Guía para la ingeniería de software
Cuerpo de conocimientos

Versión 3.0

SWEBOK ®

Un proyecto de la IEEE Computer Society

Guía para la ingeniería de software
Cuerpo de conocimientos

Versión 3.0

Editores

Pierre Bourque, École de technologie supérieure (ÉTS)
Richard E. (Dick) Fairley, Asociados de Ingeniería de Software y Sistemas (S2EA)

Tapa blanda ISBN-10: 0-7695-5166-1
Tapa blanda ISBN-13: 978-0-7695-5166-1

Las copias digitales de *SWEBOK Guide* V3.0 se pueden descargar de forma gratuita para uso personal y académico a través de www.swebok.org .

IEEE Computer Society Staff para esta publicación
Angela Burgess, Directora Ejecutiva
Anne Marie Kelly, directora ejecutiva asociada, directora de gobierno
Evan M. Butterfield, Director de Productos y Servicios
John Keppler, Gerente Senior, Educación Profesional
Kate Guillemette, Editora de Desarrollo de Producto
Dorian McClenahan, desarrolladora de productos del programa educativo
Michelle Phon, Coordinadora del Programa de Educación Profesional y Certificación
Jennie Zhu-Mai, diseñadora editorial

Productos y servicios de la IEEE Computer Society. La mundialmente conocida IEEE Computer Society publica, promueve y desarma rinde homenaje a una amplia variedad de revistas, revistas, actas de congresos y ciencias de la computación autorizadas Productos de educación profesional. Visite Computer Society en www.computer.org para obtener más información.

TABLA DE CONTENIDO

Página 6

Página 7

Página 8

Página 9

Página 10

Página 11

Pagina 12

Página 13

Página 14

Página 15

Página 16

PREFACIO

Toda profesión se basa en un cuerpo de conocimiento. borde, aunque ese conocimiento no siempre es definido de manera concisa. En casos donde no existe formalidad, el cuerpo de conocimiento es "gen- reconocidos por vía oral "por los practicantes y pueden ser codificado en una variedad de formas para una variedad de diferentes usos Pero en muchos casos, una guía para un cuerpo de conocimiento está formalmente documentado, usu- aliado en una forma que le permita ser utilizado para tal propósitos como desarrollo y acreditación de programas académicos y de formación, certificación de especialistas o licencias profesionales. Generalmente, una sociedad profesional u organismo similar mantiene administración de la definición formal de un cuerpo de conocimiento.

Durante los últimos cuarenta y cinco años, la ingeniería de software Neering ha evolucionado a partir de una conferencia de captura- frase en una profesión de ingeniería, personaje- ized por 1) una sociedad profesional, 2) estándares que especificar prácticas profesionales generalmente aceptadas, 3) un código de ética, 4) actas de la conferencia, 5) libros de texto, 6) pautas y currículos curriculares ricula, 7) criterios de acreditación y acreditados programas de grado, 8) certificación y licencia, y 9) esta Guía del Cuerpo del Conocimiento.

En esta *guía para el organismo de ingeniería de software de Conocimiento* , la IEEE Computer Society pres- Entra una versión revisada y actualizada del cuerpo de conocimiento anteriormente documentado como SWEBOK 2004; esta versión revisada y actualizada se denota SWEBOK V3. Este trabajo está en cumplimiento parcial de la responsabilidad de la Sociedad de promover el avance de la teoría y la práctica para el profesión de ingeniería de software.

Cabe señalar que esta *guía* no presentar todo el cuerpo de conocimiento para soft- ingeniería de artículos, sino que sirve como guía para el cuerpo de conocimiento que se ha desarrollado durante más de cuatro décadas. El software de ingeniería el cuerpo de conocimiento de neering está en constante evolución En g. Sin embargo, esta *guía* constituye un valor

En 1958, John Tukey, la estadística de renombre mundial istician, acuñó el término *software* . El término blando la ingeniería de la cerámica se utilizó en el título de una OTAN conferencia celebrada en Alemania en 1968. El IEEE Computer Society publicó por primera vez sus *Transacciones de Ingeniería de Software* en 1972, y un compromiso tee para desarrollar estándares de ingeniería de software Dards se estableció dentro de la computadora IEEE Sociedad en 1976.

En 1990, se inició la planificación de una estándar nacional para proporcionar una visión general de software Ingeniería de artículos. El estándar se completó en 1995 con designación ISO / IEC 12207 y dada el título de *Standard for Software Life Cycle Pro- cesses* . Se publicó la versión IEEE de 12207. en 1996 y proporcionó una base importante para el cuerpo de conocimiento capturado en SWEBOK 2004. La versión actual de 12207 se designa como ISO / IEC 12207: 2008 e IEEE 12207-2008; eso proporciona la base para este SWEBOK V3.

Esta *guía para el cuerpo de ingeniería de software of Knowledge* se le presenta a usted, el lector, como Un mecanismo para adquirir el conocimiento que usted necesita en su desarrollo profesional de por vida como profesional de ingeniería de software.

Dick Fairley, presidente
Comité de Ingeniería de Software y Sistemas
IEEE Computer Society

Don Shafer, Vicepresidente
Junta de Actividades Profesionales
IEEE Computer Society

caracterización capaz de la ingeniería de software
profesión.

Página 18

PRÓLOGO A LA EDICIÓN 2004

En esta *guía* , la IEEE Computer Society establece
riega por primera vez una línea de base para el cuerpo
de conocimiento para el campo del ingeniero de software
ing, y el trabajo cumple parcialmente con la Sociedad
responsabilidad de promover el avance de
tanto teoría como práctica en este campo. Al hacerlo,
la sociedad se ha guiado por la experiencia
de disciplinas con historias más largas pero no fue
obligado ya sea por sus problemas o sus soluciones.

Cabe señalar que la *Guía* no tiene
puerto para definir el cuerpo de conocimiento, sino más bien para
servir como compendio y guía para el cuerpo de
conocimiento que se ha estado desarrollando y evolucionando
ing en las últimas cuatro décadas. Además,
Este cuerpo de conocimiento no es estático. La *guía*
debe, necesariamente, desarrollarse y evolucionar como software
La ingeniería madura. Sin embargo, constituye
un elemento valioso de la ingeniería de software
infraestructura.

En 1958, John Tukey, la estadística de renombre mundial
istician, acuñó el término *software* . El término *blando*
la ingeniería de la cerámica se utilizó en el título de una OTAN
conferencia celebrada en Alemania en 1968. El IEEE
Computer Society publicó por primera vez sus *Transacciones*
en Ingeniería de Software en 1972. El comité
establecido dentro de la IEEE Computer Society
para desarrollar estándares de ingeniería de software
fue fundado en 1976.

La primera visión holística del ingeniero de software.
surgirá de la IEEE Computer Society
resultado de un esfuerzo dirigido por Fletcher Buckley
desarrollar el estándar IEEE 730 para software cualificado
Garantía de seguridad, que se completó en 1979.
El propósito de IEEE Std. 730 fue proporcionar
requisitos mínimos aceptables uniformes para
preparación y contenido de garantía de calidad de software
planes ance. Este estándar fue influyente en
cumpliendo los estándares de desarrollo en los siguientes
temas: gestión de configuración, prueba de software
ing, requisitos de software, diseño de software y
Verificación y validación de software.

Durante el período 1981–1985, el IEEE Com-
La sociedad informática realizó una serie de talleres
sobre la aplicación de la ingeniería de software
normas Estos talleres involucraron prácticas
ners compartiendo sus experiencias con los estándares existentes
papás Los talleres también celebraron sesiones sobre planificación
Ning para estándares futuros, incluyendo uno que involucre
medidas y métricas para el ingeniero de software
ing productos y procesos. La planificación también
resultó en IEEE Std. 1002, *Taxonomía de software*
Estándares de ingeniería (1986), que proporcionaron un
Nueva visión holística de la ingeniería de software. los
estándar describe la forma y el contenido de un software
programas de ingeniería de mercancías taxonomía. Explica
los diversos tipos de estándares de ingeniería de software
los papás, sus relaciones funcionales y externas,
y el papel de varias funciones que participan en
El ciclo de vida del software.

En 1990, planeando un estándar internacional
Se comenzó con una vista general. El plan-
Ning se centró en conciliar el proceso del software
vistas de IEEE Std. 1074 y los Estados Unidos revisados
DoD estándar 2167A. La revisión fue eventu-
aliado publicado como DoD Std. 498. El internacional
Estándar se completó en 1995 con designación
ción, ISO / IEC 12207, y con el título de *Stan-*
dard para los procesos del ciclo de vida del software . Std. YO ASI/
IEC 12207 proporcionó un importante punto de partida
para el cuerpo de conocimiento capturado en este libro.

Fue la Junta de la IEEE Computer Society de
La aprobación de los gobernadores de la moción presentada
en mayo de 1993 por Fletcher Buckley que resultó
en la escritura de este libro. La asociación para
Aprobado por el Consejo de Maquinaria Informática (ACM)
una moción relacionada en agosto de 1993. Las dos mociones
condujo a un comité conjunto bajo el liderazgo de
Mario Barbacci y Stuart Zweben que sirvieron como
Sillas La declaración de la misión del comité conjunto
mittee fue "Para establecer los conjuntos apropiados
de criterios y normas para la práctica profesional de
ingeniería de software sobre la cual deci industrial
Siones, certificación profesional y educación
los planes de estudio pueden basarse ". El comité directivo
Grupos de trabajo organizados en las siguientes áreas:

1. Definir el cuerpo de conocimiento requerido y
   Prácticas recomendadas

Página 19

2. Definir ética y estándares profesionales.

Se espera que los lectores encuentren el uso de este libro.

3. Definir currículos educativos para estudiantes de pregrado
comió, gradué y educación continua.

Este libro proporciona el primer componente: requerido
conjunto de conocimientos y prácticas recomendadas.

El código de ética y práctica profesional.
para la ingeniería de software se completó en 1998
y aprobado por el Consejo ACM y el
IEEE Computer Society Junta de Gobernadores. Eso
ha sido adoptado por numerosas corporaciones y
otras organizaciones y está incluido en varias
Libros de texto recientes.

El currículum educativo para estudiantes universitarios.
se está completando con un esfuerzo conjunto del IEEE
Computer Society y la ACM y se espera
se completará en 2004.

Toda profesión se basa en un cuerpo de conocimiento.
borde y prácticas recomendadas, aunque
no siempre se definen de manera precisa. En
muchos casos, estos están formalmente documentados, usu-
aliado en una forma que les permita ser utilizados para
tales propósitos como la acreditación de programas académicos
gramos, desarrollo de educación y formación
programas, certificación de especialistas o profesiones
licenciamiento nacional. En general, una sociedad profesional.
u organismo relacionado mantiene la custodia de tal
mal definición En casos donde no exista tal formalidad
existe, el cuerpo de conocimiento y recomendado
las prácticas son "generalmente reconocidas" por la práctica
ners y pueden codificarse de varias maneras para
diferentes usos
fin en guiarlos hacia el conocimiento y
recursos que necesitan en su desarrollo profesional de por vida
Opción como profesionales de ingeniería de software.

El libro está dedicado a Fletcher Buckley en
reconocimiento de su compromiso con la promoción de
ingeniería de mercancías como disciplina profesional y
su excelencia como practicante de ingeniería de software
actuador en aplicaciones de radar.

Leonard L. Tripp, IEEE Fellow 2003
Presidente, Comité de Prácticas Profesionales, IEEE Computer Society (2001–2003)

Presidente, Sociedad Conjunta de Computación IEEE y ACM
Comité Directivo para el Establecimiento de
Ingeniería de software como profesión (1998–1999)

Presidente, Comité de Estándares de Ingeniería de Software,
IEEE Computer Society (1992–1998)

Página 20

EDITORES

Pierre Bourque, Departamento de Ingeniería de Software e Informática, École de technologie supérieure (ÉTS),
Canadá, pierre.bourque@etsmtl.ca
Richard E. (Dick) Fairley, Asociados de Ingeniería de Software y Sistemas (S2EA), EE. UU.,
dickfairley@gmail.com

COEDITORES

Alain Abran, Departamento de Ingeniería de Software y TI, École de technologie supérieure (ÉTS),

Canadá, alain.abran@etsmtl.ca
Juan Garbajosa, Universidad Politécnica de Madrid (Universidad Técnica de Madrid, UPM), España,
juan.garbajosa@upm.es
Gargi Keeni, Tata Consultancy Services, India, gargi@ieee.org
Beijun Shen, Escuela de Software, Universidad Shanghai Jiao Tong, China, bjshen@sjtu.edu.cn

EDITORES CONTRIBUYENTES

Las siguientes personas contribuyeron a editar la *Guía SWEBOK* V3:
Don Shafer
Linda Shafer
Mary Jane Willshire
Kate Guillemette

TABLERO DE CONTROL DE CAMBIOS

Las siguientes personas sirvieron en el tablero de control de cambios de *SWEBOK Guide* V3:
Pierre Bourque
Richard E. (Dick) Fairley, presidente
Dennis Frailey
Michael Gayle
Thomas Hilburn
Paul Joannou
James W. Moore
Don Shafer
Steve Tockey

EDITORES DE ÁREAS DE CONOCIMIENTO

Requisitos de Software

Gerald Kotonya, Escuela de Informática y Comunicaciones, Universidad de Lancaster, Reino Unido,
gerald@comp.lancs.ac.uk
Peter Sawyer, Facultad de Informática y Comunicaciones, Universidad de Lancaster, Reino Unido,
sawyer@comp.lancs.ac.uk

Diseño de software

Yanchun Sun, Escuela de Ingeniería Electrónica e Informática, Universidad de Pekín, China,
sunyc@pku.edu.cn

Construcción de software

Xin Peng, Escuela de Software, Universidad de Fudan, China, pengxin@fudan.edu.cn

Pruebas de software

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

Mantenimiento del software

Alain April, École de technologie supérieure (ÉTS), Canadá, alain.april@etsmtl.ca
Mira Kajko-Mattsson, Escuela de Tecnología de Información y Comunicación,
KTH Real Instituto de Tecnología, mekm2@kth.se

Gestión de configuración de software

Roger Champagne, École de technologie supérieure (ÉTS), Canadá, roger.champagne@etsmtl.ca

Alain April, École de technologie supérieure (ÉTS), Canadá, alain.april@etsmtl.ca

Gerencia de Ingeniería de Software

James McDonald, Departamento de Ciencias de la Computación e Ingeniería de Software,
Universidad de Monmouth, EE. UU., Jamesmc@monmouth.edu

Proceso de ingeniería de software

Annette Reilly, Lockheed Martin Information Systems & Global Solutions, Estados Unidos,
annette.reilly@computer.org
Richard E. Fairley, Asociados de Ingeniería de Software y Sistemas (S2EA), EE. UU.,
dickfairley@gmail.com

Modelos y métodos de ingeniería de software

Michael F. Siok, Lockheed Martin Aeronautics Company, EE. UU., Mike.f.siok@lmco.com

Calidad de software

J. David Blaine, EE. UU., Jdavidblaine@gmail.com
Durba Biswas, Tata Consultancy Services, India, durba.biswas@tcs.com

Página 22

Práctica profesional de ingeniería de software

Aura Sheffield, EE. UU., Arsheff@acm.org
Hengming Zou, Universidad Jiao Tong de Shanghai, China, zou@sjtu.edu.cn

Ingeniería de Software Economía

Christof Ebert, Vector Consulting Services, Alemania, christof.ebert@vector.com

Fundamentos de computación

Hengming Zou, Universidad Jiao Tong de Shanghai, China, zou@sjtu.edu.cn

Fundamentos matemáticos

Nabendu Chaki, Universidad de Calcuta, India, nabendu@ieee.org

Fundamentos de ingeniería

Amitava Bandyopadhayay, Instituto de Estadística de la India, India, bamitava@isical.ac.in
Mary Jane Willshire, Asociados de Ingeniería de Software y Sistemas (S2EA), EE. UU.,
mj.fairley@gmail.com

Apéndice B: Normas IEEE e ISO / IEC que admiten SWEBOK

James W. Moore, EE. UU., James.W.Moore@ieee.org

Página 23

EDITORES DE ÁREAS DE CONOCIMIENTO DE VERSIONES DE SWEBOK ANTERIORES

Las siguientes personas sirvieron como Editores Asociados para la versión de prueba publicada en 2001 o para La versión 2004.

Requisitos de Software

Peter Sawyer, Departamento de Informática, Universidad de Lancaster, Reino Unido.
Gerald Kotonya, Departamento de Informática, Universidad de Lancaster, Reino Unido.

Diseño de software

Guy Tremblay, Departamento de Información, UQAM, Canadá

Construcción de software

Steve McConnell, Software Construx, Estados Unidos
Terry Bollinger, la Corporación MITRE, EE. UU.
Philippe Gabrini, Departamento de Información, UQAM, Canadá
Louis Martin, Departamento de Informática, UQAM, Canadá

Pruebas de software

Antonia Bertolino, ISTI-CNR, Italia
Eda Marchetti, ISTI-CNR, Italia

Mantenimiento del software

Thomas M. Pigoski, Techsoft Inc., EE. UU.
Alain April, École de technologie supérieure, Canadá

Gestión de configuración de software

John A. Scott, Lawrence Livermore National Laboratory, Estados Unidos
David Nisse, Estados Unidos

Gerencia de Ingeniería de Software

Dennis Frailey, Raytheon Company, EE. UU.
Stephen G. MacDonell, Universidad Tecnológica de Auckland, Nueva Zelanda
Andrew R. Gray, Universidad de Otago, Nueva Zelanda

Proceso de ingeniería de software

Khaled El Emam, sirvió en el Consejo Nacional de Investigación de Canadá, Canadá

Herramientas y métodos de ingeniería de software

David Carrington, Escuela de Tecnología de la Información e Ingeniería Eléctrica,
La universidad de Queensland, Australia

Página 24

Calidad de software

Alain April, École de technologie supérieure, Canadá
Dolores Wallace, jubilada del Instituto Nacional de Estándares y Tecnología, EE. UU.
Larry Reeker, NIST, EE. UU.

Editor de referencias

Marc Bouisset, Departamento de Información, UQAM

Página 25

REVISAR EQUIPO

Las personas que figuran a continuación participaron en el proceso de revisión pública de *SWEBOK Guide* V3. Miembro-
El envío de la IEEE Computer Society no era un requisito para participar en este proceso de revisión, y
No se solicitó información de membresía a los revisores. Más de 1500 comentarios individuales fueron
recogido y debidamente adjudicado.

