Software estará gratuitamente a disposición del público por parte del *IEEE Computer Society* en un sitio Web u otros medios.

Para más información, consúltese:
`http://www.computer.org/copyright.htm`

APÉNDICE B
EVOLUCIÓN DE LA GUÍA DEL CUERPO DE CONOCIMIENTO DE LA INGENIERÍA DEL SOFTWARE

INTRODUCCIÓN

Aunque la Guía al Cuerpo de Conocimiento de la Ingeniería del Software es un hito al haber alcanzado un amplio consenso en el contenido de la disciplina de la ingeniería del software, no es final del proceso. La Guía del 2004, es simplemente la presente edición de la Guía que continuara evolucionando para alcanzar las necesidades de la comunidad de la ingeniería del software. La planificación de su evolución no se ha completado todavía, pero en esta sección se proporciona un borrador. A la hora de escribir esto, el proceso ha sido endorsado por el Comité Ejecutivo Profesional y comunicado a la Junta de Gobierno del *IEEE Computer Society*, pero ha sido ni financiado ni implementado.

STAKEHOLDERS

La amplia adopción del SWEBOK ha generado una amplia comunidad de personas involucradas a parte de IEEE Computer Society. Existen un número de proyectos – dentro y fuera del IEEE Computer Society que están coordinando sus esfuerzos basándose su contenido en la Guía del SWEBOK. Varias empresas, incluyendo alguna de los miembros del Comité Ejecutivo Profesional, han adoptado la Guía para sus programas internos de educación. En un sentido más amplio, la comunidad de profesionales de la ingeniería del software y la comunidad de académicos prestan atención a la Guía del SWEBOK ayudando a definir el ámbito de sus esfuerzos. Un notable grupo de certificación del IEEE Computer Society – Certified Software Development Professional (CSDP) – por que el contenido del examen CSDP está completamente alineado con los contenidos de la Guía del SWEBOK.

Actualmente, el IEEE Computer Society y otras organizaciones están llevando a cabo un número de proyectos que dependen de la evolución de la Guía del SWEBOK:

- El examen CSDP, inicialmente desarrollado en paralelo con la Guía del SWEBOK, evolucionará muy cercano al SWEBOK tanto en su alcance[6] como en sus materiales de referencia.
- El plan de estudios para ingenieros del software para la educación a distancia del IEEE Computer Society tendrá el mismo alcance y material de referencia.
- Aunque los objetivos de la educación de grado difieren de los del desarrollo profesional, la unión de la ACM e IEEE-CS para el desarrollo de planes de estudio de grado son bastante consistes con el alcance de la Guía del SWEBOK.
- El IEEE-CS *Software Engineering Standards Committee* (SESC) ha organizado su colección de estándares siguiendo las Áreas de Conocimiento del SWEBOK, y el *IEEE Standards Association* ha publicado un CD-ROM que ya refleja esta organización.
- ISO/IEC JTC1/SC7, la organización internacional de estándares para ingeniería de sistemas e ingeniería software, está adoptando la Guía del SWEBOK como ISO/IEC Technical Report 19759 y armonizando su colección con los del IEEE.
- El IEEE Computer Society Press, en cooperación con el SESC, está desarrollando una serie de libros basándose en los estándares de la ingeniería del software y de Guía del SWEBOK.
- El portal de Ingeniería del Software del IEEE Computer Society ("SE Online") será organizado por las Áreas de Conocimiento de la Guía del SWEBOK.
- El Prueba de Uso de la Guía del SWEBOK ha sido traducido al Japonés. Se anticipa que la Versión 2004 sea también traducida al Japonés, Chino y posiblemente otros lenguajes.

EL PROCESO DE EVOLUCIÓN

Obviamente, un proceso como de este calibre debe evolucionar de manera abierta, consensuada, deliberada y transparente tal que otros proyectos puedan sucesivamente coordinar sus esfuerzos. La estrategia actualmente planificada es planificar la evolución siguiendo intervalos de tiempo específicos. De esta manera se permite al SWEBOK y proyectos coordinados revisiones que converjan en una fecha determinada. El intervalo planificado es de cuatro años.

Al comienzo del intervalo, consultando los proyectos coordinados, se determinara un plan para los cuatro años. Durante el primer año, también se determinaran cambios

[6] El CSDP añade una nueva Área de Conocimiento, Practicas de Negocio e Ingeniería Económica, a las 10 Áreas de Conocimiento de la Guía del SWEBOK.

estructurales a la Guía del SWEBOK, (por ejemplo, cambios en el número de Áreas de Conocimiento). Durante el segundo y tercer años, la selección y tratamiento de los temas dentro del Área de Conocimiento serán revisados. Durante el cuarto año, el texto de las descripciones de Área de Conocimiento será revisado y se seleccionaran referencias actualizadas.

El proyecto será gestionado por un comité de voluntarios del *IEEE Computer Society* y representantes de proyectos coordinados. El comité sería responsable de los planes globales, coordinar todos los grupos interesados en el proyecto, y la recomendación sobre la aprobación de la versión final. El comité será aconsejado por el *SWEBOK Advisory Committee* (SWAC) compuesto por los grupos que han adoptado la Guía del SWEBOK. En el pasado, la financiación por parte de la empresa nos ha permitido que el Guía del SWEBOK esté disponible gratuitamente en la Web. Futura financiación nos permitirá continuar esta práctica.

Cada uno de los cuatro años incluirá un ciclo de *workshops*, borrador, votación y resolución de la votación. Un ciclo anual puede contener las siguientes actividades:

- Un *workshop*, que organizado asociado a un congreso, especificara temas a tratar para el siguiente año, priorizar asuntos a tratar, recomendar como abordar los asuntos a tratar y nominar a quienes llevarán a cabo dicha tarea.
- Cada grupo haciendo borradores escribirá y modificará la descripción de un Área de Conocimiento utilizando las recomendaciones de los *workshops* y referencias disponibles. En el último año disponible las personas encargadas de los borradores recomendarán referencias actualizadas para su cita en la guía del SWEBOK. También se encargarán de modificar los borradores de acuerdo a las recomendaciones recibidas.
- Cada ciclo anual incluirá votaciones en las revisiones. Los votantes revisarán los borradores de las descripciones de las áreas de conocimiento y referencias recomendadas. La votación será abierta a los miembros de la *Computer Society* y otros participantes cualificados (los que no sean miembros tendrán que pagar para sufragar los gastos de la votación). Los que tengan el certificado CSDP serán particularmente bienvenidos como miembros de las votaciones o como voluntarios en otros roles.
- Un *Comité de Resolución de las Votaciones* será seleccionado por el comité de gestión para servir como intermediarios entre los encargados de los borradores y los votantes. Su trabajo es determinar el consenso para los cambios pedidos por el grupo de votantes y asegurar que se implementan los cambios necesarios. En algunos casos, *Comité de Resolución de las Votaciones* puede proponer preguntas a los votantes y usar sus respuestas para guiar las revisiones de los borradores. El objetivo de cada año es alcanzar consenso entre el grupo de votantes en las Áreas de Conocimiento revisadas o nuevas y alcanzar la aprobación de los votantes. Aunque la guía del SWEBOK no será cambiada hasta el final del ciclo, el material aprobado en cada ciclo anual se hará disponible gratuitamente.

Al final del ciclo, el producto completo, SWEBOK Guide 2008, será revisado y aprobado para su publicación por el Panel de Gobierno de la *Computer Society*. Si se consigue soporte financiero, el producto estará disponible gratuitamente en la Web.

CAMBIOS ANTICIPADOS

Nótese que la Guía del SWEBOK es un documento conservador por varias razones. Primero, se limita a conocimiento característico de la ingeniería del software; por lo tanto, la información de las disciplinas relacionadas (incluso aquellas que aquellas que se les aplica la ingeniería del software) son descartadas. Segundo, está desarrollada y aprobada por un proceso consensuado, por lo que no se almacena la información que no haya recibido una aceptación general. Tercero, el conocimiento contemplado como especializado en dominios específicos es excluido. Finalmente y muy importante, la Guía contiene sólo el conocimiento que es "generalmente aceptado". A veces es necesario en la actualidad el uso de técnicas de validación para obtener una aceptación general dentro de la comunidad.

Este enfoque conservador es evidente en la actual Guía SWEBOK. Después de 6 años de trabajo, aún contiene las 10 Áreas de Conocimiento. Una posible pregunta es si la selección de las Áreas de Conocimiento será algún día alterada. El plan de evolución contempla algunos criterios para añadir un Área de Conocimiento nuevo o para modificar el alcance de éstas. En principio, el candidato debe ser ampliamente reconocido dentro y fuera de la comunidad de ingeniería del software como representación de un área distintiva de conocimiento y el conocimiento generalmente aceptado dentro del área propuesta debe de estar suficientemente detallada y completa para ser tratada con el mismo mérito que aquellas que están ya presentes en la Guía SWEBOK. En término operacionales, debe de estar lo menos posible acoplado el Área de Conocimiento propuesto de las áreas ya existentes, y ese desacoplamiento debe de añadir un valor significante sobre la taxonomía del conocimiento provisto por la Guía. No obstante, simplemente por ser un área transversal no tiene justificación de ser tratada separadamente por separar, en muchos casos, simplemente genera un problema de solapamiento. En general, el crecimiento del número total de Áreas de Conocimiento tiende a ser evitado cuando hace que aumenten los esfuerzos de los lectores por encontrar la información deseada.

Añadiendo un tópico a Área de Conocimiento es más fácil. En principio, debe de estar madura (o, por lo menos, alcanzando la madurez rápidamente) y generalmente aceptada[2]. Los indicios para una aceptación general pueden ser encontrados en muchos sitios, incluyendo currículos y estándares de ingeniería del software, y

ampliamente usado en libros de texto. Por supuesto, los temas se deben de ajustar a un nivel de licenciado con cuatro años de experiencia[3].
Ese diseño incrementa el volumen de material referenciado en la Guía SWEBOK. La cantidad total de material debería estar diseñado para un nivel de licenciado con cuatro años de experiencia. Actualmente, el equipo editorial considera apropiado la cantidad de 5000 páginas como material de texto. Durante la evolución de la Guía, deberá ser necesario el administrar las listas de del material citado, por lo que las referencias son actualmente accesibles, abasteciendo una cobertura apropiada de las Áreas de Conocimiento, por medio de una considerada cantidad grande de material.

Un último tema es el rol que debe ser tomado por los usuarios de la Guía SWEBOK en su evolución. El equipo editorial cree que la continuada publicación de comentarios es el "combustible" para dirigir la evolución de la Guía SWEBOK. Los comentarios publicados incrementarán los temas a ser tratados por el taller anual, estableciendo el escenario para la revisión de la Guía SWEBOK. Nosotros esperamos proveer un foro público en línea para comentar por cualquier miembro de la comunidad de ingeniería del software y que sirva como un punto para realizar actividades.

---

2. Para la definición de "generalmente aceptado", usamos el que utiliza IEEE, en la Adopción de un Estándar PMI (Una Guía del Proyecto de la Administración del Cuerpo del Conocimiento): "Generalmente aceptado significa que el conocimiento y las prácticas descritas son aplicables a la mayoría de los proyectos la mayor parte del tiempo, y hay un extendido consenso sobre su valor y utilidad. Esto no significa que el conocimiento y las prácticas deban de ser aplicadas de forma uniforme en todos los proyectos sin considerar si son apropiados."
3. Por supuesto, esta especificación particular está indicada en términos relevantes en US. En otros países, puede ser indicada de forma diferente.