Carlos C. Amaro, Estados Unidos
Mark Ardis, Estados Unidos
Mora-Soto Arturo, España
Ohad Barzilay, Israel
Gianni Basaglia, Italia
Denis J. Bergquist, Estados Unidos
Alexander Bogush, Reino Unido
Christopher Bohn, Estados Unidos
Steve Bollweg, Estados Unidos
Reto Bonderer, Suiza
Alexei Botchkarev, Canadá
Pieter Botman, Canadá
Robert Bragner, Estados Unidos
Kevin Brune, Estados Unidos
Ogihara Bryan, Estados Unidos
Luigi Buglione, Italia
Rick Cagle, Estados Unidos
Barbara Canody, Estados Unidos
Rogerio A. Carvalho, Brasil
Daniel Cerys, Estados Unidos
Philippe Cohard, Francia
Ricardo Colomo-Palacios, España
Mauricio Coria, Argentina
Marek Cruz, Reino Unido
Stephen Danckert, Estados Unidos
Bipul K. Das, Canadá
James D. Davidson, Estados Unidos
Jon Dehn, Estados Unidos
Lincoln P. Djang, Estados Unidos
Andreas Doblander, Austria
Yi-Ben Doo, Estados Unidos
Scott J. Dougherty, Reino Unido
Regina DuBord, EE. UU.
Fedor Dzerzhinskiy, Rusia
Ann M. Eblen, Australia
David M. Endres, Estados Unidos
Marilyn Escue, Estados Unidos
Varuna Eswer, India
Istvan Fay, Hungría
José L.Fernández-Sánchez, España
Dennis J. Frailey, Estados Unidos
Tihana Galinac Grbac, Croacia
Colin Garlick, Nueva Zelanda
Garth JG Glynn, Reino Unido
Jill Gostin, Estados Unidos
Christiane Gresse von Wangenheim, Brasil
Thomas Gust, Estados Unidos
HN Mok, Singapur
Jon D. Hagar, Estados Unidos
Anees Ahmed Haidary, India
Duncan Hall, Nueva Zelanda
James Hart, Estados Unidos
Jens HJ Heidrich, Alemania
Rich Hilliard, Estados Unidos
Bob Hillier, Canadá
Norman M. Hines, Estados Unidos
Dave Hirst, Estados Unidos
Theresa L. Hunt, Estados Unidos
Kenneth Ingham, Estados Unidos
Masahiko Ishikawa, Japón
Michael A. Jablonski, Estados Unidos
G. Jagadeesh, India
Sebastian Justicia, España
Umut Kahramankaptan, Bélgica
Pankaj Kamthan, Canadá
Perry Kapadia, Estados Unidos
Tarig A. Khalid, Sudán
Michael KA Klaes, Alemania
Maged Koshty, Egipto
Claude C. Laporte, Canadá
Dong Li, China
Ben Linders, Países Bajos
Claire Lohr, Estados Unidos
Vladimir Mandic, Serbia
Matt Mansell, Nueva Zelanda
John Marien, Estados Unidos

Page 26

Stephen P. Masticola, Estados Unidos
Nancy Mead, Estados Unidos
Fuensanta Medina-Domínguez, España
Silvia Judith Meles, Argentina
Oscar A. Mondragon, México
David W. Mutschler, Estados Unidos
Maria Nelson, Brasil
John Noblin, Estados Unidos
Bryan G. Ogihara, Estados Unidos
Takehisa Okazaki, Japón
Hanna Oktaba, México
Chin Hwee Ong, Hong Kong
Venkateswar Oruganti, India
Birgit Penzenstadler, Alemania
Larry Peters, Estados Unidos
SK Pillai, India
Vaclav Rajlich, Estados Unidos
Kiron Rao, India
Luis Reyes, Estados Unidos
Hassan Reza, Estados Unidos
Steve Roach, Estados Unidos
Teresa L. Roberts, Estados Unidos
Dennis Robi, Estados Unidos
Warren E. Robinson, Estados Unidos
Jorge L. Rodriguez, Estados Unidos
Thom Schoeffling, Estados Unidos
Reinhard Schrage, Alemania
Neetu Sethia, India
Cindy C. Shelton, Estados Unidos
Alan Shepherd, Alemania
Katsutoshi Shintani, Japón
Erik Shreve, Estados Unidos
Jaguaraci Silva, Brasil
M. Somasundaram, India
Peraphon Sophatsathit, Tailandia
John Standen, Reino Unido
Joyce Statz, Estados Unidos
Perdita P. Stevens, Reino Unido
David Struble, Estados Unidos
Ohno Susumu, Japón
Urcun Tanik, EE.UU.
Talin Tasciyan, Estados Unidos
J. Barrie Thompson, Reino Unido
Steve Tockey, Estados Unidos
Miguel Eduardo Torres Moreno, Colombia
Dawid Trawczynski, EE. UU.
Adam Trendowicz, Alemania
Norio Ueno, Japón
Cenk Uyan, Turquía
Chandra Sekar Veerappan, Singapur

Alberto C. Sampaio, Portugal
Ed Samuels, Estados Unidos
Maria-Isabel Sanchez-Segura, España
Vineet Sawant, Estados Unidos
R. Schaaf, Estados Unidos
James C. Schatzman, Estados Unidos
Oscar A. Schivo, Argentina
Florian Schneider, Alemania
Oruganti Venkateswar, India
Jochen Vogt, Alemania
Hironori Washizaki, Japón
Ulf Westermann, Alemania
Don Wilson, Estados Unidos
Aharon Yadin, Israel
Hong Zhou, Reino Unido

EXPRESIONES DE GRATITUD

Financiación para el desarrollo de la *Guía SWEBOK* V3 ha sido proporcionado por la computadora IEEE Sociedad. Los editores y coeditores aprecian la importante trabajo realizado por los editores de KA y los editores contribuyentes así como también los miembros miembros de la Junta de Control de Cambios. La editorial el equipo también debe reconocer lo indispensable contribución de los revisores.

El equipo editorial también desea agradecer a los siguientes bajando a las personas que contribuyeron al proyecto en varias formas: Pieter Botman, Evan Butterfield, Carine Chauny, Pierce Gibbs, Diane Girard, John Keppler, Dorian McClenahan, Kenza Meridji, Samuel Redwine, Annette Reilly y Pam Thompson.

Finalmente, seguramente hay otras personas que tienen contribuido a esta *guía* , ya sea directamente o indi directamente, cuyos nombres hemos omitido inadvertidamente ted Para esas personas, ofrecemos nuestro aprecio tácito ciación y disculpas por haber omitido explícitamente reconocimiento.

PRESIDENTES DE LA SOCIEDAD DE COMPUTADORAS IEEE

Dejan Milojicic, Presidente 2014
David Alan Grier, Presidente 2013
Thomas Conte, presidente de 2015

JUNTA DE ACTIVIDADES PROFESIONALES, MEMBRESÍA 2013

Donald F. Shafer, presidente
Pieter Botman, PCSD
Pierre Bourque
Richard Fairley, PCSD
Dennis Frailey
S. Michael Gayle
Phillip Laplante, PCSD
Jim Moore, PCSD

Linda Shafer, PCSD
Steve Tockey, PCSD
Charlene "Chuck" Walrad

Página 28

MOCIONES SOBRE LA APROBACIÓN DE LA GUÍA SWEBOK V3.0

La *Guía SWEBOK* V3.0 fue sometida a votación por miembros verificados de la IEEE Computer Society en Noviembre de 2013 con la siguiente pregunta: "¿Aprueba este manuscrito de la *Guía SWEBOK?* V3.0 para avanzar al formato y publicación?

Los resultados de esta votación fueron 259 votos Sí y 5 No votos.

La siguiente moción fue adoptada por unanimidad por la Junta de Actividades Profesionales del Comité de IEEE. sociedad informática en diciembre de 2013:

La Junta de Actividades Profesionales de la IEEE Computer Society considera que la Guía de Soft- La *versión 3.0* del cuerpo de conocimiento de ingeniería de Ware *se ha completado con éxito; y respalda la* Guía del Cuerpo de Conocimientos de Ingeniería de Software *Versión 3.0 y la recomienda a Junta de Gobernadores de la IEEE Computer Society para su aprobación.*

La siguiente moción fue adoptada por la Junta de Gobernadores de la IEEE Computer Society en diciembre de 2013:

MOVIDO, que la Junta de Gobernadores de la IEEE Computer Society aprueba la Versión 3.0 de la Guía del Cuerpo de Conocimientos de Ingeniería del Software *y autoriza a la Presidencia de la Profes- Junta de Actividades Internacionales para proceder con la impresión.*

MOCIONES SOBRE LA APROBACIÓN DE LA GUÍA SWEBOK VERSIÓN 2004

La siguiente moción fue adoptada por unanimidad por la Junta Asesora Industrial de la *Guía SWEBOK* proyecto en febrero de 2004:

El Consejo Asesor Industrial considera que el proyecto de Cuerpo de Conocimientos de Ingeniería de Software tiado en 1998 se ha completado con éxito; y respalda la versión 2004 de la Guía de la SWEBOK *y lo elogia a la Junta de Gobernadores de la IEEE Computer Society para su aprobación.*

La siguiente moción fue adoptada por la Junta de Gobernadores de la IEEE Computer Society en febrero de 2004:

MOVIDO, que la Junta de Gobernadores de la IEEE Computer Society aprueba la Edición 2004 de la Guía del Cuerpo de Conocimientos de Ingeniería del Software *y autoriza a la Presidencia de la Profes- Comité Nacional de Prácticas para proceder con la impresión.*

Tenga en cuenta también que la edición de 2004 de la *Guía del conjunto de conocimientos de ingeniería de software* fue presentado por la IEEE Computer Society a ISO / IEC sin ningún cambio y fue reconocido como Informe técnico ISO / IEC TR 19759: 2005.

Página 29

INTRODUCCIÓN A LA GUÍA

KA	Área de conocimiento
SWEBOK	Cuerpo de Ingeniería de Software de Conocimiento

Publicación de la versión 2004 de esta *Guía para el Cuerpo de conocimiento de ingeniería de software* (SWEBOK 2004) fue un hito importante en el establecimiento de ingeniería de software como ingeniería reconocida disciplina. El objetivo en el desarrollo de esta actualización para SWEBOK es mejorar la moneda, la legibilidad, consistencia y usabilidad de la *Guía* .

Todas las áreas de conocimiento (KAs) han sido actualizadas para reflejar los cambios en la ingeniería de software desde publicación de SWEBOK 2004. Cuatro nuevas fundaciones dación KAs y Profesión de Ingeniería de Software Se han agregado las prácticas nacionales KA. Lo suave-Ware Herramientas y métodos de ingeniería KA tiene revisado como Modelos de ingeniería de software y métodos. Herramientas de ingeniería de software es ahora un tema en cada uno de los KAs. Tres apéndices provide las especificaciones para la descripción de KA, un conjunto anotado de estándares relevantes para cada KA, y una lista de las referencias citadas en la *Guía* .

Esta *guía* , escrita bajo los auspicios de la Junta de Actividades Profesionales del Comité IEEE puter Society, representa un próximo paso en la evolución ción de la profesión de ingeniería de software.

¿QUE ES LA INGENIERIA DE SOFTWARE?

Sistemas ISO / IEC / IEEE e Ingeniería de Software El vocabulario (SEVOCAB) define la ingeniería del softwaneering como "la aplicación de una disciplina sistemática enfoque sencillo y cuantificable del desarrollo, operación y mantenimiento de software; eso es el aplicación de ingeniería al software) ". [1]

¿CUÁLES SON LOS OBJETIVOS DE LA GUÍA DE SWEBOK?

La *guía* no debe confundirse con el cuerpo del Conocimiento mismo, que existe en el publicado literatura. El propósito de la *Guía* es describir la porción del Cuerpo de Conocimiento que es genaceptado por vía general, para organizar esa porción, y para Proporcionar acceso tópico a él.

La *guía para el cuerpo de ingeniería de software de conocimiento* (*Guía SWEBOK*) se estableció con los siguientes cinco objetivos:

1. Promover una visión coherente del software. ingeniería en todo el mundo
2. Especificar el alcance y aclarar el lugar. de ingeniería de software con respecto a otros disciplinas como informática, proyectos gestión de ect, ingeniería informática y matemáticas
3. Caracterizar los contenidos del software. disciplina de ingeniería
4. Para proporcionar un acceso tópico al Software Cuerpo de conocimiento de ingeniería
5. Para proporcionar una base para el plan de estudios desarrollo y para certificación individual y material de licencia

El primero de estos objetivos, un mundo consistente amplia vista de la ingeniería de software, fue compatible por un proceso de desarrollo que involucró aproximadamente Más de 150 revisores de 33 países. Más la información sobre el proceso de desarrollo puede se encuentra en el sitio web (www.swebok.org). Prosociedades profesionales y eruditas y agencias públicas involucrados en ingeniería de software fueron contactados, informado de este proyecto para actualizar SWEBOK, y invitado a participar en el proceso de revisión. KA ediTors fueron reclutados de América del Norte, el Pacífico Rim y Europa. Las presentaciones sobre el proyecto fueron hecho en varios lugares internacionales.

El segundo de los objetivos, el deseo de especificar el alcance de la ingeniería de software, motiValora la organización fundamental de la Guía. El material que se reconoce dentro esta disciplina está organizada en los quince KAs enumerados en la tabla I.1. Cada uno de estos KAs se trata en un capítulo en esta *guía* .

1 Ver www.computer.org/sevocab.

Página 30

Cuadro I.1. Los 15 KE SWEBOK

Requisitos de Software

Diseño de software

Construcción de software

Pruebas de software

Mantenimiento del software

ORGANIZACION JERARQUICA

La organización de los capítulos de KA apoya el tercero de los objetivos del proyecto, una caracterización ción de los contenidos de ingeniería de software. los especificaciones detalladas proporcionadas por el proyecto equipo editorial a los editores asociados con respecto a

Gestión de configuración de software
Gerencia de Ingeniería de Software
Proceso de ingeniería de software
Modelos y métodos de ingeniería de software
Calidad de software
Práctica profesional de ingeniería de software
Ingeniería de Software Economía
Fundamentos de computación
Fundamentos matemáticos
Fundamentos de ingeniería

Al especificar el alcance, también es importante identificar las disciplinas que se cruzan con el software Ingenieria. Con este fin, SWEBOK V3 también reconoce siete disciplinas relacionadas, enumeradas en la Tabla I.2 Los ingenieros de software deberían, por supuesto, tener conocimiento del material de estas disciplinas (y las descripciones de KA en esta *Guía* pueden hacer referencia a ellos). Sin embargo, no es un objective de la *Guía SWEBOK* para caracterizar el conocimiento de las disciplinas relacionadas.

Cuadro I.2. Disciplinas relacionadas

Ingeniería Informática
Ciencias de la Computación
Administración General
Matemáticas
Gestión de proyectos
Gestión de la calidad
Ingeniería de Sistemas

Los elementos relevantes de la informática. y las matemáticas se presentan en la informática Fundamentos y Fundamentos Matemáticos KAs de la *Guía* (Capítulos 13 y 14).

se puede encontrar el contenido de las descripciones de KA en el apéndice A.

La *guía* utiliza una organización jerárquica para descomponer cada KA en un conjunto de temas con rec Etiquetas identificables. Un nivel dos (a veces tres) desglose proporciona una manera razonable de encontrar temas de interés La *guía* trata a los seleccionados temas de manera compatible con las principales escuelas de pensamiento y con averías generalmente encontradas en la industria y en la literatura de ingeniería de software y normas. Los desgloses de temas no presumir dominios de aplicaciones particulares, negocios usos, filosofías de gestión, desarrollo métodos, y así sucesivamente. El alcance de cada tema la descripción es solo la necesaria para entender el naturaleza generalmente aceptada de los temas y para el lector para encontrar con éxito el material de referencia; el cuerpo del conocimiento se encuentra en la referencia materiales en sí, no en la *Guía* .

MATERIAL DE REFERENCIA Y MATRIZ

Para proporcionar acceso tópico al conocimiento, el cuarto de los objetivos del proyecto: la *Guía* identifica material de referencia autorizado para cada KA. El Apéndice C proporciona un Consolidado Lista de referencias para la *guía* . Cada KA incluye referencias relevantes de la referencia consolidada Lista y también incluye una matriz que relaciona material de referencia a los temas incluidos.

Cabe señalar que la *Guía* no intento de ser comprensivo en sus citas. Mucho material adecuado y excelente. no está referenciado Material incluido en la Con- La lista de referencia solidada proporciona cobertura de temas descritos

PROFUNDIDAD DE TRATAMIENTO

Para lograr el quinto objetivo de SWEBOK: propresentando una base para el desarrollo curricular,

certificación y licenciamiento, el criterio de *generación* se han aplicado conocimientos *aceptados por vía oral* , a distinguirse de avanzado e investigación conocimiento (por razones de madurez) y de conocimiento especializado (por motivos de Ality de aplicación).

El término equivalente *generalmente reconocido* proviene del Project Management Institute: "Generalmente reconocido significa el conocimiento y las prácticas descritas son aplicables a la mayoría proyecta la mayor parte del tiempo, y hay consenso sobre su valor y utilidad ". [2]

Sin embargo, los términos "generalmente aceptado" o "Generalmente reconocido" no implica que el des- el conocimiento identificado debe aplicarse uniformemente a todos los esfuerzos de ingeniería de software, cada proyecto Las necesidades de ect determinan eso, pero implica que ingenieros de software competentes y capaces deben estar equipado con este conocimiento para el potencial solicitud. Más precisamente, generalmente aceptado el conocimiento debe incluirse en el compañero de estudio

El desglose de temas en cada KA consti- Tutes el núcleo de la descripción de KA, que describe la descomposición del KA en subáreas, arriba- ics y subtemas. Para cada tema o subtema, un se da una breve descripción, junto con uno o más referencias

El material de referencia fue elegido porque es considerado como la mejor presentación de El conocimiento relativo al tema. Una matriz de enlaces Los temas del material de referencia.

La última parte de cada descripción de KA es la lista de referencias recomendadas y (opcionalmente) fur- Otras lecturas. Los estándares relevantes para cada KA son presentado en el Apéndice B de la *Guía* .

APÉNDICE A. DESCRIPCIÓN KA PRESUPUESTO

El apéndice A describe las especificaciones proporcionadas por el equipo editorial a los editores asociados para el contenido, referencias recomendadas, formato,

rial para el examen de licencia de ingeniería de software nación que los graduados tomarían después de ganar Cuatro años de experiencia laboral. Aunque este cri terion es específico para el estilo de educación y no necesariamente se aplica a otros países, nosotros lo considero útil

ESTRUCTURA DE LAS DESCRIPCIONES KA

Las descripciones de KA están estructuradas de la siguiente manera:

En la introducción, una breve definición del KA y una visión general de su alcance y de su relación Se presentan con otros KA.

2 *Una guía para el organismo de gestión de proyectos de Conocimiento* , 5ª ed., Project Management Institute, 2013; www.pmi.org .

y estilo de las descripciones de KA.

APÉNDICE B. ASIGNACIÓN DE STANDARDOS A KAS

El apéndice B es una lista anotada de los estándares, principalmente de IEEE e ISO, para cada uno de los KAs de la *Guía SWEBOK* .

APÉNDICE C. CONSOLIDADO LISTA DE REFERENCIA

El Apéndice C contiene la lista consolidada de referencias omitidas citadas en los KAs (estos las referencias están marcadas con un asterisco (*) en el texto).

Página 32

CAPÍTULO 1

REQUISITOS DE SOFTWARE

SIGLAS

CIA	Confidencialidad, Integridad y Disponibilidad
TROZO DE GATO	Gráfico Acíclico Dirigido
FSM	Medida de tamaño funcional
INGRESO	Consejo Internacional de Sistemas Ingenieria
UML	Lenguaje de modelado unificado
SysML	Lenguaje de modelado de sistemas

INTRODUCCIÓN

El área de conocimiento de Requisitos de software (KA) se ocupa de la obtención, análisis, especi ficación y validación de requisitos de software así como la gestión de requisitos durante ing todo el ciclo de vida del producto de software. Es ampliamente reconocido entre los investigadores. y profesionales de la industria que proyectan software son críticamente vulnerables cuando los requisitos Las actividades relacionadas están mal realizadas.

Los requisitos de software expresan las necesidades y restricciones impuestas a un producto de software que no implica, sin embargo, que un ingeniero de software No se pudo realizar la función.

Un riesgo inherente al desglose propuesto es que se puede inferir un proceso similar a una cascada. A protegerse de esto, tema 2, Proceso de requisitos, está diseñado para proporcionar una visión general de alto nivel de la proceso de requisitos estableciendo los recursos y restricciones bajo las cuales opera el proceso y qué acto configurarlo.

Una descomposición alternativa podría usar un producto estructura basada en uct (requisitos del sistema, software-requisitos de hardware, prototipos, casos de uso y pronto). El desglose basado en el proceso refleja el hecho de que los requisitos se procesan, si es para ser exitoso, debe ser considerado como un proceso implica actividades complejas y estrechamente vinculadas (tanto secuencial como concurrente), en lugar de como un actividad discreta y única realizada desde el principio de un proyecto de desarrollo de software.

Los requisitos de software KA están relacionados estrechamente al diseño de software, pruebas de software, Mantenimiento de software, configuración de software Gestión, Ingeniería de Software Gestión-ment, Proceso de Ingeniería de Software, Software Modelos y métodos de ingeniería y software Calidad KAs.

contribuir a la solución de algún mundo real problema.

El término "ingeniería de requisitos" es ampliamente usado en el campo para denotar el manejo sistemático de requisitos. Por razones de coherencia, el El término "ingeniería" no se utilizará en este KA que no sea para ingeniería de software per se.

Por la misma razón, "ingeniero de requisitos" un término que aparece en parte de la literatura, tampoco será utilizado. En cambio, el término "software ingeniero "o, en algunos casos específicos," requiere- especialista en mentos ", se utilizará este último donde el papel en cuestión generalmente lo realiza un individuo que no sea ingeniero de software. Esta

DESGLOSE DE TEMAS PARA REQUISITOS DE SOFTWARE

El desglose de temas para el Software Requisitos KA se muestra en la Figura 1.1.

1. Fundamentos de los requisitos de software
[1 *, c4, c4s1, c10s1, c10s4] [2 *, c1, c6, c12]

1.1. Definición de un requisito de software

En su forma más básica, un requisito de software es un propiedad que debe ser exhibida por algo en

Figura 1.1. Desglose de temas para los requisitos de software KA

Para resolver algún problema en el mundo real. Eso puede apuntar a automatizar parte de una tarea para alguien para apoyar los procesos de negocio de una organización ción, para corregir las deficiencias del software existente, o para controlar un dispositivo, por nombrar solo algunos de los muchos problemas para los cuales las soluciones de software son posible. Las formas en que los usuarios, las empresas pro Los procesos y las funciones de los dispositivos son típicamente complejos. Por extensión, por lo tanto, los requisitos en par- El software ticular suele ser una combinación compleja. ción de varias personas en diferentes niveles de un organización, y quienes están de una forma u otra involucrado o conectado con esta función desde el entorno en el que operará el software.

Una propiedad esencial de todo software requiere: es que sean verificables como individuos característica como un requisito funcional o en el nivel del sistema como requisito no funcional. Eso puede ser difícil o costoso verificar ciertas aplicaciones requisitos de loza. Por ejemplo, verificación del requisito de rendimiento en un centro de llamadas puede requerir el desarrollo de simulación software. Requisitos de software, prueba de software ing, y el personal de calidad debe asegurarse de que los requisitos se pueden verificar dentro de los disponibles limitaciones de recursos.

Los requisitos tienen otros atributos adicionales ción a las propiedades de comportamiento. Ejemplos comunes incluir una calificación de prioridad para permitir compensaciones en la cara de los recursos finitos y un valor de estado para permitir que el progreso del proyecto sea monitoreado. Typi- En términos generales, los requisitos de software son identificados de forma exclusiva fied para que puedan ser sometidos a software Gestión de la figuración durante todo el ciclo de vida. de la característica y del software.

1.2. Requisitos de producto y proceso

Un requisito de producto es una necesidad o restricción en el software a desarrollar (por ejemplo, "El el software deberá verificar que un estudiante cumpla con todos los requisitos previos requisitos antes de que él o ella se inscriba en un curso ").

Un requisito de proceso es esencialmente un restringir el desarrollo del software (para ejemplo, "El software se desarrollará utilizando un proceso RUP ").

Algunos requisitos de software generan implícitos requisitos del proceso La elección de la verificación

La técnica es un ejemplo. Otro podría ser el uso de técnicas de análisis particularmente rigurosas (como los métodos formales de especificación) para reducir fallas que pueden conducir a una confiabilidad inadecuada. Por los requisitos de cess también pueden imponerse directamente por la organización de desarrollo, su cliente, o un tercero como un regulador de seguridad.

1.3. Requisitos funcionales y no funcionales

Los requisitos *funcionales* describen las funciones. que el software debe ejecutarse; por ejemplo, para- enmarañar algún texto o modular una señal. Ellos a veces se conocen como capacidades o características. También se puede describir un requisito funcional como uno para el que puede ser un conjunto finito de pasos de prueba escrito para validar su comportamiento.

Los requisitos *no funcionales* son los que actuar para restringir la solución. No funcional los requisitos a veces se conocen como restricciones o requisitos de calidad. Pueden ser más clas- clasificado de acuerdo a si son rendimiento requisitos, requisitos de mantenibilidad, requisitos de seguridad, requisitos de confiabilidad, requisitos de seguridad, requiere interoperabilidad ments o uno de muchos otros tipos de software requisitos (ver Modelos y características de calidad- istics en el Software Quality KA).

1.4. Propiedades emergentes

Algunos requisitos representan propiedad emergente vínculos de software, es decir, requisitos que pueden no será abordado por un solo componente sino que dependerá de cómo todos los componentes del software interoperar El requisito de rendimiento para un centro de llamadas dependería, por ejemplo, de cómo el sistema telefónico, el sistema de información y todos los operadores interactuaron bajo operación real condiciones de ing. Las propiedades emergentes son cruciales depende de la arquitectura del sistema.

1.5. Requisitos cuantificables

Los requisitos de software deben establecerse claramente y lo más inequívocamente posible, y, donde apropiado, cuantitativamente. Es importante evitar requisitos vagos y no verificables que dependen para su interpretación de subjetivo juicio ("el software será confiable"; "el el software debe ser fácil de usar "). Esto es par- Particularmente importante para requisitos no funcionales ments. Dos ejemplos de requisitos cuantificados. son los siguientes: el software de un centro de llamadas debe aumentar el rendimiento del centro en un 20%; y un el sistema tendrá una probabilidad de generar un error fatal durante cualquier hora de operación de menos que $1 * 10^{-8}$. El requisito de rendimiento está en un nivel muy alto y deberá usarse para derivar Una serie de requisitos detallados. La fiabilidad Su requisito limitará estrictamente el sistema arquitectura.

1.6. Requisitos del sistema y software Requisitos

En este tema, "sistema" significa

> una combinación interactiva de elementos para lograr un objetivo definido. Estas incluye hardware, software, firmware, personas, información, técnicas, instalaciones, servicios y otros elementos de soporte,

según lo definido por el Consejo Internacional de Soft- Ingeniería de sistemas y artículos (INCOSE) [3].

Los requisitos del *sistema* son los requisitos para El sistema en su conjunto. En un sistema que contiene componentes de software, *los* requisitos de *software* son derivado de los requisitos del sistema.

Este KA define "requisitos de usuario" en un forma restringida, como los requisitos del sistema clientes o usuarios finales de tem. Sistema requerido Por el contrario, los requisitos abarcan los requisitos del usuario, requisitos de otras partes interesadas (como regu- autoridades lamentarias), y requisitos sin un fuente humana identificable.

2. Proceso de requisitos

[1 *, c4s4] [2 *, c1–4, c6, c22, c23]

Esta sección presenta los requisitos de software proceso, orientando los cinco temas restantes y mostrando cómo los requisitos procesan los detalles con el proceso general de ingeniería de software.

Página 35

2.1. Modelos de proceso

El objetivo de este tema es proporcionar un a menudo se necesita gente de marketing para establecer Lish lo que el mercado necesita y actuar como clientes proxy.

de pie que el proceso de requisitos

- no es una actividad de front-end discreta del soft- ciclo de vida del software, sino más bien un proceso iniciado al comienzo de un proyecto que continúa ser refinado a lo largo del ciclo de vida;
- identifica los requisitos de software como configu- racionar elementos y los gestiona utilizando el misma gestión de configuración de software prácticas como otros productos del software procesos del ciclo de vida;
- necesita adaptarse a la organización y contexto del proyecto

En particular, el tema trata de cómo las actividades de obtención, análisis, especificación ción y validación están configuradas para diferentes tipos de proyectos y limitaciones. El tema tambien incluye actividades que aportan información al proceso de requisitos, como marketing y fea- Estudios de posibilidad.

2.2. Actores de proceso

Este tema presenta los roles de las personas que participar en el proceso de requisitos. Este pro- cess es fundamentalmente interdisciplinario, y el especialista en requisitos necesita mediar entre el dominio de la parte interesada y el de soft- Ingeniería de artículos. A menudo hay muchas personas involucrado además del especialista en requisitos, cada de los cuales tiene una participación en el software. La estaca- los titulares variarán entre proyectos, pero siempre incluir usuarios / operadores y clientes (que necesitan no sea lo mismo)

Ejemplos típicos de partes interesadas del software incluye (pero no se limita a) lo siguiente:

- Usuarios: este grupo comprende aquellos que operar el software. A menudo es un heteroge- grupo neoso que involucra a personas con diferentes Roles y requisitos.
- Clientes: este grupo comprende aquellos que han encargado el software o quién representa resiente el mercado objetivo del software.
- Analistas de mercado: un producto de mercado masivo no tendrá un cliente de puesta en servicio, así que como la banca y el transporte público, son regu- Lated El software en estos dominios debe ser compatible cumplir con los requisitos de la normativa autoridades.
- Ingenieros de software: estos individuos tienen un interés legítimo en beneficiarse del desarrollo optando por el software, por ejemplo, reutilizando componentes en o de otros productos. Si, en este escenario, un cliente de un particular El producto lar tiene requisitos específicos que comprometer el potencial de componente reutilizar, los ingenieros de software deben cuidadosamente sopesar su propia apuesta contra las de los cliente. Requisitos específicos, particu- limitaciones importantes, pueden tener un gran impacto en costo del proyecto o entrega porque encajan bien o mal con el conjunto de habilidades del ingenieros Importantes compensaciones entre tales Se deben identificar los requisitos.

No será posible satisfacer perfectamente el requisitos de cada parte interesada, y es el El trabajo del ingeniero de software para negociar compensaciones que son aceptables para los principales interesados y dentro del presupuesto, técnico, regulatorio y Otras limitaciones. Un requisito previo para esto es que todos identificar a los interesados, la naturaleza de su "Estaca" analizada, y sus requisitos suscitados.

2.3. Apoyo y gestión de procesos

Esta sección presenta la gestión del proyecto. recursos requeridos y consumidos por los requisitos proceso de ments. Establece el contexto para el primer tema (Iniciación y definición del alcance) de la Gerencia de Ingeniería de Software KA. Su prin- El propósito principal es establecer el vínculo entre el pro- cess actividades identificadas en 2.1 y los problemas de costo, recursos humanos, capacitación y herramientas.

2.4. Calidad y mejora de procesos

Este tema se refiere a la evaluación de La calidad y la mejora de los requisitos. proceso. Su propósito es enfatizar el papel clave el proceso de requisitos juega en términos de la

Page 36

costo y oportunidad de un producto de software y de La satisfacción del cliente con él. Ayudará a orientar el proceso de requisitos con un estándar de calidad modelos de mejora de procesos y dards para soft- vajilla y sistemas. Procesar la calidad y mejorar ment está estrechamente relacionado tanto con el Software Proceso de ingeniería de software y KA de calidad KA, que comprende

- requisitos proceso cobertura por proceso estándares y modelos de mejora;
- medidas del proceso de requisitos y evaluación comparativa;
- planificación e implementación de mejoras;
- seguridad / mejora de la CIA / planificación y implementación.

para asegurar el negocio más importante del cliente Las necesidades se satisfacen primero. Esto minimiza el riesgo. de especialistas en requisitos que dedican tiempo a obtener requisitos de baja importancia, o aquellos que resultan no ser relevantes cuando Se entrega el software. Por otro lado, el la descripción debe ser escalable y extensible a aceptar requisitos adicionales no expresados en el primeras listas formales y compatibles con las anteriores los contemplados en métodos recursivos.

3.1. Fuentes de requisitos

Los requisitos tienen muchas fuentes en software blando típico Ware, y es esencial que todas las fuentes potenciales ser identificado y evaluado Este tema está diseñado para promover el conocimiento de las diversas fuentes de

3. Requisitos de obtención [1 *, c4s5] [2 *, c5, c6, c9]

La obtención de requisitos se refiere a la orígenes de los requisitos de software y cómo ingeniero de software puede recogerlos. Es el primero etapa en la construcción de una comprensión del problema Se requiere el software para resolver. Es fundamental cuenta una actividad humana y es donde la parte interesada Se identifican ers y se establecen relaciones. entre el equipo de desarrollo y el cliente. Se denomina "captura de requisitos" "Descubrimiento de requisitos" y "requisitos adquisición."

Uno de los principios fundamentales de un buen El proceso de obtención de requisitos es el de efecto Comunicación activa entre las distintas partes interesadas. titulares. Esta comunicación continúa a través de todo el ciclo de vida del desarrollo de software (SDLC) proceso con diferentes partes interesadas en diferentes puntos en el tiempo. Antes del desarrollo comienza, los especialistas en requisitos pueden formar el conducto para esta comunicación. Deben medi comió entre el dominio de los usuarios del software (y otras partes interesadas) y el mundo técnico de la ingeniero de software. Un conjunto de consistencia interna los modelos a diferentes niveles de abstracción facilitan comunicaciones entre usuarios de software / estaca titulares e ingenieros de software.

Un elemento crítico de la obtención de requisitos es informando el alcance del proyecto. Esto implica proporcionar ing una descripción del software que se especifica y su propósito y priorizar los entregables

requisitos de software y de los marcos para gestionándolos. Los principales puntos cubiertos son como sigue:

- Metas. El término "objetivo" (a veces llamado "Preocupación comercial" o "factor crítico de éxito" tor ") se refiere al objetivo general de alto nivel tives del software. Los objetivos proporcionan el motivo opción para el software, pero a menudo son vagamente formulado Los ingenieros de software deben pagar especial atención a evaluar el valor (en relación con la prioridad) y el costo de los objetivos. Una fe- El estudio de la posibilidad es una forma relativamente económica de haciendo esto.
- Conocimiento del dominio. El ingeniero de software necesita adquirir o tener conocimiento disponible ventaja sobre el dominio de la aplicación. Dominio el conocimiento proporciona el trasfondo que todos los requisitos requeridos conocimiento debe establecerse para entenderlo. Sus una buena práctica para emular un ontológico enfoque en el dominio del conocimiento. Rela- relaciones entre conceptos relevantes dentro del Se debe identificar el dominio de la aplicación.
- Grupos de interés (ver sección 2.2, Proceso Actores). Gran parte del software ha resultado insaturado. isfactory porque ha enfatizado la exigencia mentores de un grupo de partes interesadas en el expensas de otros. Por lo tanto, el entregado el software es difícil de usar o subvierte el estructuras culturales o políticas del cliente Organización Tomer. El ingeniero de software necesita identificar, representar y administrar

Page 37

los "puntos de vista" de muchos tipos diferentes de partes interesadas

- Reglas del negocio. Estas son declaraciones que definir o restringir algún aspecto de la estructura ture o el comportamiento del negocio en sí. "UNA el estudiante no puede registrarse en el próximo semestre cursos si queda alguna matrícula impaga honorarios "sería un ejemplo de una regla comercial eso sería una fuente de requisitos para una unidad software de registro de cursos de versity.
- El entorno operativo. Requisitos se derivará del medio ambiente en que se ejecutará el software Estas pueden ser, por ejemplo, restricciones de tiempo en software en tiempo real o en control de rendimiento tensiones en un entorno empresarial. Estas debe buscarse activamente porque pueden afectar en gran medida la viabilidad del software y el costo como así como restringir las opciones de diseño.
- El ambiente organizacional. Software a menudo se requiere para apoyar un negocio pro cess, cuya selección puede ser condi- mencionado por la estructura, cultura e interna política de la organización. El software el ingeniero debe ser sensible a estos ya que, en general, el nuevo software no debe forzar cambio no planificado en el proceso comercial.

3.2. Técnicas de obtención

Una vez que se hayan identificado las fuentes de requisitos aún no se ha obtenido de los usuarios finales. La impor- Tancia de planificación, verificación y validación en la obtención de requisitos no puede ser exagerada. UNA Existen varias técnicas para los requisitos tation los principales son estos:

- Entrevistas. Entrevistar a los interesados es un Medios "tradicionales" para obtener requisitos. Es importante entender las ventajas. y limitaciones de las entrevistas y cómo debe llevarse a cabo.
- Escenarios. Los escenarios proporcionan un valioso medios para proporcionar contexto a la elicitación ción de los requisitos del usuario. Permiten el ingeniero de software para proporcionar un marco para preguntas sobre las tareas del usuario al permitir Preguntas de "qué pasa si" y "cómo se hace esto" Se preguntado. El tipo más común de escenografía. nario es la descripción del caso de uso. Hay un enlace aquí al tema 4.2 (Modelado conceptual) porque las anotaciones de escenarios como el caso de uso Los diagramas son comunes en el software de modelado.
- Prototipos. Esta técnica es una herramienta valiosa. para aclarar requisitos ambiguos. Ellos puede actuar de manera similar a los escenarios por pro Viding usuarios con un contexto dentro del cual pueden entender mejor qué información Necesito proporcionar. Hay una amplia gama de técnicas de creación de prototipos, a partir de simulacros de papel ups de diseños de pantalla a versiones de prueba beta de productos de software y una fuerte superposición de

tified, el ingeniero de software puede comenzar a obtener requisitos de información de ellos. Tenga en cuenta que los requisitos rara vez se obtienen listos para usar. Por el contrario, el ingeniero de software obtiene información a partir del cual él o ella formula los requisitos. Este tema se concentra en técnicas para obtener interesados humanos para articular los requisitos informacion relevante. Es una tarea muy difícil y el ingeniero de software necesita ser sensibilizado a hecho de que (por ejemplo) los usuarios pueden tener dificultades describiendo sus tareas, puede dejar información importante mación no declarada, o puede no estar dispuesta o no puede cooperar. Es particularmente importante entender que la provocación no es una actividad pasiva y que, incluso si las partes interesadas cooperativas y articuladas son disponible, el ingeniero de software tiene que trabajar duro para obtener la información correcta. Muchos negocios o los requisitos técnicos son tácitos o en comentarios que

sus usos separados para los requisitos elicitación y para la validación de requisitos (ver sección 6.2, creación de prototipos). Protocolo de baja fidelidad a menudo se prefieren los tipos para evitar a los interesados "Anclaje" en caracteres menores e incidentales estadísticas de un prototipo de mayor calidad que puede Limite la flexibilidad de diseño de formas no deseadas.

- Reuniones facilitadas. El propósito de estos reuniones es tratar de lograr un resumen efecto, mediante el cual un grupo de personas puede traer más información sobre su software requiere Ments que trabajando individualmente. Ellos puede hacer una lluvia de ideas y refinar ideas que pueden ser difícil de sacar a la superficie usando inter puntos de vista. Otra ventaja es esa conflictiva los requisitos surgen desde el principio de una manera que permite a las partes interesadas reconocer dónde estas ocurrir. Cuando funciona bien, esta técnica

puede dar lugar a una más rica y más consistente conjunto de requisitos de lo contrario ser alcanzable Sin embargo, las reuniones deben manejarse con cuidado (de ahí la necesidad de un facilitador) para evitar una situación en la que las habilidades críticas del equipo se erosionan por lealtad grupal, o en qué requisitos reflejando las preocupaciones de unos pocos francos (y quizás personas mayores) que son favorecidas en detrimento de los demás.

- Observación. La importancia del software. contexto dentro del entorno organizacional ment ha llevado a la adaptación de la observación técnicas nacionales como la etnografía para obtención de requisitos. Ingenieros de software aprenda sobre las tareas de los usuarios sumergiéndolas en el medio ambiente y observando cómo los usuarios realizan sus tareas interactuando con entre sí y con herramientas de software y otros recursos Estas técnicas son relativamente caro pero también instructivo porque ilustrar que muchas tareas de usuario y negocios los procesos son demasiado sutiles y complejos para sus actores para describir fácilmente.
- Historias de usuarios. Esta técnica es comúnmente utilizado en métodos adaptativos (ver método ágil- ods en los modelos de ingeniería de software y Métodos KA) y se refiere a corto, alto descripciones de nivel de funcionalidad requerida expresado en términos del cliente. Un usuario típico la historia tiene la forma: *"Como <role>, quiero <objetivo / deseo> para que <beneficio> ".* Un usuario la historia pretende contener la información suficiente mation para que los desarrolladores puedan producir un estimación razonable del esfuerzo para implementar Míralo. El objetivo es evitar algunos de los residuos. eso sucede a menudo en proyectos donde se detalla los requisitos se recopilan temprano pero se convierten inválido antes de que comience el trabajo. Ante un usuario se implementa la historia, una aceptación apropiada El procedimiento debe ser escrito por el cliente Tomer para determinar si los objetivos de la Se ha cumplido la historia del usuario.
- Otras técnicas. Una gama de otras técnicas. para apoyar la obtención de requisitos

4. Análisis de requisitos

[1 *, c4s1, c4s5, c10s4, c12s5]
[2 *, c7, c11, c12, c17]

Este tema se refiere al proceso de ana- requisitos de lyzing para

- detectar y resolver conflictos entre requisitos;
- descubrir los límites del software y cómo debe interactuar con su organización y entorno operativo;
- elaborar requisitos del sistema para derivar soft- requisitos de loza.