APÉNDICE C
ASIGNACIÓN DE IEEE E ISO ESTÁNDARES A LAS ÁREAS DE CONOCIMIENTO DEL SWEBOK

La siguiente tabla lista los estándares producidos por IEEE e ISO/IEC JTC1/SC7, al igual que otros estándares seleccionados de otras fuentes. Para cada estándar, se muestra su número y título. Además, la columna de descripción se proporciona un resumen sacado del propio estándar o de otros materiales introductorios. Cada uno de los estándares se mapea a las áreas de conocimiento del SWEBOK. En unos pocos casos – como estándares de terminología – el estándar es aplicable a todas las áreas de conocimiento; en estos casos aparece una "X" en cada columna. En la mayoría de los casos, cada estándar se fuerza un área primaria de conocimiento marcada con una "P" y otras secundarias marcadas con "S". La lista está ordenada por el número de estándar sin tener en cuenta su categoría (IEEE, ISO, etc.)

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE Std 610.12-1990 (R2002)	Glosario Estándar IEEE de la Terminología de la Ingeniería del Software	Este estándar es un glosario de la terminología de la ingeniería del software.	X	X	X	X	X	X	X	X	X	X
IEEE Std 730-2002	Estándar IEEE para los Planes de Aseguramiento de la Calidad del Software	Este estándar especifica el formato y el contenido de los planes de aseguramiento de la calidad del software.						S	S			P
IEEE Std 828-1998	Estándar IEEE para los Planes de Gestión de la Configuración del Software	Este estándar especifica el contenido de un plan de gestión de la configuración de un software junto con los requisitos que especifican las actividades.						P	S			
IEEE Std 829-1998	Estándar IEEE de la Documentación de Prueba del Software.	Este estándar describe la forma y el contenido de un conjunto básico de documentación para la planificación, ejecución, y la presentación de las pruebas del software.			S	P						S
IEEE Std 830-1998	Recomendación IEEE para la Práctica de la Especificación de Requisitos del Software.	Este documento recomienda el contenido y características de una especificación de requisitos del software. Esquemas de ejemplo son provistos.	P									

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE Std 982.1-1988	Estándar IEEE de Diccionario de Medidas para Producir Software Fiable.	Este estándar provee un conjunto de medidas para evaluar la fiabilidad de un producto software y para obtener fiabilidad de un producto en desarrollo.		S	S	S			S			P
IEEE Std 1008-1987 (R2003)	Estándar IEEE para las Pruebas Unitarias del Software	Este estándar describe con cierta aproximación las pruebas unitarias del software, así como los conceptos y supuestos en los cuales están basadas. También provee Fuentes de información y una guía.			S	P				S		S
IEEE Std 1012-1998 and 1012a-1998	Estándar IEEE para la Validación y Verificación del Software	Este estándar describe los procesos de verificación y validación que son utilizados para determinar si un producto software de una actividad tiene en cuenta todos los requisitos deseados y para determinar si el software satisface las necesidades del cliente. El ámbito incluye análisis, evaluación, revisión, inspección, evaluación de los resultados y prueba de los productos y procesos.										P
IEEE Std 1016-1998	Práctica IEEE Recomendada para las Descripciones del Diseño del Software	Este documento recomienda el contenido y la organización de una descripción del diseño de un software.		P								
IEEE Std 1028-1997 (R2002)	Estándar IEEE para la Revisión del Software	Este estándar define 5 tipos de revisiones del software y los procesos para su ejecución. Los tipos de revisión incluyen gestiones de las revisiones, revisiones técnicas, inspecciones, visitas guiadas y auditorías.	S	S	S			S	S			P

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE Std 1044-1993 (R2002)	Estándar IEEE para la Clasificación de Anomalías del Software.	Este estándar provee una aproximación uniforme de la clasificación de las anomalías encontradas en el software y su documentación. Incluye listas de ayuda para la clasificación de las anomalías de de la información relacionada.				S			S	S		P
IEEE Std 1045-1992 (R2002)	Estándar IEEE para las Métricas de Productividad del Software.	Este estándar provee una terminología consistente para medir la productividad del software y define la forma correcta de medir los elementos que involucrados en la productividad del software.							P	S		
IEEE Std 1058-1998	Estándar IEEE para la Planificación de la Gestión de un Proyecto Software	Este estándar describe el formato y el contenido de un plan de gestión de un proyecto software.							P			
IEEE Std 1061-1998	Estándar IEEE para la Metodología de las Métricas de Calidad del Software	Este estándar describe la metodología (abarcando todo el ciclo de vida) para establecer los requisitos de calidad, y para identificar, implementar y validar las medidas correspondientes.			S		S		S	S		P
IEEE Std 1062, Edición 1998	Práctica IEEE recomendada para la Adquisición del Software	Este documento recomienda un conjunto de útiles prácticas que pueden ser seleccionadas y aplicadas durante la adquisición del software. Esto se ajusta a la adquisición que incluye desarrollo o modificación más que a la propia compra en sí.	S						P			

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE Std 1063-2001	Estándar IEEE para la Documentación de Usuario del Software	Este estándar provee el mínimo de requisitos para la estructura, contenido, y formato de la documentación de usuario (ambos impresos y de forma electrónica).			P							S
IEEE Std 1074-1997	Estándar IEEE para el Proceso del Ciclo de Vida del Desarrollo.	Este estándar describe una aproximación para definición del proceso del ciclo de vida del software.								P		
IEEE Std 1175.1-2002	Guía IEEE para la Interconexión de Herramientas CASE. — Clasificación y Descripción.	Este estándar es el primero de las series planeadas de los estándares de integración de herramientas CASE dentro del entorno de producción de ingeniería del software. Esta parte describe conceptos fundamentales e introduce el resto de las partes.									P	
IEEE Std 1219-1998	Estándar IEEE para el Mantenimiento del Software.	Este estándar describe un proceso para el mantenimiento del software y su gestión.					P			S		
IEEE Std 1220-1998	Estándar IEEE para la Aplicación y Gestión de Procesos de Ingeniería de Sistemas.	Este estándar describe las actividades de ingeniería de sistemas y el proceso requerido en todo el ciclo de vida del sistema para el desarrollo de sistemas que permitan conocer las necesidades, requisitos de los clientes, y sus límites.								P		
IEEE Std 1228-1994	Estándar IEEE para Planes Seguros del Software.	Este estándar describe el mínimo contenido de un plan para aspectos del software tales como el desarrollo, obtención, mantenimiento, y la retirada de un sistema crítico-seguro.	S			S			S			P

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE Std 1233, Edición 1998	Guía IEEE para el Desarrollo de los Requisitos del Software.	Este documento provee una guía en el desarrollo de una especificación de requisitos del software, cubriendo la identificación, organización, presentación, y la modificación de requisitos. También provee una guía de las cualidades y características de los requisitos.	P									
IEEE Std 1320.1-1998	Estándar IEEE para el Lenguaje de Modelado Funcional — Sintaxis y Semántica para IDEF0.	Este estándar define el lenguaje de modelado IDEF0 usado para representar decisiones, acciones, y actividades de una organización o sistema. IDEF0 puede ser usado para definir los requisitos en términos de funciones para ser presentadas en el sistema deseado	S	S						S	P	
IEEE Std 1320.2-1998	Estándar IEEE para el Modelado Conceptual — Sintaxis y Semántica del Lenguaje para IDEF1X 97 (IDEFobject)	Este estándar define dos lenguajes de modelado conceptual, llamado generalmente IDEF1X97 (IDEFObject). El lenguaje soporta la implementación de bases de datos relacionales, de objetos, y modelos de objetos.	S	S							P	
IEEE Std 1362-1998	Guía IEEE para Información Tecnológica — Definición del Sistema — Documento de Concepto de Operaciones (ConOps)	Este documento provee la guía del formato y contenido de un documento de Concepto de Operaciones (ConOps), describiendo las características de un sistema propuesto desde un punto de vista del usuario.	P									

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE Std 1420.1-1995, 1420.1a-1996, y 1420.1b-1999 (R2002)	Estándar IEEE para Información Tecnológica — Reutilización del Software — Modelo de Datos para la Interoperabilidad de las Librerías de Reutilización	Este estándar y sus suplementos describen información que las librerías de reutilización del software deben ser capaces de cambiar según activos de intercambio.									P	
IEEE Std 1462-1998 // ISO/IEC 14102:1995	Estándar IEEE — Adopción del Estándar Internacional ISO/IEC 14102:1995 — Información Tecnológica — Guía para la Evaluación y la Selección de Herramientas CASE.	Este estándar provee los pasos para la evaluación y selección de las herramientas CASE.									P	
IEEE Std 1465-1998 // ISO/IEC 12119	Estándar IEEE, Adopción del Estándar Internacional ISO/IEC 12119:1994(E), Información Tecnológica — Paquete Software — Requisitos de Calidad y de prueba.	Este estándar describe la calidad de los requisitos que se ajustan específicamente a los paquetes software y a la guía de pruebas de los paquetes contra los requisitos.	S									P

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE Std 1471-2000	Recomendación Práctica IEEE para la Descripción Arquitectónica de Sistemas Intensivos de Software.	Este documento recomienda un marco conceptual y contenido para la descripción arquitectónica de sistemas intensivos de software.	**S**	**S**							**P**	
IEEE Std 1490-1998	Guía IEEE — Adopción del Estándar PMI — Una Guía para el Cuerpo de la Gestión del Proyecto del Conocimiento.	Este documento es la adopción IEEE del Cuerpo de la Gestión del Proyecto de Conocimiento definido por el Instituto de Gestión del Proyecto. Identifica y describe al conocimiento generalmente aceptado con lo que concierne a la gestión del proyecto.							**P**			
IEEE Std 1517-1999	Estándar IEEE para la Información Tecnológica — Procesos del Ciclo de Vida del Software — Reutilización de Procesos	Este estándar provee el ciclo de vida de los procesos para la reutilización sistemática del software. Los procesos son idóneos para ser utilizados con IEEE/EIA 12207.			**S**					**P**		
IEEE Std 1540-2001 // ISO/IEC 16085:2003	Estándar IEEE para los Procesos del Ciclo de Vida del Software — Gestión de problemas	Este estándar provee un proceso de ciclo de vida para gestionar problemas del software. El proceso es idóneo para ser usado con IEEE/EIA 12207.							**S**	**P**		

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE Std 2001-2002	Práctica IEEE Recomendada para Internet — Ingeniería, gestión y ciclo de vida del Sitio Web	Este documento recomienda prácticas para la ingeniería de las páginas WWW para usar entornos de redes internas y externas.										P
ISO 9001: 2000	Sistemas de Gestión de Calidad — Requisitos	Este estándar especifica los requisitos para un sistema de gestión de calidad organizacional, consiguiendo proveer requisitos de los productos y mejorar la satisfacción del usuario.								S		P
ISO/IEC 9126-1:2001	Ingeniería del Software — Calidad del Producto — Parte 1: Modelo de calidad	Este estándar provee un modelo para la calidad del producto software cubriendo la calidad interna, externa y la calidad en uso. El modelo tiene una taxonomía de las características definidas que el software debe exhibir.	P	S	S	S						
IEEE/EIA 12207.0-1996 // ISO/IEC 12207: 1995	Implementación Industrial del Estándar Internacional ISO/IEC 12207:1995, Estándar para la Información Tecnológica — Procesos del Ciclo de Vida del Software	Este estándar provee una marca de procesos usado en todo el ciclo de vida del software.	X	X	X	X	X	X	X	P	X	X