La visión tradicional del análisis de requisitos. ha sido que se reduzca a modelo conceptual usando uno de varios métodos de análisis, como el método de análisis estructurado. Mientras el modelado conceptual es importante, incluimos el clasificación de requisitos para ayudar a informar a los comerciantes Eoffs entre requisitos (requisitos clas- sificación) y el proceso de establecer estos compensaciones (negociación de requisitos).

Se debe tener cuidado al describir los requisitos. con la suficiente precisión como para permitir los requisitos para ser validado, su implementación para ser verificada, y sus costos a estimar.

4.1. Clasificación de requisitos

Los requisitos se pueden clasificar en una serie de dimensiones. Los ejemplos incluyen lo siguiente:

- Si el requisito es funcional o no funcional (ver sección 1.3, Funcional y requisitos no funcionales).
- Si el requisito se deriva de uno o más requisitos de alto nivel o una emergencia propiedad gent (ver sección 1.4, Emergente Propiedades), o se está imponiendo directamente en el software por parte de un interesado u otro fuente.
- Si el requisito está en el producto o el proceso (ver sección 1.2, Producto y

la información existe y varía desde el análisis
productos de la competencia para aplicar datos min-
Técnicas de uso de fuentes de dominio.
conocimiento o bases de datos de solicitud del cliente.

Requisitos del proceso). Requisitos sobre el
proceso puede restringir la elección de contrac-
tor, el proceso de ingeniería de software para ser
adoptado, o las normas a cumplir.

Página 39

- El requisito de prioridad. Cuanto mayor sea el precio
  oridad, cuanto más esencial es el requisito
  para cumplir los objetivos generales del software.
  A menudo se clasifica en una escala de punto fijo como
  como obligatorio, altamente deseable, deseable,
  u opcional, la prioridad a menudo tiene que ser equilibrada
  frente al costo de desarrollo y
  implementación.
- El alcance del requisito. Alcance se refiere
  en la medida en que un requisito afecta
  El software y sus componentes.
  Algunos requisitos, particularmente ciertos
  no funcionales, tienen un alcance global en
  que su satisfacción no puede asignarse a
  Un componente discreto. Por lo tanto, un requisito
  con alcance global puede afectar fuertemente el
  arquitectura de software y el diseño de muchos
  componentes, mientras que uno con un estrecho
  El alcance puede ofrecer una serie de opciones de diseño
  y tienen poco impacto en la satisfacción de
  otros requerimientos.
- Volatilidad / estabilidad. Algunos requisitos
  cambiar durante el ciclo de vida del soft-
  Ware, e incluso durante el desarrollo
  proceso en sí mismo. Es útil si alguna estimación
  de la probabilidad de que un requisito
  Se puede hacer un cambio. Por ejemplo, en un banco
  aplicación ing, requisitos para funciones
  calcular y acreditar intereses a los clientes
  es probable que las cuentas sean más estables que una
  requisito para soportar un tipo particular de
  cuenta libre de impuestos El primero refleja una diversión.
  característica fundamental del dominio bancario (que
  las cuentas pueden ganar intereses), mientras que este último
  puede quedar obsoleto por un cambio en
  legislación del gobierno. Marcar potencialmente
  los requisitos volátiles pueden ayudar al software
  ingeniero establecer un diseño que sea más tolerante
  hormiga de cambio.

Otras clasificaciones pueden ser apropiadas,
dependiendo de la práctica normal de la organización
tice y la aplicación en sí.

Existe una fuerte superposición entre los requisitos
atributos de clasificación y requisitos (ver
sección 7.3, Atributos de requisitos).

4.2. Modelado conceptual

El desarrollo de modelos de un mundo real.
problema es clave para el análisis de requisitos de software
hermanita Su propósito es ayudar a comprender el
situación en la que ocurre el problema, así como
representando una solución. Por lo tanto, modelos conceptuales.
comprender modelos de entidades del problema
dominio, configurado para reflejar su mundo real
relaciones y dependencias Este tema es
estrechamente relacionado con el Software Engineering Mod-
Els y Métodos KA.

Se pueden desarrollar varios tipos de modelos.
Estos incluyen diagramas de casos de uso, mod de flujo de datos
els, modelos de estado, modelos basados en objetivos, interfaz de usuario
acciones, modelos de objetos, modelos de datos y muchos
otros. Muchas de estas anotaciones de modelado son parte
del *lenguaje de modelado unificado (UML).* Utilizar
los diagramas de casos, por ejemplo, se usan de manera rutinaria
para representar escenarios donde el límite se separa
los actores (usuarios o sistemas en el entorno externo)
ronment) del comportamiento interno donde cada
El caso de uso representa una funcionalidad del sistema.

Los factores que influyen en la elección del modelo
La notación ing incluye estos:

- La naturaleza del problema. Algunos tipos de
  el software exige que ciertos aspectos se analicen
  lisado de forma particularmente rigurosa. Por ejemplo,
  modelos paramétricos y de estado, que son parte
  de SysML [4], es probable que sean más importantes
  Tant para software en tiempo real que para información
  sistemas de iones, mientras que generalmente sería el
  opuesto para objetos y modelos de actividad.
- La experiencia del ingeniero de software. Es
  a menudo más productivo para adoptar un modelo
  notación o método con el que el software
  El ingeniero tiene experiencia.
- Los requisitos de proceso del cliente.
  (ver sección 1.2, Producto y proceso
  Requisitos). Los clientes pueden imponer sus
  notación o método preferido o prohibir cualquier
  con el que no están familiarizados. Este factor
  Puede entrar en conflicto con el factor anterior.

Tenga en cuenta que, en casi todos los casos, es útil comenzar
construyendo un modelo del contexto del software. los
El contexto de software proporciona una conexión entre
El software previsto y su entorno externo.

Page 40

Esto es crucial para comprender el contenido del software. texto en su entorno operativo e identificar- ing sus interfaces con el medio ambiente.

Este subtema no busca "enseñar" un particular estilo de modelado lar o notación, sino que más bien proporcionar orientación sobre el propósito y la intención de modelar.

4.3. Diseño arquitectónico y requisitos Asignación

En algún momento, la arquitectura de la solución debe se deriva. El diseño arquitectónico es el punto en con el cual se superpone el proceso de requisitos diseño de software o sistemas e ilustra cómo imposible es desacoplar limpiamente las dos tareas. Este tema está estrechamente relacionado con la estructura del sistema y Arquitectura en el Diseño de Software KA. En En muchos casos, el ingeniero de software actúa como soft- arquitecto de mercancías porque el proceso de análisis y elaborar los requisitos exige que los componentes de arquitectura / diseño que serán responsable de satisfacer los requisitos ser identificado. Esta es la asignación de requisitos: la asignación a componentes de arquitectura respon- Sible para satisfacer los requisitos.

La asignación es importante para permitir un análisis detallado análisis de requisitos. Por lo tanto, por ejemplo, una vez conjunto de requisitos ha sido asignado a una empresa ponent, los requisitos individuales pueden ser más analizado para descubrir más requisitos sobre cómo el componente necesita interactuar con otros ponentes para satisfacer los requisitos asignados ments. En proyectos grandes, la asignación estimula un nueva ronda de análisis para cada subsistema. Como un ejemplo, requisitos para un frenado particular rendimiento para un automóvil (distancia de frenado, seguridad en malas condiciones de manejo, suavidad de aplicación es posible que se requiera presión de pedal, etc.) asignado al hardware de frenado (mecánico y conjuntos hidráulicos) y un freno antibloqueo sistema (ABS). Solo cuando un requisito para un sistema de frenos antibloqueo ha sido identificado, y los requisitos que se le asignan, ¿puede la capacidad lazos del ABS, el hardware de frenado y las emergencias propiedades gentiles (como el peso del automóvil) se utilizarán para Identificar los requisitos detallados del software ABS.

El diseño arquitectónico se identifica estrechamente con modelado conceptual (ver sección 4.2, Conceptual Modelado).

4.4. Negociación de requisitos

Otro término comúnmente usado para este subtema es "resolución de conflictos". Esto se refiere a la resolución Problemas con requisitos donde los conflictos ocurrir entre dos partes interesadas que requieren mutuo características incompatibles aliado, entre requisitos y recursos, o entre funcional y no requisitos funcionales, por ejemplo. En la mayoría casos, no es aconsejable que el ingeniero de software tomar una decisión unilateral, por lo que se hace necesario sary consultar con las partes interesadas para llegar a un consenso sobre una compensación adecuada. Es a menudo importante, por razones contractuales, que tal decisión Se pueden rastrear las siones hasta el cliente. Tenemos clasificó esto como un análisis de requisitos de software este es el tema porque surgen problemas como resultado de análisis Sin embargo, un caso fuerte también puede ser hecho para considerarlo una validación de requisitos tema (ver tema 6, Validación de requisitos).

La priorización de requisitos es necesaria, no solo como un medio para filtrar requisitos importantes, pero también para resolver conflictos y planificar entregas por etapas, lo que significa hacer complejas decisiones que requieren un conocimiento detallado del dominio Habilidades de estimación. Sin embargo, a menudo es difícil obtener información real que pueda actuar como una base para tales decisiones. Además, requieren a menudo dependen unos de otros, y priori los lazos son relativos. En la práctica, ingenieros de software. realizar la priorización de requisitos con frecuencia sin conocer todos los requisitos.

La priorización de requisitos puede seguir un costo enfoque de valor que implica un análisis de Los interesados definen en escala los beneficios del valor agregado que la implementación ción del requisito los trae, frente a la sanciones por no haber implementado un particular requisito. También implica un análisis de los ingenieros de software que estiman en escala costo de implementar cada requisito, relativo a otros requisitos Otros requisitos pri- enfoque de oritización llamado jerarquía analítica el proceso implica comparar todos los pares únicos de requisitos para determinar cuál de los dos es de mayor prioridad, y en qué medida.

4.5. Análisis formal

El análisis formal se refi solo al tema 4 , sino a también secciones 5.3 y te tema también está relacionado a los métodos formales (geniería de software Área de conocimiento de modelos y métodos.

El análisis formal ha tenido un impacto en algunos dominios de aplicación, particularmente aquellos de alta sistemas de integridad. La expresión formal de los requisitos requieren un idioma con formalmente Semántica definida. El uso de un análisis formal. para requisitos de expresión tiene dos beneficios. Primero, habilita los requisitos expresados en el lenguaje a especificar con precisión y sin ambigüedades un documento que puede ser revisado sistemáticamente evaluado y aprobado. Para sistemas complejos, particularmente aquellos que involucran sustanciales no suaves componentes de mercancías, hasta tres diferentes Se producen tipos de documentos: definición del sistema ción, requisitos del sistema y requisitos de software ments. Para productos de software simples, solo el Se requiere un tercio de estos. Los tres documentos son descrito aquí, con el entendimiento de que se puede combinar según corresponda. Una descripción de La ingeniería de sistemas se puede encontrar en Disciplinas del capítulo de Ingeniería del Software de esta *Guía* .

demasiado, evitando así (en principio) el potencial por mala interpretación. En segundo lugar, los requisitos pueden ser razonado, permitiendo las propiedades deseadas del software especificado que se probará. Formal el razonamiento requiere soporte de herramientas para ser práctico para cualquier cosa que no sean sistemas triviales y herramientas generalmente se dividen en dos tipos: demostradores de teoremas Damas modelo. En ningún caso la prueba puede ser completa automatizado, y el nivel de competencia en formal el razonamiento necesario para usar las herramientas restringe La aplicación más amplia del análisis formal.

La mayoría del análisis formal se centra en relativamente Etapas tardías del análisis de requisitos. Es genérico aliado contraproducente para aplicar formalización hasta los objetivos comerciales y los requisitos del usuario han llegado a un enfoque nítido a través de medios tales como los descritos en otra parte de la sección 4. Cómo: alguna vez, una vez que los requisitos se hayan estabilizado y han sido elaborados para especificar el concreto apropiado vínculos del software, puede ser beneficioso for- Malizar al menos los requisitos críticos. Este per- Mits validación estática que el software especificado por los requisitos tiene de hecho la adecuada lazos (por ejemplo, ausencia de punto muerto) que el cliente, usuarios e ingeniero de software lo esperan tener.

5. Especificación de requisitos

[1 *, c4s2, c4s3, c12s2–5] [2 *, c10]

Para la mayoría de las profesiones de ingeniería, el término "especificación" se refiere a la asignación de números valores o límites a los objetivos de diseño de un producto. En ingeniería de software, "requisitos de software especificación "se refiere típicamente a la producción de

5.1. Documento de definición del sistema

Este documento (a veces conocido como el usuario documento de requisitos o concepto de operaciones documento) registra los requisitos del sistema. Eso define los requisitos del sistema de alto nivel de La perspectiva del dominio. Sus lectores incluyen representantes de los usuarios / clientes del sistema (el marketing puede desempeñar estos roles para el mercado software controlado), por lo que su contenido debe estar redactado en términos del dominio. El documento enumera los requisitos del sistema junto con antecedentes información sobre los objetivos generales para el sistema, su entorno objetivo y una declaración de las restricciones, supuestos y no funcionales requisitos Puede incluir modelos conceptuales. diseñado para ilustrar el contexto del sistema, el uso escenarios, y las principales entidades de dominio, como así como flujos de trabajo.

5.2. Especificación de requisitos del sistema

Desarrolladores de sistemas con software sustancial. y componentes que no son de software, un aire moderno revestimiento, por ejemplo, a menudo separa la descripción ción de los requisitos del sistema a partir de la descripción de requisitos de software. En esta vista, el sistema se especifican los requisitos, el software requiere los derivados se derivan de los requisitos del sistema, y luego los requisitos para el software de comp se especifican los componentes. Estrictamente hablando, sistema la especificación de requisitos es un ingeniero de sistemas ing actividad y queda fuera del alcance de esta *Guíe* .

Page 42

5.3. Especificación de Requerimientos de Software

La especificación de requisitos de software establece la base del acuerdo entre clientes y contratistas o proveedores (en proyectos impulsados por el mercado De hecho, estos roles pueden ser desempeñados por el departamento de marketing y divisiones de desarrollo) sobre lo que el software el producto es tan bueno como lo que no se espera que hacer.

Permisos de especificación de requisitos de software Una evaluación rigurosa de los requisitos antes el diseño puede comenzar y reduce el rediseño posterior. Esto también debe proporcionar una base realista para la estimación ing costos de productos, riesgos y horarios.

Las organizaciones también pueden usar un software requerido Ment documento de especificación como la base para desarrollando una verificación y validación efectivas planes

La especificación de requisitos de software proporciona una base informada para transferir un producto de software uct a nuevos usuarios o plataformas de software. Finalmente puede proporcionar una base para la mejora del software.

Los requisitos de software a menudo se escriben en lenguaje natural, pero, en requisitos de software especificación, esto puede complementarse con for- Descripciones mal o semiformales. Selección de las anotaciones apropiadas permiten requisitos particulares aspectos y aspectos de la arquitectura de software para

6. Validación de requisitos

[1 *, c4s6] [2 *, c13, c15]

Los documentos de requisitos pueden estar sujetos a validez. procedimientos de ida y verificación. El requerimiento Los elementos de un documento pueden validarse para garantizar que el software el ingeniero ha entendido los requisitos; está También es importante verificar que un requisito docu- ment se ajusta a los estándares de la compañía y que Es comprensible, consistente y completo. En casos donde se documentaron estándares de la compañía o la terminología es inconsistente con la ampliamente aceptada estándares, un mapeo entre los dos debe ser acordado y anexado al documento.

Las anotaciones formales ofrecen la ventaja importante de permitir que se prueben las dos últimas propiedades (en un sentido restringido, al menos). Estaca diferente titulares, incluidos representantes del cliente y desarrollador, debe revisar los documentos. Los documentos de requisitos están sujetos a los mismos prácticas de gestión de configuración como el otro entregables de los procesos del ciclo de vida del software. Cuando sea práctico, los requisitos individuales son también sujeto a la gestión de la configuración, gener- Ally utilizando una herramienta de gestión de requisitos (ver tema 8, Herramientas de requisitos de software).

Es normal programar explícitamente uno o más puntos en el proceso de requisitos donde el

ser descrito de manera más precisa y concisa que
lenguaje natural. La regla general es que nota-
se deben usar las opciones que permitan los requisitos
para ser descrito con la mayor precisión posible. Esto es
particularmente crucial para la seguridad crítica, reguladora,
y ciertos otros tipos de software confiable.
Sin embargo, la elección de la notación a menudo es
tensa por la capacitación, habilidades y preferencias de
Los autores y lectores del documento.

Varios indicadores de calidad han sido
desarrollado que se puede utilizar para relacionar la calidad
de especificación de requisitos de software a otro
variables del proyecto como costo, aceptación, por-
formance, horario y reproducibilidad. Calidad
indicadores para requisitos de software individuales
las declaraciones de especificación incluyen imperativos,
directivas, frases débiles, opciones y continuidad
ances. Los indicadores para todo el software requieren:
El documento de especificaciones incluye tamaño, lectura
capacidad, especificación, profundidad y estructura del texto.

Los requisitos están validados. El objetivo es recoger
cualquier problema antes de que los recursos se comprometan a
abordando los requisitos. Requisitos vali-
la fecha se refiere al proceso de examen
ing el documento de requisitos para asegurar que
define el software correcto (es decir, el software
que los usuarios esperan).

6.1. Revisiones de requisitos

Quizás el medio más común de validación
es por inspección o revisión de los requisitos
documentos). Se asigna un grupo de revisores.
un resumen para buscar errores, suposiciones erróneas,
falta de claridad y desviación de la práctica estándar
tice La composición del grupo que dirige
la revisión es importante (al menos un representante
Se debe incluir una declaración del cliente por un
proyecto orientado al cliente, por ejemplo), y puede
ayuda para proporcionar orientación sobre qué buscar en
La forma de listas de verificación.

Las revisiones pueden constituirse al finalizar
el documento de definición del sistema, la especificación de sistemas
documento de identificación, los requisitos de software
documento de especificación, la especificación de referencia
ción para una nueva versión, o en cualquier otro paso en el
proceso.

6.2. Prototipos

La creación de prototipos es comúnmente un medio para validar
la interpretación del ingeniero de software del software
requisitos de software, así como para obtener nuevos
requisitos Al igual que con la obtención, hay un rango
de técnicas de creación de prototipos y varios puntos
en el proceso donde la validación del prototipo puede
ser apropiado La ventaja de los prototipos es
que pueden facilitar la interpretación del soft-
supuestos del ingeniero de mercancías y, cuando sea necesario,
dar comentarios útiles sobre por qué están equivocados. por
ejemplo, el comportamiento dinámico de un usuario inter-
la cara se puede entender mejor a través de un
prototipo acoplado que a través de una descripción textual
o modelos gráficos. La volatilidad de un requerimiento
Ment que se define después de la creación de prototipos
hecho es extremadamente bajo porque hay acuerdo
entre el interesado y el motor del software
neer, por lo tanto, para la seguridad crítica y crucial
características de creación de prototipos realmente ayudaría.
También desventajas, sin embargo. Estos incluyen el
peligro de distracción de la atención de los usuarios
la funcionalidad subyacente central por cosmética
problemas o problemas de calidad con el prototipo. por
Por esta razón, algunos abogan por prototipos que eviten
software, como maquetas basadas en rotafolios. Pro-
Los tipos pueden ser costosos de desarrollar. Sin embargo, si
evitan el desperdicio de recursos causado por
tratando de satisfacer requisitos erróneos, su
El costo puede justificarse más fácilmente. Protocolo tempranero
Los tipos pueden contener aspectos de la solución final.
Los prototipos pueden ser evolutivos en lugar de
tirar a la basura.

dominio, intercambio de datos. Si el análisis formal no
sistemas opciones, es posible usar razones formales
ing para probar las propiedades de especificación. Este tema es
estrechamente relacionado con el Software Engineering Mod-
Els y Métodos KA.

6.4. Prueba de aceptacion

Una propiedad esencial de un requisito de software.
isa que debería ser posible validar que el
El producto terminado lo satisface. Requisitos que
no pueden ser validados son realmente solo "deseos".
Por lo tanto, una tarea importante es planificar cómo ver
ify cada requisito En la mayoría de los casos, diseñar
las pruebas de aceptación hacen esto para saber cómo los usuarios finales escriben
Realice negocios con el sistema.