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
IEEE/EIA 12207.1 - 1996	Implementación Industrial del Estándar Internacional ISO/IEC 12207:1995, Estándar para la Información Tecnológica — Procesos del Ciclo de Vida del Software — Ciclo de vida de la Información	Este documento provee una guía para registrar información resultante de los procesos del ciclo de vida de IEEE/EIA 12207.0.	X	X	X	X	X	X	X	P	X	X
IEEE/EIA 12207.2-1997	Implementación Industrial del Estándar Internacional ISO/IEC 12207:1995, Estándar para la Información Tecnológica — Procesos del Ciclo de Vida del Software — Consideraciones en la Implementación	Este documento provee una guía adicional para la implementación del ciclo de vida de los procesos de IEEE/EIA 12207.0.	X	X	X	X	X	X	X	P	X	X
IEEE Std 14143.1-2000 // ISO/IEC 14143-1:1998	Adopción IEEE de ISO/IEC 14143-1:1998 — Información Tecnológica — Medida del Software — Medida del Tamaño Funcional — Parte 1: Definición de Conceptos	Este estándar describe los conceptos fundamentales de una clase de medida colectiva conocida como tamaño funcional.	P				S		S			S

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
ISO/IEC TR 14471:1999	Información Tecnológica — Ingeniería del Software — Guías para la Adopción de herramientas CASE	Este documento provee una guía que establece los procesos y actividades que pueden ser utilizadas en la adopción de la tecnología CASE.									P	
ISO/IEC 14598 (Seis partes)	Información Tecnológica — Evaluación del producto Software	La serie ISO/IEC 14598 da una visión de los procesos de evaluación del producto software y nos ofrece una guía y los requisitos necesarios para la evaluación organizacional o del nivel del proyecto. Los estándares nos ofrecen métodos de medida, valoración y evaluación.										P
ISO/IEC 14764:1999	Información Tecnológica — Mantenimiento del Software	Este estándar da más detalle del proceso de mantenimiento ofrecido por ISO/IEC 12207. Provee una guía de implementación de los requisitos de ese proceso.					P					
ISO/IEC 15026:1998	Información Tecnológica — Niveles de Integridad del Software y del Sistema	Este Estándar Internacional introduce los conceptos de los niveles y requisitos de integridad del software. Define los conceptos asociados con los niveles de integridad, define los procesos para determinar los niveles de integridad y los requisitos de integridad software, y establece los requisitos en cada proceso.	S	S								P

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
ISO/IEC TR 15271:1998	Información Tecnológica — Guía para ISO/IEC 12207 (Procesos del Ciclo de Vida del Software)	Este documento es una guía para el uso de ISO/IEC 12207.								P		
ISO/IEC 15288:2002	Ingeniería de Sistemas — Procesos del Ciclo de Vida del Sistema	Este estándar nos ofrece en marco de procesos usados a lo largo del ciclo de vida de los sistemas hechos por el hombre.								P		
ISO/IEC TR 15504 (9 partes) y la Preparación de IS 15504 (5 partes)	Ingeniería del Software — Proceso de Valoración	Este documento técnico (ahora está siendo revisado como un estándar) nos ofrece los requisitos para los métodos que nos permiten valorar los procesos para mejorar los procesos o la capacidad de determinación.								P		
ISO/IEC 15939:2002	Ingeniería del Software — Proceso de Medida del Software	Este estándar nos ofrece un proceso de ciclo de vida para la medición de software. El proceso se ajusta para ser usado con IEEE/EIA 12207.							S	P		S
ISO/IEC 19761:2003	Ingeniería del Software — COSMIC-FFP — Método de medición del tamaño funcional.	Este estándar describe el Método de Medición de Tamaño Funcional COSMIC-FFP, un método de medición de tamaño funcional conforme a los requisitos de ISO/IEC 14143-1.	P				S		S			S
ISO/IEC 20926:2003	Ingeniería del Software — IFPUG 4.1 Método de medida del tamaño funcional desajustado — Manual de Prácticas de Cómputo	Este estándar describe el Cómputo de Puntos de Función Desajustado de IFPUG 4.1, un método que mide el tamaño funcional conforme a los requisitos de ISO/IEC 14143-1.	P				S		S			S

Número de estándar	Nombre del estándar	Descripción	Requisitos del Software	Diseño del Software	Construcción del Software	Pruebas del Software	Mantenimiento del Software	Gestión de la configuración del Software	Gestión en la Ingeniería del Software	Procesos de la Ingeniería del Software	Herramientas y Métodos en la Ingeniería del Software	Calidad
ISO/IEC 20968:2002	Ingeniería del Software — Análisis de Puntos de Función Mk II — Manual de Prácticas de Cómputo	Este estándar describe el Análisis de Puntos de Función Mk II, un método de medición del tamaño funcional conforme a los requisitos de ISO/IEC 14143-1.	P				S		S			S
ISO/IEC 90003	Ingeniería del Software y Sistemas — Pasos para la Aplicación de ISO 9001:2000 al Software del Ordenador	Este estándar provee una serie de pasos para la organización en la aplicación de ISO 9001:2000 para la adquisición, suministro, desarrollo, operación, y mantenimiento del software del ordenador.								S		S

APÉNDICE D
CLASIFICACIÓN DE CONTENIDOS ACORDE A LA TAXONOMÍA DE BLOOM

INTRODUCCIÓN

La taxonomía de Bloom[1] es una clasificación de fines educativos cognitivos ampliamente conocidos y usados. Atendiendo a la ayuda de las personas que desean usar la Guía como herramienta en la definición del material usado en el curso, currículo universitario, criterios de acreditación en el programa universitario, descripción de trabajos, como rol descriptivo dentro un proceso de definición de ingeniería del software, programa de prueba profesional y vía de desarrollo profesional, y otras necesidades, que los niveles de la taxonomía de Bloom para los tópicos de la Guía SWEBOK son propuestos en este apéndice para graduados en ingeniería del software con unos cuatro años de experiencia. Un graduado en ingeniería del software con al menos cuatro años de experiencia es en esencia el objetivo fijado en la Guía SWEBOK por el cual es aceptable el conocimiento contenido en dicha guía (Lea la Introducción de la Guía SWEBOK).

Desde este Apéndice sólo se hace referencia a lo que puede ser considerado como conocimiento "generalmente aceptado". Es muy importante recordar que la ingeniería del software debe ser aprendida de una manera más sustancial que esta "categoría" de conocimiento. Además del conocimiento "generalmente aceptado", un graduado de ingeniería del software con cuatro años de experiencia debe de poseer algunos elementos de Disciplinas Relacionadas, como también algunos elementos de ciertos conocimientos específicos, avanzados, hasta la posibilidad de estudiar e investigar algunas materias (Mire la Introducción de la Guía SWEBOK). Los siguientes supuestos son creados cuando se especificaron los niveles propuestos de la taxonomía.

- Las evaluaciones son propuestas por una ingeniería del software "generalista" y no por una por una ingeniería del software que trabaja en un grupo específico, como puede ser un equipo de gestión de la configuración del software. Obviamente, como ingeniería del software requiere o necesita alcanzar niveles muy altos en taxonomía en el área específica de su grupo.

- Un ingeniero del software con cuatro años de experiencia está aún en el principio de su carrera y le puede ser asignado la gestión de una seria de deberes, o por lo menos, unos mayores esfuerzos. Los tópicos relacionados con la Gestión no se les asigna prioridad en las evaluaciones propuestas. Por la misma razón, los niveles de la taxonomía tienden a ser menores para un temprano ciclo de vida de contenidos tales como los relacionados con los requisitos del software, más que los contenidos de índole técnico dentro del diseño, construcción o prueba del software.

- Por lo tanto, las evaluaciones pueden ser adaptadas por más ingenieros del software sénior o por ingenieros del software especializados en ciertas áreas de conocimiento, en el que no hay ningún contenido que dé un mayor nivel de taxonomía que el Análisis. Éste es consistente con la aproximación dada en la SEEK (Software Engineering Education Bosy of Knowledge), donde no hay ningún contenido asignado de mayor nivel de taxonomía que Aplicación[2]. El propósito de SEEK es definir un cuerpo de educación de ingeniería del software de conocimiento apropiado para guiar el desarrollo del currículum de un estudiante universitario de ingeniería del software. Aunque notablemente distinto en términos de alcance, SEEK y la Guía SWEBOK son muy parecidos.[3]

La taxonomía de Bloom del Dominio Cognitivo propuesto en 1956 tiene seis niveles. La tabla 1 [4] presenta estos niveles y las palabras claves asociadas con cada nivel.

1. B. Bloom (ed.), Taxonomía de los Objetivos Educativos: La Clasificación de las Metas Educativas, Mackay, 1956.

2. Mire la Fuerza de la Tarea Conjunta en el Currículum de Cómputo – Sociedad Computacional IEEE / Asociación para la Maquinaria Computacional, Currículum de Cómputo – Volumen de Ingeniería del Software – Borrador público 1 – Ingeniería del Software de Currículum Computacional, 2003: http://sites.computer.org/ccse/.

3. Mire P.Bourque, F. Robert, J. –M. Lavoie, A. Lee, S. Trudel, T. Lethbridge, "Guía para el Conocimiento del Cuerpo de la Educación de la Ingeniería del Software (SEEK) – Un Mapeo Preliminar". Taller de Tecnología del Software y Conferencia de la Práctica de la Ingeniería (STEP 2002), 2002, pp. 8-35.

4. Tabla adaptada de http://www.nwlink.com/~donclark/hrd/bloom.html.

Tabla 1 Taxonomía de Bloom

Nivel de Taxonomía de Bloom	Palabras claves asociadas
Conocimiento: Almacenamiento de la información	Define, describe, identifica, conoce, etiqueta, listas, ajustes, nombres, esquemas, memorias, reconocimiento, reproduce, selecciona, estados
Comprensión: Entender el significado, traducción, interpolación e interpretación de las instrucciones y problemas; plantea un problema en una de las propias palabras.	Comprende, convierte, defiende, distingue, estima, explica, prolonga, generaliza, da ejemplos, deduce, interpreta, ilustra, predice, reescribe, resume, traduce.
Aplicación: Utiliza un concepto en una nueva situación o usa una abstracción espontánea; aplica lo que fue aprendido en la clase en situaciones novedosas en el lugar del trabajo.	Aplica, cambia, computa, construye, demuestra, descubre, manipula, modifica, opera, predice, prepara, muestra, resuelve y usa.
Análisis: Separa material o conceptos en partes que su estructura organizacional puede ser comprendida; distingue entre hechos e interferencias.	Analiza, descompone, compara, contrasta, diagramas, destruye, diferencia, discrimina, distingue, identifica, ilustra, infiere, esquematiza, relata, selecciona, separa.
Síntesis: Crea una estructura o prototipo de una serie de diversos elementos; une partes para crear un conjunto, con énfasis en la creación de una nueva estructura o significado.	Categoriza, combina, compila, compone, crea, idea, diseña, explica, genera, modifica, organiza, planifica, cambia de lugar, reestructura, relata, reorganiza, revisa, reescribe, resume, cuenta, escribe.
Evaluación: Hace un juicio sobre el valor de las ideas o materiales.	Tasa, compa, concluye, contrasta, critica, defiende, describe, discrimina, evalúa, explica, interpreta, justifica, relata, resume, acepta.