Identificación y diseño de pruebas de aceptación.
puede ser difícil para requisitos no funcionales
(ver sección 1.3, Funcional y no funcional
Requisitos). Para ser validados, primero deben
ser analizado y descompuesto hasta el punto donde
Se pueden expresar cuantitativamente.

Se puede encontrar información adicional en Aceptar
tance / Calificación / Pruebas de conformidad en el
Pruebas de software KA.

7. Consideraciones prácticas

Existen [1 *, c4s1, c4s4, c4s6, c4s7]
[2 *, c3, c12, c14, c16, c18–21]

El primer nivel de descomposición del tema pre
presentado en este KA puede parecer describir un lineal
secuencia de actividades. Esta es una vista simplificada
del proceso

El proceso de requisitos abarca todo
ciclo de vida del software. La gestión del cambio y la
mantenimiento de los requisitos en un estado que
refleja con precisión el software que se va a construir, o que
ha sido construido, son clave para el éxito del software
proceso de ingeniería de artículos.

No todas las organizaciones tienen una cultura de documentación.
Mentoría y gestión de requisitos. Es com

6.3. Modelo de validación

Normalmente es necesario validar la calidad de Los modelos desarrollados durante el análisis. Para examen-ple, en modelos de objetos, es útil realizar un análisis estático para verificar que las vías de comunicación existen entre objetos que, en las partes interesadas

mon en empresas dinámicas de nueva creación, impulsadas por un fuerte "visión del producto" y recursos limitados, para ver la documentación de requisitos como innecesaria gastos generales. Sin embargo, muy a menudo, ya que estos se expanden, a medida que crece su base de clientes, y A medida que su producto comienza a evolucionar, descubren que necesitan recuperar los requisitos que

características motivadas del producto para evaluar la impacto de los cambios propuestos. Por lo tanto, los requisitos la documentación y la gestión del cambio son clave para el éxito de cualquier proceso de requisitos.

7.1. Naturaleza iterativa de los requisitos Proceso

Existe una presión general en la industria del software. intente ciclos de desarrollo cada vez más cortos, y esto es particularmente pronunciado en altamente competitivo, sectores impulsados por el mercado. Además, la mayoría de los proyectos están limitados de alguna manera por su entorno, y muchas son actualizaciones o revisiones de, existen ing software donde la arquitectura es un hecho. En práctica, por lo tanto, casi siempre es poco práctico para implementar el proceso de requisitos como lineal, proceso determinista en el que el software requiere los interesados se obtienen de las partes interesadas, la base alineado, asignado y entregado al software Equipo de desarrollo. Ciertamente es un mito que el los requisitos para grandes proyectos de software son siempre perfectamente entendido o perfectamente especificado.

En cambio, los requisitos suelen iterar hacia Un nivel de calidad y detalle suficiente para permitir que las decisiones de diseño y adquisición sean hecho. En algunos proyectos, esto puede resultar en los requisitos se basaron antes de todos sus Las propiedades se entienden completamente. Esto conllevaría reelaboración sive si los problemas surgen tarde en el soft- proceso de ingeniería de artículos. Sin embargo, el software los ingenieros están necesariamente limitados por proyecto planes de manejo y por lo tanto deben tomar medidas para garantizar que la "calidad" de los requisitos sea lo más alto posible dados los recursos disponibles. Deberían, por ejemplo, hacer explícito cualquier supuestos que sustentan los requisitos como así como cualquier problema conocido.

Para productos de software que se desarrollan iter- activamente, un equipo de proyecto puede basar solo aquellos requisitos necesarios para la iteración actual. los especialista en requisitos puede continuar desarrollando requisitos para futuras iteraciones, mientras se desarrolla ers proceden con el diseño y construcción del iteración actual Este enfoque proporciona ers con valor comercial rápidamente, mientras minimiza ing el costo de retrabajo.

En casi todos los casos, comprensión de requisitos continúa evolucionando como diseño y desarrollo producto. Esto a menudo conduce a la revisión de requisitos al final del ciclo de vida. Quizás el punto más crucial en la comprensión del software requisitos es que una proporción significativa de los requisitos *serán* cambiar. Esto es a veces debido a errores en el análisis, pero con frecuencia es un consecuencia inevitable del cambio en el "medio ambiente ment ", por ejemplo, la operación del cliente o entorno empresarial, procesos regulatorios impuesto por las autoridades, o el mercado en qué software debe vender. Cualquiera sea la causa, es importante reconocer la inevitabilidad del cambio y tome medidas para mitigar sus efectos. El cambio tiene para ser administrado asegurando que los cambios propuestos pasar por un programa definido de revisión y aprobación cess y aplicando requisitos cuidadosos ing, análisis de impacto y configuración de software gestión (ver la Configuración del software Gestión KA). Por lo tanto, los requisitos pro cess no es simplemente una tarea de front-end en software desarrollo, pero abarca toda la vida del software ciclo. En un proyecto típico, el software requiere: Las actividades evolucionan con el tiempo a partir de la obtención para cambiar la gestión. Una combinación de top- análisis de abajo y métodos de diseño y fondo implementación y métodos de refactorización que reunirse en el medio podría proporcionar lo mejor de ambos mundos Sin embargo, esto es difícil de lograr en prácticas, ya que depende en gran medida de la madurez y experiencia de los ingenieros de software.

7.2. Gestión del cambio

La gestión del cambio es fundamental para la gestión. de requisitos. Este tema describe el papel de gestión del cambio, los procedimientos que necesitan estar en su lugar, y el análisis que debe aplicarse a los cambios propuestos. Tiene fuertes vínculos con el Soft- Gestión de la configuración de software KA.

7.3. Atributos de requisitos

Los requisitos deben consistir no solo en una especi ficación de lo que se requiere, pero también de lo auxiliar información, que ayuda a gestionar e interpretar los requisitos. Los requisitos de los atributos deben ser definido, grabado y actualizado como el software El material en desarrollo o mantenimiento evoluciona. Esto debe incluir las diversas clasificaciones

dimensiones del requisito (ver sección 4.1, Clasificación de requisitos) y la verificación método o sección de plan de prueba de aceptación relevante. También puede incluir información adicional, como Como resumen de cada requerimiento, el fuente de cada requisito y un historial de cambios. El atributo de requisitos más importante, cómo: siempre, es un identificador que permite los requisitos ser identificado de manera única e inequívoca.

7.4. Rastreo de requisitos

El seguimiento de requisitos se refiere a la recuperación ing la fuente de requisitos y prediciendo el efectos de los requisitos. El rastreo es fundamental para realizar análisis de impacto cuando los requisitos cambio. Un requisito debe ser rastreable Atento a los requisitos y partes interesadas que motivado (de un requisito de software de vuelta a los requisitos del sistema que ayuda a satisfacer, por ejemplo). Por el contrario, un requisito debería ser rastreable hacia adelante en los requisitos y diseñar entidades que lo satisfagan (por ejemplo, de un requisito del sistema en el software requerido menciones que se han elaborado a partir de ella, y en en los módulos de código que lo implementan, o el probar casos relacionados con ese código e incluso un determinado sección del manual del usuario que describe el funcionalidad real) y en el caso de prueba que lo verifica

El rastreo de requisitos para un proyecto típico ect formará un gráfico acíclico dirigido complejo (DAG) (ver Gráficos en la Fundación de Computación-KA) de los requisitos. Mantener un hasta gráfico de fechas o matriz de trazabilidad es una actividad que debe ser considerado durante todo el ciclo de vida de un producto. Si la información de trazabilidad no es actualizado a medida que continúan los cambios en los requisitos para suceder, la información de trazabilidad se convierte poco confiable para el análisis de impacto.

7.5. Requerimientos de medición

Como cuestión práctica, generalmente es útil tener algún concepto del "volumen" de la exigencia mentos para un producto de software en particular. Esta número es útil para evaluar el "tamaño" de un cambio en los requisitos, en la estimación del costo de una tarea de desarrollo o mantenimiento, o simplemente para utilizar como denominador en otras medidas. La medición del tamaño funcional (FSM) es una tecnología Nique para evaluar el tamaño de un cuerpo de funciones requisitos nacionales

Información adicional sobre la medida del tamaño y los estándares se encontrarán en Software Engi-Proceso Neering KA.

8. Herramientas de requisitos de software

Las herramientas para tratar con los requisitos de software caen a grandes rasgos en dos categorías: herramientas para modelar y herramientas para gestionar los requisitos.

Las herramientas de gestión de requisitos suelen ser compatibles portar una gama de actividades, incluida la documentación ción, seguimiento y gestión del cambio, y tener tuvo un impacto significativo en la práctica. De hecho, trac-En realidad, la gestión del cambio y el cambio son solo prácticas relacionable si es compatible con una herramienta. Desde requisitos la gestión es fundamental para un buen requerimiento práctica práctica, muchas organizaciones han invertido en herramientas de gestión de requisitos, aunque muchos más gestionan sus requisitos en más formas ad hoc y generalmente menos satisfactorias (por ejemplo, usando hojas de cálculo).

Página 46

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	ile 2011 erv [1 *] metro gm S	03 [2 *] iegers 20 W
1. Fundamentos de los requisitos de software		

1.1. Definición de un requisito de software	c4	c1
1.2. Requisitos de producto y proceso	c4s1	c1, c6
1.3. Requisitos funcionales y no funcionales	c4s1	c12
1.4. Propiedades emergentes	c10s1	
1.5. Requisitos cuantificables		c1
1.6. Requisitos del sistema y requisitos de software	c10s4	c1
2. Proceso de requisitos		
2.1. Modelos de proceso	c4s4	c3
2.2. Actores de proceso		c1, c2, c4, c6
2.3. Apoyo y gestión de procesos		c3
2.4. Calidad y mejora de procesos		c22, c23
3. Requisitos de obtención		
3.1. Fuentes de requisitos	c4s5	c5, c6, c9
3.2. Técnicas de obtención	c4s5	c6
4. Análisis de requisitos		
4.1. Clasificación de requisitos	c4s1	c12
4.2. Modelado conceptual	c4s5	c11
4.3. Diseño arquitectónico y asignación de requisitos	c10s4	c17
4.4. Negociación de requisitos	c4s5	c7
4.5. Análisis formal	c12s5	
5. Especificación de requisitos		
5.1. Documento de definición del sistema	c4s2	c10
5.2. Especificación de requisitos del sistema	c4s2, c12s2, c12s3, c12s4, c12s5	c10
5.3. Especificación de Requerimientos de Software	c4s3	c10
6. Validación de requisitos		
6.1. Revisiones de requisitos	c4s6	c15
6.2. Prototipos	c4s6	c13
6.3. Modelo de validación	c4s6	c15
6.4. Prueba de aceptacion	c4s6	c15

Page 47

	ile 2011 erv[1 *] metro om S	**03 [2 *] iegers 20 W**
7. Consideraciones prácticas		
7.1. Naturaleza iterativa del proceso de requisitos	c4s4	c3, c16
7.2. Gestión del cambio	c4s7	c18, c19
7.3. Atributos de requisitos	c4s1	c12, c14
7.4. Rastreo de requisitos		c20
7.5. Requerimientos de medición	c4s6	c18
8. Herramientas de requisitos de software		c21

LECTURAS ADICIONALES

I. Alexander y L. Beus-Dukic, *Descubriendo Requisitos* [5].

Un producto fácilmente digerible y prácticamente orientado. libro sobre requisitos de software, esto es quizás el mejor de los libros de texto actuales sobre cómo los diferentes Los elementos de los requisitos de software encajan entre sí. Es está lleno de consejos prácticos sobre (por ejemplo) cómo para identificar los diversos interesados del sistema y Cómo evaluar soluciones alternativas. Esta cubierto- la edad es ejemplar y sirve como referencia útil para técnicas clave como el modelado de casos de uso y Priorización de requisitos.

C. Potts, K. Takahashi y A. Antón, "Consulta- Análisis de requisitos basado "[6].

Este artículo es una descripción del trabajo fácilmente digerible que ha demostrado ser muy influyente en el desarrollo Opción de manejo de requisitos. Describe cómo y por qué la elaboración de requisitos no puede ser un proceso lineal por el cual el analista simplemente transcribe y reformula los requisitos obtenido del cliente. El papel de los escenarios. se describe de una manera que ayuda a definir su uso en descubrir y describir requisitos.

A. van Lamsweerde, *Requisitos Ingeniería: de los objetivos del sistema a UML Modelos a especificaciones de software* [7].

Sirve como una buena introducción a los requisitos. ingeniería pero su valor único es como referencia Esto para los requisitos orientados a objetivos de KAOS lenguaje de modelado Explica por qué modelo de objetivo ling es útil y muestra cómo puede integrarse con técnicas de modelado convencionales utilizando UML.

O. Gotel y A. Finkelstein, "Un análisis de la Problema de trazabilidad de requisitos "[8].

Este artículo es un trabajo clásico de referencia sobre una clave elemento de gestión de requisitos. Residencia en estudios empíricos, expone las razones y las barreras para el rastreo efectivo de requisitos ments. Es una lectura esencial para entender de por qué el rastreo de requisitos es un elemento esencial ment de un proceso de software efectivo.

N. Maiden y C. Ncube, "Adquisición de COTS Requisitos de selección de software "[9].

Este artículo es significativo porque reconoce explícitamente que los productos de software a menudo se integran componentes de terceros. Ofrece información sobre el problemas de selección de software estándar para Satisfacer los requisitos: generalmente hay una falta de coincidencia. Esto desafía algunos de los supuestos bajo fijando gran parte de los requisitos tradicionales han- dling, que tiende a asumir software personalizado.

Página 49

Referencias

[1 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[2 *] KE Wiegers, *Requisitos de software* , 2do ed., Microsoft Press, 2003.

[3] INCOSE, *Manual de Ingeniería de Sistemas: Una guía para los procesos del ciclo de vida del sistema y Actividades* , versión 3.2.2, Internacional Consejo de Ingeniería de Sistemas, 2012.

[4] S. Friedenthal, A. Moore y R. Steiner, *A Guía práctica de SysML: los sistemas Lenguaje de modelado* , 2ª ed., Morgan Kaufmann, 2012.

[5] I. Alexander y L. Beus-Deukic, *Descubriendo los requisitos: cómo especificar Productos y Servicios* , Wiley, 2009.

[6] C. Potts, K. Takahashi y AI Antón, "Análisis de requisitos basado en consultas" *Software IEEE,* vol. 11, no. 2, marzo de 1994, págs. 21–32.

[7] A. van Lamsweerde, *Requisitos. Ingeniería: de los objetivos del sistema a UML Modelos a especificaciones de software* , Wiley, 2009

[8] O. Gotel y CW Finkelstein, "Un análisis del problema de trazabilidad de requisitos " *Proc. 1er Conf. Internacional Requisitos Ing.* , IEEE, 1994.

[9] NA Maiden y C. Ncube, "Adquiriendo Requisitos de selección de software COTS " *Software IEEE,* vol. 15, no. 2 de marzo a abril 1998, págs. 46–56.

CAPITULO 2

DISEÑO DE SOFTWARE

SIGLAS

ADL	Descripción de la arquitectura Idioma
CBD	Diseño basado en componentes
CRC	Colaborador de responsabilidad de clase
DFD	Diagrama de flujo de datos
ERD	Relación diagrama de entidad
IDL	Lenguaje de descripción de interfaz
MVC	Controlador de vista de modelo
OO	Orientado a objetos
PDL	Lenguaje de diseño de programa

INTRODUCCIÓN

El diseño se define como "el proceso de definición". ing la arquitectura, componentes, interfaces y otras características de un sistema o componente " y "el resultado de [ese] proceso" [1]. Visto como un proceso, el diseño de software es el ingeniero de software ing actividad del ciclo de vida en la que el software requiere Los análisis se analizan para producir una descripción. ción de la estructura interna del software que Servir de base para su construcción. Un software diseño (el resultado) describe el archivo de software tecture, es decir, cómo se descompone el software y organizado en componentes, y el inter caras entre esos componentes. También debe Describir los componentes a un nivel de detalle que permite su construcción.

El diseño de software juega un papel importante en desarrollo de software: durante el diseño de software, los ingenieros de software producen varios modelos que forman una especie de modelo de la solución para ser implementado. Podemos analizar y evaluar estos modelos para determinar si o no nos permitirá cumplir con los diversos requisitos. También podemos examinar y evaluar alternativas soluciones y compensaciones. Finalmente, podemos usar el modelos resultantes para planificar el desarrollo posterior actividades, como la verificación y validación del sistema ción, además de usarlos como entradas y como punto de partida de construcción y prueba.

En una lista estándar de programas de ciclo de vida del software ceses, como el de ISO / IEC / IEEE Std. 12207, *Procesos del ciclo de vida del software* [2], diseño de software consiste en dos actividades que se ajustan entre software Análisis de requisitos y construcción de software:

- Diseño arquitectónico de software (a veces llamado diseño de alto nivel): desarrolla el nivel superior estructura y organización del software e identifica los diversos componentes.
- Diseño detallado del software: especifica cada componente con suficiente detalle para facilitar su construcción.

Esta área de conocimiento de diseño de software (KA) no trata todos los temas que incluyen el palabra "diseño". En la terminología de Tom DeMarco [3], los temas discutidos en este acuerdo de KA principalmente con D-design (diseño de descomposición), el objetivo de los cuales es mapear software en componente piezas. Sin embargo, debido a su importancia en en el campo de la arquitectura de software, también lo haremos abordar el diseño FP (diseño de patrón familiar), el cuyo objetivo es establecer una empresa explotable Monalidades en una familia de productos de software. Esta KA no aborda el diseño I (diseño de invención), que generalmente se realiza durante el software proceso de requisitos con el objetivo de conceptu- software de especificación y especificación para satisfacer el descubrimiento Necesidades y requisitos, ya que este tema es considerado parte del proceso de requisitos (Consulte los requisitos de software KA).

Este diseño de software KA está relacionado específicamente directamente a los requisitos de software, software

Figura 2.1. Desglose de temas para el diseño de software KA

Construcción, Ingeniería de Software Gestión-
Ment, Modelos de Ingeniería de Software y Metodos
ods, calidad de software y base de computación
iones KAs.

DESGLOSE DE TEMAS PARA DISEÑO DE SOFTWARE

El desglose de temas para el diseño de software KA se muestra en la Figura 2.1.

1. Fundamentos del diseño de software

Los conceptos, las nociones y la introducción terminológica forzado aquí forma una base subyacente para de pie el papel y el alcance del diseño de software.

1.1. Conceptos generales de diseño

[4 *, c1]

En el sentido general, el diseño puede ser visto como un forma de resolver problemas. Por ejemplo, el con-excepto un problema perverso, un problema sin solución definitiva: es interesante en términos de Comprender los límites del diseño. Un numero de otras nociones y conceptos también son de interés en entender el diseño en su sentido general: objetivos, restricciones, alternativas, representaciones y soluciones (ver Técnicas de resolución de problemas en el Fundaciones Informáticas KA).

1.2. Contexto del diseño de software

[4 *, c3]

El diseño de software es una parte importante del software. proceso de desarrollo de artículos. Para entender el papel del diseño de software, debemos ver cómo encaja en el ciclo de vida del desarrollo de software. Por lo tanto, Es importante comprender las principales características tics de análisis de requisitos de software, software diseño, construcción de software, prueba de software, y mantenimiento de software.

1.3. Proceso de diseño de software

[4 *, c2]

El diseño de software generalmente se considera un proceso de paso:

- Diseño arquitectónico (también conocido como alta- diseño de nivel y diseño de nivel superior) describe cómo se organiza el software en componentes.
- El diseño detallado describe el comportamiento deseado ior de estos componentes.

La salida de estos dos procesos es un conjunto de modelos y artefactos que registran las principales decisiones Siones que se han tomado, junto con una explicación. ración de la justificación de cada decisión no trivial. Al registrar la justificación, el mantenimiento a largo plazo Se mejora la capacidad del producto de software.

1.4. Principios de diseño de software

[4 *] [5 *, c6, c7, c21] [6 *, c1, c8, c9]

Un *principio* es "un enfoque integral y fundamental tal ley, doctrina o suposición "[7]. Software Los principios de diseño son nociones clave que proporcionan la base para muchos diseños de software diferentes el software se divide en una serie de pequeños componentes nombrados que tienen bien definidos interfaces que describen la interacción de componentes iones Por lo general, el objetivo es colocar diferentes funcionalidades y responsabilidades en diferentes componentes ent.