La separación de contenidos en las tablas no coincide perfectamente con la separación de las Áreas de Conocimiento. La evaluación para este Apéndice fue preparado mientras algunos comentarios aún llegaban.

Finalmente, mantén por favor en tu mente que la evaluación de este Apéndice debe ser visto definitivamente como una propuesta para ser desarrollada y validada.

REQUISITOS DEL SOFTWARE[5]

Desglose de contenidos	Nivel de taxonomía
1. Requisitos fundamentales del software	
Definición de requisitos del software	C
Requisitos del producto y proceso	C
Requisitos funcionales y no funcionales	C
Propiedades emergentes	C
Requisitos cuantificables	C
Requisitos del software y del sistema	C
2. Requisitos del proceso	
Modelos del proceso	C
Actores del proceso	C
Gestión y soporte del proceso	C
Calidad y mejora del proceso	C
3. Obtención de requisitos	
Fuentes de los requisitos	C
Técnicas de obtención	AP
4. Requisitos de análisis	
Clasificación de los requisitos	AP
Modelo conceptual	AN
Diseño arquitectónico y requisitos de cuota	AN
Requisitos de negociación	AP
5. Especificación de requisitos	
Documento de definición del sistema	C
Especificación de los requisitos del sistema	C
Especificación de los requisitos del software	AP
6. Especificación de requisitos	
Análisis de requisitos	AP
Prototipado	AP
Validación del modelo	C
Test de aceptación	AP
7. Especificación de requisitos	
Naturaleza iterativa de los requisitos del proceso	C
Gestión de cambio	AP
Atributos de los requisitos	C
Localización de requisitos	AP
Requisitos de medida	AP

5. K: Conocimiento, C: Comprensión, AP: Aplicación, AN: Análisis, E: Evaluación, S: Síntesis.

DISEÑO SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Especificación de requisitos	
Conceptos generales de diseño	C
Contexto del diseño software	C
Proceso del diseño software	C
Técnicas de habilitación	AN
2. Temas clave en el diseño software	
Concurrencia	AP
Manejo y control de eventos	AP
Distribución de componentes	AP
Manejo de errores y excepciones y falta de tolerancia	AP
Interacción y presentación	AP
Persistencia de los datos	AP
3. Estructura y arquitectura del software	
Estructuras y puntos de vistas de la arquitectura	AP
Estilos de arquitectura (Patrones de arquitectura)	AN
Patrones de diseño (Patrones de arquitectura)	AN
Familias de programas y de marcos de trabajo	C
4. Análisis y evaluación de la calidad del diseño software	
Atributos de calidad	C
Técnicas de análisis y evaluación de la calidad	AN
Medidas	C
5. Notaciones de diseño del software	
Descripciones estructurales (Estático)	AP
Descripciones del comportamiento (Dinámico)	AP
6. Métodos y estrategias del diseño del software	
Estrategias generales	AN
Diseño orientado a funciones (Estructurado)	AP
Diseño orientado a objetos	AN
Diseño centrado en la estructura de datos	C
Diseño basado en componentes (CBC)	C
Otros métodos	C

CONSTRUCCIÓN DEL SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Fundamentos de la construcción del software	
Minimización de la complejidad	AN
Anticipación al cambio	AN
Construyendo para verificar	AN
Estándares en la construcción	AP
2. Gestión de la construcción	
Métodos de construcción	C
Plan de construcción	AP
Medidas de construcción	AP
3. Consideraciones prácticas	
Fuentes de los requisitos	C
Técnicas de obtención	AP
4. Requisitos de análisis	
Diseño de la construcción	AN
Lenguajes de la construcción	AP
Codificación	AN
Pruebas de la construcción	AP
Calidad de la construcción	AN
Integración	AP

PRUEBAS DEL SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Fundamentos de las pruebas del software	
Terminología relacionada con las pruebas	C
Contenidos claves	AP
Relación de las pruebas con otras actividades	C
2. Niveles de pruebas	
El objetivo de las pruebas	AP
Los fines de probar	AP
3. Técnicas de pruebas	
Basado en la intuición y experiencia del probador	AP
Basados en la especificación	AP
Basados en el código	AP
Basados en los fallos	AP
Basados en el uso	AP
Basados en la naturaleza de la aplicación	AP
Seleccionando y combinando técnicas	AP
4. Medidas relacionadas con las pruebas	
Evaluación de un programa bajo pruebas	AN
Evaluación de las pruebas desarrolladas	AN
5. Proceso de prueba	
Lo que la gestión incumbe	C
Actividades de las pruebas	AP

MANTENIMIENTO DEL SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Fundamentos del mantenimiento del software	
Definiciones y terminología	C
Naturaleza del mantenimiento	C
Necesidad del mantenimiento	C
Mayoría de los costes del mantenimiento	C
Evolución del software	C
Categorías del mantenimiento	AP
2. Contenidos claves en el mantenimiento del software	
Técnico	
Comprensión limitada	C
Pruebas	AP
Análisis de impacto	AN
Mantenibilidad	AN
Temas de gestión	
Alineación con asuntos organizacionales	C
Personas	C
Asuntos de procesos	C
Organizacional	C
Estimación del coste de mantenimiento	
Estimación del coste	AP
Modelos parametrizados	C
Experiencia	AP
Medida del mantenimiento del software	AP
3. Proceso de mantenimiento	
Modelos de procesos de mantenimiento	C
Actividades de mantenimiento	
Actividades de unicidad	AP
Actividades de soporte	AP
4. Técnicas para el mantenimiento	
Comprensión del programa	AN
Reingeniería	C
Ingeniería inversa	C

GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Gestión de los procesos SCM	
Contexto organizacional para SCM	C
Restricciones y guía para SCM	C
Planificación para SCM	
Organización y responsabilidades SCM	AP
Esquemas y recursos SCM	AP
Selección de herramientas e implementación	AP
Control del vendedor/subcontratista	C
Control de la interface	C
Plan de gestión de la configuración del software	C
Gestión de la configuración de la vigilancia del software	
Medición y medidas SCM	AP
Auditorías de SCM	C
2. Identificación de la configuración del software	
Identificación de los elementos a ser controlados	
Configuración del software	AP
Elementos de la configuración del software	AP
Relaciones de los elementos de la configuración del software	AP
Versiones del software	AP
Línea base	AP
Elementos de la configuración de la adquisición del software	AP
Librería software	C
3. Control de la configuración del software	
Solicitación, evaluación y aprobación de los cambios software	
Tabla de control de la configuración del software	AP
Proceso de petición del cambio software	AP
Implementación de los cambios software	AP
Desviaciones y renuncias	C

4. Informe del estado de la configuración software	
Información del estado de la configuración del software	C
Reporte del estado de la configuración del software	AP
5. Auditoría de la configuración del software	
Auditoría de la configuración funcional del software	C
Auditoría de la configuración física del software	C
Proceso de auditoría de en la línea base del software	C
5. Gestión de la salida al mercado y entrega del software	
Construcción del software	AP
Gestión de la venta al mercado del software	

GESTIÓN DE LA INGENIERÍA DEL SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Iniciación y alcance de la definición	
Determinación y negociación de los requisitos	AP
Análisis de viabilidad	AP
Proceso para el análisis y revisión del software	C
2. Planificación del proyecto software	
Proceso de planificación	C
Determinación de las entregas	AP
Esfuerzo, programa y estimación de coste	AP
Asignación de recursos	AP
Gestión de problemas	AP
Gestión de la calidad	AP
Plan de gestión	C
3. Difusión del proyecto software	
Implementación de planes	AP
Gestión del contrato de proveedores	C
Implementación de procesos de medición	AP
Proceso de monitoreo	AN
Proceso de control	AP
Informando	AP
4. Revisión y evaluación	
Determinación de los requisitos de satisfacción	AP
Revisión y evaluación del rendimiento	AP
5. Cierre	
Determinación del cierre	AP
Actividades del cierre	AP
6. Medición de la ingeniería del software	
Establecimiento y mantenimiento de las mediciones	C
Plan para el proceso de medición	C
Realización del proceso de medición	C
Evaluación de la medición	C

PROCESO DE INGENIERÍA DEL SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Proceso de implementación y cambio	
Infraestructura del proceso	
Grupo de procesos de ingeniería del software	C
Factoría de experiencia	C
Actividades	AP
Modelos para el proceso de implementación y cambio	K
Consideraciones prácticas	C
2. Definición del proceso	
Modelos de ciclo de vida	AP
Procesos de ciclo de vida del software	C
Notaciones para las definiciones del proceso	C
Proceso de adaptación	C
Automatización	C
3. Valoración del proceso	
Modelos de valoración del proceso	C
Métodos de valoración del proceso	C
4. Medición del proceso y del producto	
Medición del proceso software	AP
Medición del producto software	AP
Medición del tamaño	AP
Medición de la estructura	AP
Medición de la calidad	AP
Resultados de las mediciones de la calidad	AN
Modelos de información del software	
Modelo de construcción	AP
Modelo de implementación	AP
Técnicas de medición	
Técnicas analíticas	AP
Técnicas de punto de referencia	C

MÉTODOS Y HERRAMIENTAS DE INGENIERÍA DEL SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Herramientas software	
Herramientas de requisitos del software	AP
Herramientas de diseño del software	AP
Herramientas de construcción del software	AP
Herramientas de pruebas del software	AP
Herramientas de mantenimiento del software	AP
Herramientas del proceso de ingeniería del software	AP
Herramientas de calidad del software	AP
Herramientas de gestión de la configuración del software	AP
Herramientas de gestión de la ingeniería del software	AP
Herramientas para otros diversos asuntos	AP
2. Métodos de ingeniería del software	
Métodos heurísticos	AP
Notaciones y métodos formales	C
Métodos de prototipado	AP
Métodos para otros diversos asuntos	C

CALIDAD DEL SOFTWARE

Desglose de contenidos	Nivel de taxonomía
1. Fundamentos de la calidad del software	
Ética y cultura de la ingeniería del software	AN
Valor y coste de la calidad	AN
Características y modelos de la calidad	
Calidad del proceso software	AN
Calidad del producto software	AN
Mejora de la calidad	AP
2. Procesos de gestión de la calidad del software	
Aseguramiento de la calidad del software	AP
Verificación y validación	AP
Revisión y auditoría	
Inspecciones	AP
Revisiones minuciosas	AP
Revisión de la funcionalidad, entradas y salidas del software	AP
Pruebas	AP
Auditorías	C
3. Consideraciones prácticas	
Aplicación de los requisitos de la calidad	
Nivel de criticidad de los sistemas	C
Dependencia	C
Integridad de los niveles del software	C
Caracterización imperfecta	AP
Técnicas de gestión de la calidad del software	
Técnicas estáticas	AP
Técnicas intensivas de personas	AP
Técnicas analíticas	AP
Técnicas dinámicas	AP
Medición de la calidad del software	AP