- *Encapsulación y* medios de *ocultación de información.* agrupando y empacando los detalles internos de una abstracción y haciendo esos detalles inaccesible a entidades externas.
- *Separación de interfaz e implementación.* Interfaz e implementación separadas implica definir un componente especificando ing una interfaz pública (conocida por los clientes) eso está separado de los detalles de cómo componente se realiza (ver encapsulación y información oculta arriba).
- *Suficiencia, integridad y primitividad.* Alcanzar la suficiencia y la integridad significa asegurar que un componente de software

enfoques y conceptos. Principios de diseño de software ples incluyen abstracción; acoplamiento y cohesión; descomposición y modularización; encapsula- ocultar / ocultar información; separación de interfaz e implementación; suficiencia, integridad, y primitividad; y separación de preocupaciones.

- La *abstracción* es "una vista de un objeto que se centra en la información relevante para un propósito particular e ignora el resto der de la información "[1] (ver Abstracción en las Fundaciones de Computación KA). En el contexto de diseño de software, dos abstracciones clave Los mecanismos de acción son parametrización y especificación. Abstracción por parametrización resúmenes de los detalles de la representación de datos representaciones representando los datos como nombrados parámetros Abstracción por especificación conduce a tres tipos principales de abstracción: abstracción procesal, abstracción de datos y abstracción de control (iteración).
- *Acoplamiento y cohesión.* El acoplamiento está definido como "una medida de la interdependencia entre módulos en un programa de computadora ", mientras que la cohesión se define como "una medida de la fuerza de asociación de los elementos dentro un módulo "[1].
- *Descomposición y modularización.* Descomponer posar y modularizar significa que grande captura todas las características importantes de Una abstracción y nada más. Primitivo- ness significa que el diseño debe basarse en patrones que son fáciles de implementar.
- *Separación de preocupaciones.* Una preocupación es un "Área de interés con respecto a un software diseño "[8]. Una preocupación de diseño es un área de diseño que es relevante para uno o más de sus partes interesadas Cada arquitectura ve marcos Una o más preocupaciones. Preocupaciones separadas por puntos de vista permite a los interesados interesados centrarse en algunas cosas a la vez y ofrece una medios de gestión de la complejidad [9].

2. Cuestiones clave en el diseño de software

Una serie de cuestiones clave deben abordarse cuando diseño de software Algunas son preocupaciones de calidad que todo software debe abordar, por ejemplo, rendimiento, seguridad, fiabilidad, usabilidad, etc. Otro tema importante es cómo descomponer, organizar y empaquetar componentes de software. Esto es tan fundamental que todos los enfoques de diseño abordarlo de una forma u otra (ver sección 1.4, Principios de diseño de software, y tema 7, Soft- Estrategias y métodos de diseño de software). A diferencia de, otros asuntos "tratan con algún aspecto del software comportamiento que no está en el dominio de la aplicación, pero que aborda algunos de los apoyos

Page 53

dominios "[10]. Tales problemas, que a menudo se cruzan Se ha hecho referencia a la funcionalidad del sistema. como *aspectos* que "tienden a no ser unidades de soft- descomposición funcional de la vajilla, sino más bien ser propiedades que afectan el rendimiento o semanal tics de los componentes de manera sistémica "[11]. Algunos de estos problemas clave y transversales son discutido en las siguientes secciones (presentado en orden alfabetico).

2.1. Concurrencia

[5 *, c18]

El diseño para la concurrencia tiene que ver con la descomposición. posando software en procesos, tareas e hilos y lidiar con problemas relacionados de eficiencia, atomicidad, sincronización y programación.

2.2. Control y manejo de eventos

[5 *, c21]

Este problema de diseño se refiere a cómo organizar el flujo de datos y control, así como cómo para manejar eventos reactivos y temporales a través de varios mecanismos como la invocación implícita y devoluciones de llamadas.

2.3. Persistencia de datos

[12 *, c9]

Este problema de diseño se refiere a cómo manejar dle datos de larga vida.

2.4. Distribución de componentes

2.6. Interacción y Presentación

[5 *, c16]

Este problema de diseño se refiere a cómo estructurar Ture y organice interacciones con los usuarios también como la presentación de información (por ejemplo, separación de presentación y lógica de negocios usando el enfoque Modelo-Vista-Controlador). Tenga en cuenta que este tema no especifica la interfaz de usuario detalles, que es la tarea del diseño de la interfaz de usuario (Consulte el tema 4, Diseño de la interfaz de usuario).

2.7. Seguridad

[5 *, c12, c18] [13 *, c4]

El diseño para la seguridad tiene que ver con cómo ventilar divulgación no autorizada, creación, cambio, eliminación o denegación de acceso a la información y otros recursos. También le preocupa cómo tolerar ataques o violaciones relacionadas con la seguridad por Limitación de daños, servicio continuo, exceso de velocidad reparación y recuperación, y falla y recuperación de forma segura El control de acceso es un problema fundamental concepto de seguridad, y uno también debe garantizar la Uso adecuado de la criptología.

3. Estructura y arquitectura del software

En su sentido estricto, una arquitectura de software es "El conjunto de estructuras necesarias para razonar sobre el sistema, que comprende elementos de software, relaciones entre ellos y propiedades de ambos " [14 *]. A mediados de los noventa, sin embargo, la arquitectura de software comenzó a emerger como una más amplia

[5 *, c18]

Este problema de diseño se refiere a cómo deshabilitar homenajear el software a través del hardware (incluido ing hardware de computadora y hardware de red), cómo se comunican los componentes y cómo el middleware se puede usar para lidiar con heteroge- software neous.

2.5. Manejo de errores y excepciones y falla Tolerancia

[5 *, c18]

Este problema de diseño se refiere a cómo pre ventilar, tolerar y procesar errores y lidiar con condiciones excepcionales

disciplina que involucró el estudio del software estructuras y arquitecturas en una forma más genérica camino. Esto dio lugar a una serie de interesantes conceptos sobre diseño de software en diferentes niveles Els de la abstracción. Algunos de estos conceptos pueden ser útil durante el diseño arquitectónico (para ejemplo, estilos arquitectónicos) así como durante El diseño detallado (por ejemplo, diseño golondrinas de mar). Estos conceptos de diseño también se pueden usar para diseñar familias de programas (también conocidos como líneas de productos). Curiosamente, la mayoría de estos con- los conceptos pueden ser vistos como intentos de describir, y así reutilizar, diseñar conocimiento.

Page 54

Diseño de software 2-5

3.1. Estructuras arquitectónicas y puntos de vista

[14 *, c1]

Diferentes facetas de alto nivel de un diseño de software. puede ser descrito y documentado Estas facetas a menudo se llaman vistas: "Una vista representa un parcial aspecto de una arquitectura de software que muestra propiedades específicas de un sistema de software "[14 *]. Puntos de vista pertenecen a distintos problemas asociados con el software diseño, por ejemplo, la vista lógica (satisfactoria los requisitos funcionales) frente a la vista del proceso (problemas de concurrencia) frente a la vista física (distribución problemas de bution) frente a la vista de desarrollo (cómo el el diseño se divide en unidades de implementación con representación explícita de las dependencias entre las unidades). Varios autores usan diferentes terminologías —como comportamiento vs. funcional vs. vistas de modelado estructural frente a datos. En resumen, un El diseño de software es un artefacto multifacético producido por el proceso de diseño y generalmente compuesto de Vistas relativamente independientes y ortogonales.

3.2. Estilos arquitectonicos

[14 *, c1, c2, c3, c4, c5]

Un estilo arquitectónico es "una especialización de ele- tipos de relaciones y relaciones, junto con un conjunto de restricciones sobre cómo se pueden usar "[14 *]. Un Así, el estilo arquitectónico puede ser visto como proporcionando La organización de alto nivel del software. Varios los autores han identificado una serie de archivos importantes estilos tecturales:

- Estructuras generales (por ejemplo, capas, tuberías). y filtros, pizarra)
- Sistemas distribuidos (por ejemplo, cliente- servidor, tres niveles, corredor)
- Sistemas interactivos (por ejemplo, Model-View- Controlador, Presentación-Abstracción-Control)
- Sistemas adaptables (por ejemplo, microker- nel, reflejo)
- Otros (por ejemplo, lote, intérpretes, pro control de cess, basado en reglas).

3.3. Patrones de diseño

[15 *, c3, c4, c5]

Sucintamente descrito, un patrón es "un común patrones que describen la organización de alto nivel de software, se pueden usar otros patrones de diseño para describir detalles en un nivel inferior. Estos más bajos los patrones de diseño de nivel incluyen lo siguiente:

- Patrones de creación (por ejemplo, constructor, fábrica, prototipo, singleton)
- Patrones estructurales (por ejemplo, adaptador, puente, composite, decorador, fachada, fly- peso, proxy)
- Patrones de comportamiento (por ejemplo, comando, intérprete, iterador, mediador, recuerdo, observador, estado, estrategia, plantilla, visitante).

3.4. Decisiones de diseño de arquitectura

[5 *, c6]

El diseño arquitectónico es un proceso creativo. Dur- En el proceso de diseño, los diseñadores de software tienen para tomar una serie de decisiones fundamentales que afectar profundamente el software y el desarrollo proceso de ment. Es útil pensar en el archi- proceso de diseño tectural a partir de una toma de decisiones perspectiva más que desde una perspectiva de actividad tive A menudo, el impacto en los atributos de calidad y las compensaciones entre los atributos de calidad competitivos son La base para las decisiones de diseño.

3.5. Familias de programas y marcos

[5 *, c6, c7, c16]

Un enfoque para proporcionar la reutilización de software diseños y componentes es diseñar familias de programas, también conocidos como líneas de productos de software . Esto se puede hacer identificando los puntos en común entre miembros de tales familias y diseñando componentes reutilizables y personalizables para tener en cuenta por la variabilidad entre los miembros de la familia.

En la programación orientada a objetos (OO), una clave la noción relacionada es la de un marco *:* una parte sistema de software completo que se puede extender creando instancias de extensiones específicas (como complementos).

4. Diseño de interfaz de usuario

El diseño de la interfaz de usuario es una parte esencial de la proceso de diseño de software. Diseño de interfaz de usuario

solución a un problema común en un contexto dado " [dieciséis]. Mientras que los estilos arquitectónicos se pueden y la máquina debe asegurar que la interacción entre lo humano proporciona una operación efectiva

Página 55

y control de la máquina. Para que el software alcanzar su máximo potencial, la interfaz de usuario debería estar diseñado para que coincida con las habilidades, experiencias y expectativas de sus usuarios anticipados.

4.1. Principios generales de diseño de interfaz de usuario [5 *, c29-web] [17 *, c2] [1]

- *Aprendizaje* . El software debe ser fácil de aprender para que el usuario pueda comenzar a trabajar rápidamente ing con el software.
- *Familiaridad del usuario* . La interfaz debe usar términos y conceptos extraídos de la experiencia Asesoramiento de las personas que utilizarán el software.
- *Consistencia* . La interfaz debe ser consistente tienda de campaña para que las operaciones comparables se activan imbuido de la misma manera.
- *Mínima sorpresa.* El comportamiento del software. No debería sorprender a los usuarios.
- *Recuperación.* La interfaz debe proporcionar mecanismos que permiten a los usuarios recuperarse de errores
- *Orientación del usuario.* La interfaz debería dar retroalimentación significativa cuando ocurren errores y Proporcionar ayuda relacionada con el contexto a los usuarios.
- *Diversidad de usuarios* . La interfaz debe proporcionar mecanismos de interacción apropiados para diversos tipos de usuarios y para usuarios con diferentes capacidades (ciega, mala vista, sordo, daltónico, etc.).

4.2. Problemas de diseño de la interfaz de usuario [5 *, c29-web] [17 *, c2]

El diseño de la interfaz de usuario debe resolver dos problemas clave:

- ¿Cómo debe interactuar el usuario con el ¿software?
- ¿Cómo debe ser la información del software? ser presentado al usuario?

El diseño de la interfaz de usuario debe integrar al usuario interacción y presentación de información. Usuario el diseño de la interfaz debe considerar un compromiso entre los estilos de interacción más apropiados y presentación para el software, el fondo y experiencia de los usuarios del software, y la dispositivos disponibles.

4.3. El diseño de modalidades de interacción del usuario [5 *, c29-web] [17 *, c2]

La interacción del usuario implica la emisión de comandos y proporcionando datos asociados al software. Usuario Los estilos de interacción se pueden clasificar en los siguientes bajando estilos primarios:

- *Pregunta-respuesta.* La interacción es esencial. restringido a una sola pregunta-respuesta intercambio entre el usuario y el software. El usuario hace una pregunta al software, y el software devuelve la respuesta a pregunta.
- *Manipulación directa* . Los usuarios interactúan con objetos en la pantalla de la computadora. Directo la manipulación a menudo incluye señalar dispositivo (como un mouse, trackball o una aleta ger en pantallas táctiles) que manipula un objeta e invoca acciones que especifican qué debe hacerse con ese objeto.
- *Selección de menú* . El usuario selecciona un comando de una lista de menú de comandos.
- *Completar formulario* . El usuario rellena los campos de un formar. A veces los campos incluyen menús, en en cuyo caso el formulario tiene botones de acción para El usuario para iniciar la acción.
- *Lenguaje de comando* . El usuario emite un mand y proporciona parámetros relacionados para dirigir el software qué hacer.
- *Lenguaje natural* . El usuario emite un Mand en lenguaje natural. Es decir, lo natural el lenguaje es una interfaz para un comando calibre y se analiza y traduce a soft- Comandos de Ware.

4.4. El diseño de la presentación de información [5 *, c29-web] [17 *, c2]

La presentación de la información puede ser textual o gráfica. Cal en la naturaleza. Un buen diseño mantiene la información. presentación separada de la información misma. El enfoque MVC (Modelo-Vista-Controlador) es Una forma efectiva de mantener la presentación de la información separándose de la información presentada.

1 Capítulo 29 es un capítulo basado en la web disponible a http://ifs.host.cs.st-andrews.ac.uk/Books/SE9/WebChapters /.

Page 56

Los ingenieros de software también consideran el software tiempo de respuesta y retroalimentación en el diseño de información

4.6. Localización e internacionalización [17 *, c8, c9]

Presentación del mation. El tiempo de respuesta es generalmente
medido desde el punto en el que un usuario ejecuta
cierta acción de control hasta que el software responda
con una respuesta Una indicación de progreso es el deseo
capaz mientras el software prepara la respuesta.
La retroalimentación se puede proporcionar al reiniciar el usuario
entrada mientras se completa el procesamiento.

Las visualizaciones abstractas se pueden usar cuando son grandes
Se deben presentar cantidades de información.

Según el estilo de presentación de la información
Además, los diseñadores también pueden usar el color para mejorar
interfaz. Hay varias pautas importantes:

- Limite la cantidad de colores utilizados.
- Use el cambio de color para mostrar el cambio de soft-
  estado de las mercancías
- Utilice la codificación de colores para apoyar la tarea del usuario.
- Utilice la codificación de colores de forma reflexiva y coherente.
  Tienda de campaña.
- Use colores para facilitar el acceso de las personas.
  con daltonismo o deficiencia de color
  (por ejemplo, use el cambio de saturación de color y
  brillo de color, trate de evitar el azul y el rojo
  combinaciones).
- No dependa solo del color para transmitir
  información importante para usuarios con diferentes
  capacidades (ceguera, mala visión, color-
  ceguera, etc.).

4.5. Proceso de diseño de interfaz de usuario
[5 *, c29-web] [17 *, c2]

El diseño de la interfaz de usuario es un proceso iterativo;
Los prototipos de interfaz a menudo se utilizan para determinar
Las características, la organización y el aspecto del software
Interfaz de usuario de Ware. Este proceso incluye tres
Actividades centrales Actividades principales:

- *Análisis de usuario.* En esta fase, el diseñador ana-
  analiza las tareas de los usuarios, el entorno de trabajo
  ment, otro software y cómo interactúan los usuarios
  con otras personas.
- *Creación de prototipos de software.* Prototipo en desarrollo
  software ayuda a los usuarios a guiar la evolución de
  La interfaz.
- *Evaluación de la interfaz.* Los diseñadores pueden obser-
  experiencias de los usuarios con la interfaz en evolución

El diseño de la interfaz de usuario a menudo necesita considerar
nacionalización y localización, que son medios
de adaptar el software a los diferentes idiomas,
diferencias regionales y los requisitos técnicos
mentos de un mercado objetivo. La internacionalización es el
proceso de diseño de una aplicación de software para que
Se puede adaptar a varios idiomas y regiones
sin grandes cambios de ingeniería. Localización
es el proceso de adaptación de software internacionalizado
Ware para una región o idioma específico agregando
componentes específicos del entorno local y la traducción de
texto. La localización y la internacionalización deberían
considerar factores como símbolos, números, curs
regencia, tiempo y unidades de medida.

4.7. Metáforas y modelos conceptuales
[17 *, c5]

Los diseñadores de interfaces de usuario pueden usar metáforas y
modelos conceptuales para configurar asignaciones entre
software y algún sistema de referencia conocido por el
usuarios en el mundo real, que pueden ayudar a los usuarios a
Aprenda y use más fácilmente la interfaz. Para examen-
ple, la operación "eliminar archivo" se puede convertir en un
metáfora usando el icono de un bote de basura.

Al diseñar una interfaz de usuario, el software
los neers deben tener cuidado de no usar más de uno
metáfora de cada concepto. Las metáforas también pres-
problemas potenciales con respecto a la internacionalización
alización, ya que no todas las metáforas son significativas
o se aplican de la misma manera dentro de todas las culturas.

5. Análisis de calidad de diseño de software y Evaluación

Esta sección incluye una serie de análisis de calidad.
Temas de evaluación y análisis que son específicamente
relacionado con el diseño de software. (Ver también el Software
Calidad KA.)

5.1. Atributos de calidad
[4 *, c4]

Varios atributos contribuyen a la calidad de
un diseño de software, que incluye varias "-bilidades"
(mantenibilidad, portabilidad, comprobabilidad, usabilidad)

y "-nesses" (corrección, robustez). Ahi esta
Una distinción interesante entre calidad
butes discernibles en tiempo de ejecución (por ejemplo, per-
formance, seguridad, disponibilidad, funcionalidad,
usabilidad), aquellos no discernibles en tiempo de ejecución (por
ejemplo, modificabilidad, portabilidad, reutilización,
comprobabilidad), y aquellos relacionados con la arquitectura
cualidades intrínsecas (por ejemplo, integ. conceptual)
Rity, corrección, integridad). (Ver también el
Calidad de software KA.)

5.2. Análisis de calidad y técnicas de evaluación
[4 *, c4] [5 *, c24]

Varias herramientas y técnicas pueden ayudar en el análisis
Evaluación y evaluación de la calidad del diseño del software.

5.3. Medidas
[4 *, c4] [5 *, c24]

Las medidas se pueden utilizar para evaluar o cuantificar
(estimar tativamente varios aspectos de un software
diseño; por ejemplo, tamaño, estructura o calidad.
La mayoría de las medidas propuestas dependen
sobre el enfoque utilizado para producir el diseño.
Estas medidas se clasifican en dos amplias
categorías:

- Metodo de diseño basado en funciones (estructurado)
  sures: medidas obtenidas mediante el análisis de funciones
  descomposición regional; generalmente representado
  usando un gráfico de estructura (a veces llamado un
  diagrama jerárquico) en el que varias medidas

• Revisiones de diseño de software: informales y para-técnicas malizadas para determinar la calidad de artefactos de diseño (por ejemplo, arquitectura revisiones, revisiones de diseño e inspecciones; técnicas basadas en escenarios; requisitos rastreo). Las revisiones de diseño de software también pueden evaluar la seguridad Ayudas para la instalación, operación acción y uso (por ejemplo, manuales y archivos de ayuda) pueden ser revisados.

• Análisis estático: estático formal o semiformal. análisis (no ejecutable) que se puede usar para evaluar un diseño (por ejemplo, falla-análisis de árbol o verificación cruzada automatizada). Análisis de vulnerabilidad de diseño (por ejemplo, análisis estático para debilidades de seguridad) puede se realizará si la seguridad es una preocupación. Formas el análisis de diseño usa modelos matemáticos que permiten a los diseñadores predicar el comportamiento y validar el rendimiento del software en lugar de tener que depender completamente de las pruebas. El análisis de diseño formal se puede utilizar para detectar especificación residual y errores de diseño (por-Haps causados por imprecisión, ambigüedad y a veces otros tipos de errores). (Ver también los modelos de ingeniería de software y Métodos KA.)

• Simulación y creación de prototipos: tecnología dinámica. niques para evaluar un diseño (por ejemplo, simulación de rendimiento o viabilidad prototipos).

se pueden calcular los sures.

• Medidas de diseño orientado a objetos: el diseño la estructura se representa típicamente como una clase diagrama, en el que se pueden tomar varias medidas calculado Medidas sobre las propiedades de la El contenido interno de cada clase también puede ser calculado

6. Notaciones de diseño de software

Existen muchas anotaciones para representar el diseño de software artefactos Algunos se usan para describir la estructura organización de un diseño, otros para representar soft-comportamiento de la vajilla. Ciertas anotaciones se usan principalmente durante el diseño arquitectónico y otros principalmente durante el diseño detallado, aunque algunos nota-se pueden usar para ambos propósitos. Adicionalmente, algunas anotaciones se usan principalmente en el contexto de métodos de diseño específicos (ver tema 7, Software Estrategias y métodos de diseño). Tenga en cuenta que el diseño de software a menudo se logra usando mul-anotaciones tiples. Aquí, se clasifican en anotaciones para describir el estructural (estático) vista frente a la vista conductual (dinámica).

6.1. Descripciones estructurales (vista estática)

[4 *, c7] [5 *, c6, c7] [6 *, c4, c5, c6, c7] [12 *, c7] [14 *, c7]

Las siguientes anotaciones, principalmente pero no siempre gráfica, describir y representar lo estructural aspectos de un diseño de software, es decir, son

se usa para describir los componentes principales y cómo están interconectados (vista estática):

- Lenguajes de descripción de arquitectura (ADL): idiomas textuales, a menudo formales, utilizados para describir la arquitectura del software en términos de componentes y conectores.
- Diagramas de clase y objeto: se utilizan para representar envió un conjunto de clases (y objetos) y sus interrelaciones
- Diagramas de componentes: se utilizan para representar un conjunto de componentes ("físico y reemplazar-parte [s] capaz de un sistema que [se ajusta] a y [proporcionar] la realización de un conjunto de caras "[18]) y sus interrelaciones.
- Tarjetas de colaborador de responsabilidad de clase (CRC): se usa para denotar los nombres de los componentes (clase), sus responsabilidades y sus nombres de componentes colaboradores.
- Diagramas de implementación: se utilizan para representar un conjunto de nodos (físicos) y sus interrelaciones relaciones y, por lo tanto, modelar lo físico aspectos del software.
- Diagramas de relación de entidad (ERD): utilizados representar modelos conceptuales de datos almacenados en repositorios de información.
- Lenguajes de descripción de interfaz (IDL): lenguajes de programación utilizados para definir las interfaces (nombres y tipos de exportados operaciones) de componentes de software.
- Diagramas de actividad: se utilizan para mostrar el flujo de control. de actividad en actividad. Se puede usar para representar resentido actividades concurrentes.
- Diagramas de comunicación: se utilizan para mostrar las interacciones que ocurren entre un grupo de objetos; El énfasis está en los objetos, su enlaces y los mensajes que intercambian en esos enlaces
- Diagramas de flujo de datos (DFD): se utilizan para mostrar flujo de datos entre elementos. Un flujo de datos dia-gram proporciona "una descripción basada en el modelo-ing el flujo de información alrededor de una red de elementos operativos, con cada elemento haciendo uso o modificando la información fluyendo hacia ese elemento ”[4 *]. Flujos de datos (y, por lo tanto, los diagramas de flujo de datos) pueden ser se utilizan para el análisis de seguridad, ya que ofrecen iden-tificación de posibles caminos para ataque y distensión cierre de información confidencial.
- Tablas de decisión y diagramas: utilizados para representar resentir combinaciones complejas de condiciones y acciones.
- Diagramas de flujo: se utilizan para representar el flujo de control y las acciones asociadas a ser realizado.
- Diagramas de secuencia: se utilizan para mostrar el inter acciones entre un grupo de objetos, con énfasis en la ordenación del tiempo de los mensajes pasado entre objetos.
- Transición de estado y diagramas de gráficos de estado:

• Gráficos de estructura: se utilizan para describir la llamada. estructura de programas (qué los módulos llaman, y son llamados por, qué otros módulos).

6.2. Descripciones de comportamiento (Vista dinámica)
[4 *, c7, c13] [5 *, c6, c7] [6 *, c4, c5, c6, c7] [14 *, c8]

Las siguientes anotaciones e idiomas, algunos gráficos y algunos textuales, se utilizan para describir el comportamiento dinámico de los sistemas de software y componentes. Muchas de estas anotaciones son de uso mayormente, pero no exclusivamente, durante el detallado diseño. Además, las descripciones de comportamiento pueden incluir una justificación para la decisión de diseño, como cómo un diseño cumplirá los requisitos de seguridad.

se usa para mostrar el flujo de control de estado a estado y cómo el comportamiento de un componente cambios basados en su estado actual en un estado máquina.
• Lenguajes de especificación formales: lenguaje textual indicadores que usan nociones básicas de matemáticas ematics (por ejemplo, lógica, conjunto, secuencia) definir rigurosa y abstractamente el software interfaces y comportamiento de componentes, a menudo en términos de precondiciones y poscondiciones. (Ver también los modelos y métodos de ingeniería de software ods KA.)
• Pseudocódigo y lenguajes de diseño de programas. (PDL): programación estructurada como lan- indicadores utilizados para describir, generalmente en el etapa de diseño detallado, el comportamiento de un pro- procedimiento o método.

Page 59

7. Estrategias y métodos de diseño de software

Existen varias estrategias generales para ayudar guiar el proceso de diseño. En contraste con general estrategias, los métodos son más específicos porque generalmente proporciona un conjunto de anotaciones para ser utilizado con el método, una descripción del proceso para se utilizará al seguir el método y un conjunto de pautas para usar el método. Tales métodos son útiles como marco común para equipos de ingenieros de software (Ver también el Software Engi- Modelos y métodos de orientación KA).

7.1. Estrategias generales
[4 *, c8, c9, c10] [12 *, c7]

Algunos ejemplos frecuentemente citados de estrategias generales útiles en el proceso de diseño incluyen la división y-conquistar y estrategias de refinamiento paso a paso, estrategias de arriba hacia abajo versus de abajo hacia arriba, estrategias haciendo uso de la heurística, el uso de patrones y patrones tern idiomas, y el uso de un iterativo e incremental Enfoque mental.

7.2. Diseño orientado a funciones (estructurado)
[4 *, c13]

Este es uno de los métodos clásicos de software. diseño, donde la descomposición se centra en la identificación ing las principales funciones de software y luego elab- orándolos y refinándolos en una jerarquía jerárquica abajo de manera. El diseño estructurado se usa generalmente después de un análisis estructurado, produciendo (entre otras cosas) diagramas de flujo de datos y asociados descripciones de procesos. Los investigadores han propuesto varias estrategias (por ejemplo, transformación análisis, análisis de transacciones) y heurística (para ejemplo, fan-in / fan-out, alcance del efecto vs. alcance de control) para transformar un DFD en un software arquitectura generalmente representada como una estructura gráfico.

7.3. Diseño orientado a objetos
[4 *, c16]

diseño de mediados de la década de 1980 (sustantivo = objeto; verbo = método; adjetivo = atributo), donde herencia- tance y polimorfismo juegan un papel clave, para el campo de diseño basado en componentes, donde metain- la formación se puede definir y acceder (a través de reflexión, por ejemplo). Aunque el diseño de OO raíces provienen del concepto de abstracción de datos, diseño impulsado por la responsabilidad ha sido propuesto como un enfoque alternativo para el diseño OO.

7.4. Diseño centrado en la estructura de datos
[4 *, c14, c15]

El diseño centrado en la estructura de datos comienza a partir de los datos estructura un programa manipula en lugar de La función que realiza. El ingeniero de software describe primero las estructuras de datos de entrada y salida y luego desarrolla la estructura de control del programa basado en estos diagramas de estructura de datos. Varios heurísticas han propuesto para tratar con especial casos, por ejemplo, cuando hay una falta de coincidencia entre las estructuras de entrada y salida.

7.5. Diseño basado en componentes (CBD)
[4 *, c17]

Un componente de software es una unidad independiente, tener interfaces bien definidas y dependencias ies que pueden ser compuestas y desplegadas independientemente pendiente. Direcciones de diseño basadas en componentes problemas relacionados con el suministro, desarrollo y integrando dichos componentes para mejorar reutilizar. Software de software reutilizado y listo para usar los ponentes deben cumplir los mismos requisitos de seguridad Ments como nuevo software. La gestión de confianza es una preocupación de diseño; componentes tratados como hav- ing cierto grado de confiabilidad debe no dependa de componentes menos confiables o servicios.

7.6. Otros metodos
[5 *, c19, c21]

También existen otros enfoques interesantes (ver el

Numerosos métodos de diseño de software basados en objetos han sido propuestos. El campo tiene evolucionado desde el principio orientado a objetos (OO)

Modelos y métodos de ingeniería de software
KA). Implementación de métodos iterativos y adaptativos. incremente el software y reduzca el énfasis en rigurosos requisitos de software y diseño .

60

Diseño de software 2-11

El diseño orientado a aspectos es un método por el cual el software se construye utilizando aspectos para implementar Mencione las preocupaciones y extensiones transversales que se identifican durante el requerimiento de software proceso de ments. La arquitectura orientada al servicio es una forma de construir software distribuido usando web servicios ejecutados en computadoras distribuidas. Suave- los sistemas de hardware a menudo se construyen utilizando servicios vicios de diferentes proveedores porque estándar protocolos (como HTTP, HTTPS, SOAP) tienen sido diseñado para apoyar la comunicación del servicio e intercambio de información de servicios.

8. Herramientas de diseño de software

[14 *, c10, Apéndice A]

Las herramientas de diseño de software se pueden utilizar para soportar creación de los artefactos de diseño de software durante El proceso de desarrollo de software. Pueden suplir portar parte o la totalidad de las siguientes actividades:

- traducir el modelo de requisitos en un representación de diseño;
- proporcionar apoyo para representar funciones componentes nacionales y sus interfaces;
- implementar el refinamiento heurístico y particionamiento;
- proporcionar pautas para la evaluación de la calidad.

Página 61

2-12 *SWEBOK® Guide* V3.0

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	03 gen 20 re Bu [4 *]	**ile 2011 erv metro om S [5 *]**	**es 1999 [6 *] age-Jon PAG**	**08 oreja 20 sh [12 *] Arroyo**	**08 llen 20 UNA [13 *]**	**ts et al. 2010 [14 *] en lem do**	**a et al. 1994 [15 *] metro a.m sol**	**[17 *] ielsen 1993 norte**
1. Diseño de software Fundamentos								
1.1. Diseño general Conceptos	c1							
1.2. El contexto de Diseño de software	c3							
1.3. El software Proceso de diseño	c2							
1.4. Diseño de software Principios	c1	c6, c7, c21	c1, c8, c9					
2. Cuestiones clave en Diseño de software								
2.1. Concurrencia		c18						
2.2. Control y Manejo de eventos		c21						
2.3. Persistencia de datos				c9				
2.4. Distribución de Componentes		c18						
2.5. Error y Manejo de excepciones y tolerancia a fallas		c18						
2.6. Interacción y Presentación		c16						
2.7. Seguridad		c12, c18			c4			
3. Estructura del software y arquitectura								
3.1. Arquitectónico Estructuras y Puntos de vista						c1		
3.2. Arquitectónico Estilos						c1, c2, c3, c4, c5		
3.3. Patrones de diseño							c3, c4, c5	

Page 62

	03 gen 20 re Bu [4 *]	**ile 2011 erv metro om S [5 *]**	**es 1999 [6 *] age-Jon PAG**	**08 oreja 20 sh [12 *] Arroyo**	**08 llen 20 UNA [13 *]**	**ts et al. 2010 [14 *] en lem do**	**a et al. 1994 [15 *] metro a.m sol**	**[17 *] ielsen 1993 norte**
3.4. Arquitectura Decisiones de diseño		c6						

3.5. Familias de Programas y Marcos		c6, c7, c16							
4. Interfaz de usuario Diseño									
4.1. Usuario general Diseño de interfaz Principio		c29-web							c2
4.2. Interfaz de usuario Problemas de diseño		c29-web							
4.3. El diseño de La interacción del usuario Modalidades		c29-web							
4.4. El diseño de información Presentación		c29-web							
4.5. Interfaz de usuario Proceso de diseño		c29-web							
4.6. Localización y Internacionalizacion									c8, c9
4.7. Metáforas y Modelos conceptuales									c5
5. Diseño de software Análisis de calidad y Evaluación									
5.1. Calidad Atributos	c4								
5.2. Calidad Análisis y Evaluación Tecnicas	c4	c24							
5.3. Medidas	c4	c24							

Page 63

	03			**08**	**08**			
	gen 20 [4 *] Fe Bu	**ile 2011 erv [5 *] metro om S**	**es 1999 [6 *] age-Jon PAG**	**oreja 20 sh [12 *] Arroyo**	**llen 20 [13 *] UNA**	**ts et al. 2010 [14 *] en lem do**	**a et al. 1994 [15 *] metro a.m sol**	**[17 *] ielsen 1993 norte**
6. Diseño de software Anotaciones								
6.1. Estructural Descripciones (estática Ver)	c7	c6, c7	c4, c5, c6, c7	c7		c7		
6.2. Comportamiento Descripciones (Vista dinámica)	c7, c13, c18	c6, c7	c4, c5, c6, c7			c8		
7. Diseño de software Estrategias y Métodos								
7.1. General Estrategias	c8, c9, c10			c7				
7.2. Función-Orientado Diseño (estructurado)	c13							
7.3. Orientado a objetos	c16							

Diseño		
7.4. Estructura de datos-Diseño Centrado	c14, c15	
7.5. Componente-Diseño basado (CBD)	c17	
7.6. Otros metodos	c19, c21	
8. Diseño de software Herramientas		c10, App. UNA

Página 64

LECTURAS ADICIONALES

Roger Pressman, *Ingeniería de software: A Enfoque del practicante (séptima edición)* [19]

Durante aproximadamente tres décadas, Roger Pressman *Ingeniería de software: un enfoque profesional* ha sido uno de los principales libros de texto del mundo en Ingeniería de software. Notablemente, esto complementa El libro de texto de Tary a [5 *] presenta exhaustivamente diseño de software, incluidos conceptos de diseño, diseño arquitectónico, diseño a nivel de componentes, diseño de interfaz de usuario, diseño basado en patrones y Diseño de aplicaciones web.

"El modelo de arquitectura 4 + 1 View" [20].

El trabajo seminal "The 4 + 1 View Model" orga- da una descripción de una arquitectura de software usando cinco vistas concurrentes. Las cuatro vistas de el modelo es la vista lógica, el desarrollo vista, la vista de proceso y la vista física. Además, los casos o escenarios de uso seleccionados son utilizado para ilustrar la arquitectura. Por lo tanto, la modelo contiene 4 + 1 vistas. Las vistas están acostumbradas a describir el software tal como lo han previsto diferentes partes interesadas, como usuarios finales, desarrolladores y gerentes de proyecto.

Len Bass, Paul Clements y Rick Kazman, *Arquitectura de software en la práctica* [21].

Este libro presenta los conceptos y las mejores prácticas. tices de arquitectura de software, lo que significa cuán suave El software está estructurado y cómo los componentes del software Las redes interactúan. Basándose en su propia experiencia, los autores cubren los temas técnicos esenciales para diseñar, especificar y validar software arquitecturas También enfatizan la importancia parte del contexto empresarial en el que grandes

Referencias

[1] *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010.

[2] *IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Norma para sistemas y Ingeniería de software: ciclo de vida del software Procesos* , IEEE, 2008.

[3] T. DeMarco, "La paradoja del software Arquitectura y diseño ", Premio Stevens Conferencia, 1999.

[4 *] D. Budgen, *Diseño de software* , 2ª ed., Addison-Wesley, 2003.

[5 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[6 *] M. Page-Jones, *Fundamentos del objeto-Diseño orientado en UML* , 1ª ed., Addison-Wesley, 1999.

[7] *Diccionario colegiado de Merriam-Webster* , 11a ed., 2003.

[8] *IEEE Std. 1069-2009 Estándar para Tecnología de la información: sistemas Diseño: descripciones de diseño de software* , IEEE, 2009.

[9] *ISO / IEC 42010: 2011 Sistemas y software Ingeniería: práctica recomendada para Descripción arquitectónica del software Sistemas Intensivos* , ISO / IEC, 2011.

[10] J. Bosch, *diseño y uso de software Arquitecturas: Adoptando y evolucionando a Enfoque de línea de productos* , ACM Press, 2000.

La vajilla está diseñada. Su objetivo es presentar software arquitectura en un entorno del mundo real, reflejando tanto las oportunidades como las limitaciones que organizan encuentro de organizaciones. Este es uno de los mejores libros. actualmente disponible en arquitectura de software.

[11] G. Kiczales et al., “Orientado a aspectos Programación, " *Proc. XI Conf. Europea Programación Orientada a Objetos* (ECOOP 97), Springer, 1997.

Página 65

[12 *] JG Brookshear, *Ciencias de la computación: un Descripción general* , décima edición, Addison-Wesley, 200K

[13 *] JH Allen et al., *Seguridad de software Ingeniería: una guía para el proyecto Gerentes* , Addison-Wesley, 2008.

[14 *] P. Clements et al., *Software de documentación Arquitecturas: Vistas y más allá* , 2ª ed., Pearson Education, 2010.

[15 *] E. Gamma et al., *Patrones de diseño: Elementos de objetos orientables reutilizables Software* , 1ª ed., Addison-Wesley Profesional, 1994.

[16] I. Jacobson, G. Booch y J. Rumbaugh, *El desarrollo de software unificado Proceso* , Addison-Wesley Professional, 1999

[17 *] J. Nielsen, *Ingeniería de usabilidad* , Morgan aufmann, 1993.

[18] G. Booch, J. Rumbaugh y I. Jacobson, El usuario del lenguaje de modelado unificado Guía, Addison-Wesley, 1999.

[19] RS Pressman, *Ingeniería de software: A Enfoque del practicante* , 7ª ed., McGraw-Hill, 2010.

[20] PB Kruchten, “El modelo de vista 4 + 1 de Arquitectura ”, *IEEE Software,* vol. 12, no. 6, 1995, págs. 42–55.

[21] L. Bass, P. Clements y R. Kazman, *Arquitectura de software en la práctica* , 3ª ed., Addison-Wesley Professional, 2013.

CAPÍTULO 3

CONSTRUCCION DE SOFTWARE

SIGLAS

API	Programación de aplicaciones Interfaz
Cunas	Comercial fuera de la plataforma
GUI	Interfaz gráfica del usuario
IDE	Desarrollo integrado Ambiente
Dios mio	grupo de administración de objetos
POSIX	Sistema operativo portátil Interfaz
TDD	Desarrollo guiado por pruebas
UML	Lenguaje de modelado unificado

INTRODUCCIÓN

El término construcción de software se refiere a la creación detallada de software de trabajo a través de un combinación de codificación, verificación, pruebas unitarias, pruebas de integración y depuración.

El área de conocimiento de Construcción de software (KA) está vinculado a todos los otros KAs, pero es más fuertemente vinculado al diseño de software y software Pruebas porque el proceso de construcción del software implica un importante diseño y prueba de software. El proceso utiliza el resultado del diseño y proporciona un entrada a la prueba ("diseño" y "prueba" en este caso refiriéndose a las actividades, no a los KAs). Boundaries entre diseño, construcción y pruebas (si cualquiera) variará según el ciclo de vida del software procesos que se utilizan en un proyecto.

Aunque algunos diseños detallados pueden ser performado antes de la construcción, mucho trabajo de diseño se realiza durante la actividad de construcción. Por lo tanto, la construcción de software KA está estrechamente vinculado al Software Design KA.

A lo largo de la construcción, ingenieros de software tanto la prueba unitaria como la integración prueban su trabajo. Por lo tanto, la construcción de software KA está estrechamente vinculado al Software Testing KA también.

La construcción de software generalmente produce el mayor número de elementos de configuración que necesitan para ser administrado en un proyecto de software (archivos fuente, documentación, casos de prueba, etc.). Por lo tanto, la Software Construction KA también está estrechamente vinculado al Software Configuration Management KA.

Si bien la calidad del software es importante en todos los KAs, el código es el último entregable de un soft-proyecto de software y, por lo tanto, la calidad de software KA es estrechamente vinculado a la construcción de software KA.

Dado que la construcción de software requiere conocimiento de algoritmos y prácticas de codificación, está muy cerca relacionado con las Fundaciones de Computación KA, que está preocupado por la base de la informática iones que apoyan el diseño y construcción de productos de software. También está relacionado con el proyecto mangestión, en la medida en que la gestión de la construcción ción puede presentar desafíos considerables.

DESGLOSE DE TEMAS PARA CONSTRUCCION DE SOFTWARE

La figura 3.1 da una representación gráfica de descomposición de nivel superior del desglose de la Construcción de software KA.

1. Fundamentos de construcción de software

Los fundamentos de la construcción de software incluyen

- minimizando la complejidad
- anticipando el cambio
- construcción para verificación
- reutilizar
- normas en construcción.

En cuanto a la construcción. Las siguientes secciones definen Los primeros cuatro conceptos se aplican también al diseño.

Figura 3.1. Desglose de temas para la construcción de software KA

Página 68

estos conceptos y describen cómo se aplican a construcción.

1.1. Minimizando Complejidad

[1 *]

La mayoría de las personas tienen una capacidad limitada para estructuras complejas e información en su recuerdos de trabajo, especialmente a largo plazo ods de tiempo. Esto demuestra ser un factor importante influyendo en cómo las personas transmiten la intención de se daña y conduce a uno de los impulsos más fuertes en la construcción de software: *minimizando* complejos ity. La necesidad de reducir la complejidad se aplica a esencialmente todos los aspectos de la construcción de software y es particularmente crítico para probar el software construcciones

En la construcción de software, complejidad reducida se logra enfatizando la creación de código eso es simple y legible en lugar de inteligente. Eso se logra mediante el uso de estándares (ver sección 1.5, Normas en la construcción), diseño modular (ver sección 3.1, Construcción Pruebas independientes y actividades operativas. Técnicas específicas que apoyan la construcción de la verificación incluye los siguientes estándares de codificación para admite revisiones de códigos y pruebas unitarias, organización código para admitir pruebas automatizadas y restringir Uso del lenguaje complejo o difícil de entender. estructuras de medida, entre otras.

1.4. Reutilizar

[2 *]

La reutilización se refiere al uso de activos existentes para resolver diferentes problemas En la construcción de software, tip- Los activos icos que se reutilizan incluyen bibliotecas, mods ules, componentes, código fuente y comercial activos disponibles en el mercado (COTS). Reutilizar es la mejor práctica emitido sistemáticamente, de acuerdo con un bien definido, proceso repetible La reutilización sistemática puede permitir productividad, calidad y software significativos Mejoras de costos.

La reutilización tiene dos facetas estrechamente relacionadas: "construcción para reutilización "y" construcción con reutilización ". medios anteriores para crear activos de software reutilizables,

Diseño) y muchas otras técnicas específicas (Ver sección 3.3, Codificación). También es apoyado por técnicas de calidad centradas en la construcción (ver sección sección 3.7, Calidad de la construcción).

1.2. Anticipando el cambio

[1 *]

La mayoría del software cambiará con el tiempo, y el La anticipación del *cambio* impulsa muchos aspectos de construcción de software; cambios en el medio ambiente Los elementos en los que funciona el software también afectan los objetivos Ware de diversas maneras.

Anticipar el cambio ayuda a los ingenieros de software construir software extensible, lo que significa que pueden mejorar un producto de software sin interrumpir La estructura subyacente.

Anticipar el cambio es apoyado por muchos espe técnicas específicas (ver sección 3.3, Codificación).

1.3. Construyendo para la Verificación

[1 *]

Construir para la verificación significa construir software de tal manera que las fallas puedan leerse ily encontrado por los ingenieros de software que escriben el software, así como por los probadores y usuarios durante mientras que esto último significa reutilizar activos de software en La construcción de una nueva solución. Reutilizar a menudo trasciende el límite de los proyectos, lo que significa los activos reutilizados se pueden construir en otros proyectos u organizaciones.

1.5. Estándares en construcción

[1 *]

Aplicando estándares de desarrollo externo o interno los colores durante la construcción ayudan a lograr un proyecto los objetivos de ect de eficiencia, calidad y costo. Específicamente, las opciones de programa permitido los subconjuntos de lenguaje ming y los estándares de uso son Ayudas importantes para lograr una mayor seguridad.

Estándares que afectan directamente la construcción. los problemas incluyen

- métodos de comunicación (por ejemplo, stan- Dards para formatos de documentos y contenidos)
- lenguajes de programación (por ejemplo, lan- medir estándares para lenguajes como Java y C ++)
- estándares de codificación (por ejemplo, estándares para convenciones de nombres, diseño y sangría)
- plataformas (por ejemplo, estándares de interfaz para llamadas al sistema operativo)

Página 69

- herramientas (por ejemplo, estándares esquemáticos para anotaciones como UML (modelado unificado Idioma)).

Uso de estándares externos. Construcción depende del uso de estándares externos para lenguajes de construcción, herramientas de construcción, técnicas interfaces e interacciones entre el software Construcción KA y otras KAs. Estándares vienen de numerosas fuentes, incluyendo hardware y especificaciones de la interfaz de software (como el Object Management Group (OMG)) e interna- organizaciones nacionales (como IEEE o ISO).

Uso de normas internas. Las normas también pueden ser creado sobre una base organizativa en la empresa nivel de consumo o para uso en proyectos específicos. Estas estándares de apoyo a la coordinación de actividades grupales lazos, minimizando la complejidad, anticipando el cambio, y construyendo para verificación.

2. Gestión de la construcción

2.1. Construcción en modelos de ciclo de vida

[1 *]

Se han creado numerosos modelos para desarrollar software; algunos enfatizan la construcción más que otros.

Algunos modelos son más lineales desde el contexto punto de vista de la construcción, como la cascada y modelos de ciclo de vida de entrega por etapas. Estos modelos tratar la construcción como una actividad que solo ocurre después de que se haya completado un trabajo de prerrequisitos significativo completado, incluido el trabajo de requisitos detallados, Amplio trabajo de diseño y planificación detallada. Los enfoques más lineales tienden a enfatizar Las actividades que preceden a la construcción (requieren la gestión de software y el proceso de software KAs).

En consecuencia, lo que se considera "con- estructura "depende hasta cierto punto de la vida modelo de ciclo utilizado. En general, el software con- la construcción es principalmente codificación y depuración, pero también implica planificación de la construcción, detallada diseño, pruebas unitarias, pruebas de integración y otras ocupaciones.

2.2. Planificación de la construcción

[1 *]

La elección del método de construcción es un aspecto clave. de la actividad de planificación de la construcción. La elección del método de construcción afecta el grado de qué requisitos previos de construcción se llevan a cabo, el orden en que se realizan y el grado en que deben completarse antes comienzan los trabajos de construcción.

El enfoque de la construcción afecta el proyecto. La capacidad del equipo de ect para reducir la complejidad, anticipar cambiar y construir para verificación. Cada uno de estos objetivos también pueden abordarse en el pro- cess, requisitos y niveles de diseño, pero será influenciado por la elección de la construcción método.

La planificación de la construcción también define el orden en el que los componentes se crean e integran, la estrategia de integración (por ejemplo, por fases o integración incremental), la calidad del software procesos de gestión, la asignación de tareas asignación de ingenieros de software específicos, y otras tareas, según el método elegido.

2.3. Medida de construcción

[1 *]

ments and design) y para crear secciones más distintas
Araciones entre actividades. En estos modelos, el
El énfasis principal de la construcción puede ser la codificación.

Otros modelos son más iterativos, como
prototipos evolutivos y desarrollo ágil
ment. Estos enfoques tienden a tratar la construcción
ción como una actividad que ocurre simultáneamente con
Otras actividades de desarrollo de software (incluyendo
requisitos, diseño y planificación) o que
las vueltas Estos enfoques tienden a mezclar diseño,
actividades de codificación y prueba, y a menudo tratan
La combinación de actividades como construcción (ver

Numerosas actividades de construcción y artefactos pueden
íder, incluido el código desarrollado, el código
modificado, código reutilizado, código destruido, código com-
plexidad, estadísticas de inspección de código, corrección de fallas y
tasas de búsqueda de fallas, esfuerzo y programación. Estos mea-
las garantías pueden ser útiles para gestionar
construcción, asegurando calidad durante la construcción,
y mejorando el proceso de construcción, entre
otros usos (ver el proceso de ingeniería de software
KA para más información sobre la medición).

Page 70

3. Consideraciones prácticas

La construcción es una actividad en la que el software
ingeniero tiene que lidiar a veces caótico y
cambiando las restricciones del mundo real, y él o ella
debe hacerlo con precisión. Debido a la influencia de lo real
limitaciones mundiales, la construcción está más impulsada por
consideraciones prácticas que otros KAs,
y la ingeniería de software es quizás la más artesanal
como en las actividades de construcción.

3.1. Diseño de la construcción

[1 *]

Algunos proyectos asignan considerables actividades de diseño
ity to construction, mientras que otros asignan diseño
a una fase centrada explícitamente en el diseño. Considerar-
menos de la asignación exacta, un diseño detallado
el trabajo ocurrirá a nivel de construcción, y eso
el trabajo de diseño tiende a ser dictado por restricciones
impuesto por el problema del mundo real que está siendo
abordado por el software.

Así como los trabajadores de la construcción construyen una
la estructura cal debe hacer modificaciones a pequeña escala
para tener en cuenta las lagunas imprevistas en el
planes de constructores, trabajadores de construcción de software
debe hacer modificaciones en una más pequeña o más grande
escalar para desarrollar detalles del diseño del software
durante la construcción.

Los detalles de la actividad de diseño en la construc-
el nivel de la sección es esencialmente el mismo que se describe en
Software Design KA, pero se aplican en
una escala más pequeña de algoritmos, estructuras de datos y
interfaces

3.2. Lenguajes de construcción

[1 *]

Los lenguajes de construcción incluyen todas las formas de
comunicación por la cual un humano puede especificar un
problema ejecutable solución a un problema. Estafa-
lenguajes de construcción y sus implementaciones
(por ejemplo, compiladores) puede afectar el software
atributos de calidad de rendimiento, fiabilidad, por-
Tability, y así sucesivamente. Pueden ser problemas graves
tributores a las vulnerabilidades de seguridad.

El tipo más simple de lenguaje de construcción.
es un *lenguaje de configuración,* en el que el software
los ingenieros eligen de un conjunto limitado de pre
opciones definidas para crear software nuevo o personalizado
instalaciones Los archivos de configuración basados en texto.
utilizado tanto en Windows como en Unix
los sistemas son ejemplos de esto, y el estilo de menú
listas de selección de algunos generadores de programas consistentes
tute otro ejemplo de un lenguaje de configuración.

Los lenguajes del kit de herramientas se utilizan para crear aplicaciones.
para de elementos en kits de herramientas (conjuntos integrados
de piezas reutilizables específicas de la aplicación); son
más complejo que los lenguajes de configuración.
Los idiomas del kit de herramientas pueden definirse explícitamente como
lenguajes de programación de aplicaciones o la aplicación
los cationes simplemente pueden estar implicados por el conjunto de herramientas
de interfaces.

Los lenguajes de script son tipos comúnmente utilizados
lenguajes de programación de aplicaciones. En algunos
lenguajes de script, los scripts se denominan archivos por lotes
o macros.

Los lenguajes de programación son los más flexibles.
tipo de lenguajes de construcción. También contienen
la menor cantidad de información sobre específicos
áreas de aplicación y procesos de desarrollo:
por lo tanto, requieren la mayor capacitación y habilidad
para usar de manera efectiva. La elección de la programación de lan-
el indicador puede tener un gran efecto en la probabilidad de
vulnerabilidades introducidas durante la codificación:
por ejemplo, el uso no crítico de C y C ++ son
elecciones cuestionables desde un punto de vista de seguridad.

Hay tres tipos generales de notación utilizados
para lenguajes de programación, a saber

- lingüística (p. Ej., C / C ++, Java)
- formal (p. Ej., Evento-B)
- visual (p. Ej., MatLab).

Las notaciones lingüísticas se distinguen en par-
ticular mediante el uso de cadenas de texto para representar
construcciones complejas de software. La combina-
La combinación de cadenas textuales en patrones puede tener una
sintaxis tipo oración. Usado adecuadamente, cada uno de ellos
la cadena debe tener una fuerte connotación semántica
proporcionando una comprensión intuitiva inmediata
de lo que sucederá cuando el software construc-
Se ejecuta la acción.

Las anotaciones formales se basan menos en intuitivos, todos
significado del día de palabras y cadenas de texto y más
en definiciones respaldadas por precisas, sin ambigüedades
definiciones propias y formales (o matemáticas).
Notaciones formales de construcción y métodos formales.
los ods están en la base semántica de la mayoría de las formas de

Page 71

anotaciones de programación del sistema, donde la precisión
el comportamiento del tiempo y la capacidad de prueba son más importantes
que la facilidad de mapear en lenguaje natural. Por-
construcciones mal también usan formas definidas con precisión
de combinar símbolos que eviten la ambigüedad
de muchas construcciones de lenguaje natural.

Las anotaciones visuales dependen mucho menos del texto.
anotaciones de construcción lingüística y formal
y en su lugar confiar en la interpretación visual directa
y colocación de entidades visuales que representan el
software subyacente La construcción visual tiende a
estar algo limitado por la dificultad de hacer
Declaraciones "complejas" utilizando solo el arreglo-
Mención de iconos en una pantalla. Sin embargo, estos íconos
pueden ser herramientas poderosas en casos donde el primario
la tarea de programación es simplemente construir y "ajustar"
una interfaz visual para un programa, el detallado
comportamiento del cual tiene una definición subyacente.

3.3. Codificación [1 *]

Las siguientes consideraciones se aplican al software
actividad de codificación de construcción de artículos:

- Técnicas para crear comprensibles.
  código fuente, incluidas las convenciones de nomenclatura
  y diseño de código fuente;
- Uso de clases, tipos enumerados, variables,
  constantes con nombre y otras entidades similares;
- Uso de estructuras de control;
- Manejo de condiciones de error, ambas antici
  pated y excepcional (entrada de datos incorrectos, para
  ejemplo);
- Prevención de infracciones de seguridad a nivel de código
  (desbordamientos de búfer o límites de índice de matriz, para
  ejemplo);
- Uso de recursos mediante el uso de mecanismos de exclusión
  nismos y disciplina para acceder en serie
  recursos reutilizables (incluidos hilos y
  bloqueos de bases de datos);
- Organización del código fuente (en estado-
  menciones, rutinas, clases, paquetes u otros
  estructuras);
- Documentación del código;
- Ajuste de código,

3.4. Pruebas de construcción [1 *]

La construcción implica dos formas de prueba,
que suelen realizar los ingenieros de software
Neer quien escribió el código:

- Examen de la unidad
- Pruebas de integración.

El propósito de las pruebas de construcción es reducir
la brecha entre el momento en que se insertan las fallas
en el código y el momento en que esas fallas son
detectado, reduciendo así el costo incurrido para
arreglalos. En algunos casos, los casos de prueba son escritos
diez después de que se haya escrito el código. En otros casos
Se pueden crear casos de prueba antes de escribir el código.

Las pruebas de construcción generalmente implican un
subconjunto de los diversos tipos de pruebas, que
se describen en el Software Testing KA. por
ejemplo, las pruebas de construcción no suelen
incluye pruebas del sistema, pruebas alfa, pruebas beta,
pruebas de estrés, pruebas de configuración, pruebas de usabilidad
ing u otros tipos de pruebas más especializadas.

Se han publicado dos normas sobre el tema.
de pruebas de construcción: IEEE Standard 829-1998 ,
Norma IEEE para la documentación de prueba de software,
y IEEE Standard 1008-1987, *IEEE Standard*
para pruebas de unidad de software .

(Consulte las secciones 2.1.1., Pruebas unitarias y 2.1.2.,
Pruebas de integración, en el Software Testing KA
para material de referencia más especializado).

3.5. Construcción para reutilización [2 *]

La construcción para su reutilización crea software que tiene
un potencial para ser reutilizado en el futuro para el
proyecto actual u otros proyectos que toman una amplia
perspectiva basada en sistemas múltiples. Construcción para
la reutilización generalmente se basa en análisis de variabilidad y
diseño. Para evitar el problema de los clones de código,
se desea encapsular fragmentos de código reutilizables
en bibliotecas o componentes bien estructurados.

Las tareas relacionadas con la construcción de software para
La reutilización durante la codificación y las pruebas son las siguientes:

Page 72

- Implementación de variabilidad con meca-
  nismos como parametrización, condicional
  compilación, patrones de diseño, etc.
- Encapsulación de variabilidad para hacer el suave
  activos de hardware fáciles de configurar y personalizar.
- pruebas unitarias y pruebas de integración (ver sección
  sección 3.4, Pruebas de construcción)
- desarrollo de prueba primero (ver sección 2.2 en el
  Prueba de software KA)
- uso de afirmaciones y programación defensiva

- Probar la variabilidad proporcionada por el reusactivos de software capaces.
- Descripción y publicación de software reutilizable. bienes de consumo.

3.6. Construcción con reutilización [2 *]

Construcción con reutilización significa crear nuevos software con la reutilización de software existente bienes. El método más popular de reutilización es reutilice el código de las bibliotecas proporcionadas por el lenindicador, plataforma, herramientas que se utilizan o una organización repositorio nacional. Aparte de estos, la aplicación iones desarrolladas hoy hacen uso de muchos bibliotecas de código abierto. Reutilizado y listo para usar el software a menudo tiene la misma calidad (o mejor) requisitos como software recientemente desarrollado (para ejemplo, nivel de seguridad).

Las tareas relacionadas con la construcción de software con La reutilización durante la codificación y las pruebas son las siguientes:

- La selección de las unidades reutilizables, datosbases, procedimientos de prueba o datos de prueba.
- La evaluación del código o la reutilización de la prueba.
- La integración de los activos de software reutilizables. en el software actual.
- El informe de la información de reutilización en nuevos código, procedimientos de prueba o datos de prueba.

3.7. Calidad de la construcción [1 *]

Además de las fallas resultantes de los requisitos y diseño, fallas introducidas durante la construcción puede dar lugar a graves problemas de calidad, para examen ple, vulnerabilidades de seguridad. Esto incluye no solo fallas en la funcionalidad de seguridad pero también fallas en otros lugares que permiten evitar este funcional ity y otras debilidades o violaciones de seguridad.

Existen numerosas técnicas para garantizar la calidad ity de código como se construye. La tecnología primaria Las niqueras utilizadas para la calidad de la construcción incluyen

- depuración
- inspecciones
- revisiones técnicas, incluyendo seguridad-ori- Revisiones ented (ver sección 2.3.2 en Soft- Ware Quality KA)
- análisis estático (ver sección 2.3 del Soft- Ware Quality KA)

La técnica o técnicas específicas seleccionadas dependerá de la naturaleza del software que se esté considerando estructurado, así como en el conjunto de habilidades del software ingenieros realizando las actividades de construcción combatación. Los programadores deben conocer buenas prácticas. y vulnerabilidades comunes, por ejemplo, de listas ampliamente reconocidas sobre vulner- común habilidades. Análisis estático automatizado de código para las debilidades de seguridad están disponibles para varias empresas lenguajes de programación mon y se pueden utilizar en proyectos críticos de seguridad.

Las actividades de calidad de construcción son diferentes. las otras actividades de calidad por su enfoque. Las actividades de calidad de la construcción se centran en el código y artefactos que están estrechamente relacionados con el código, como como diseño detallado, a diferencia de otros artefactos que están menos directamente conectados al código, como como requisitos, diseños de alto nivel y planes.

3.8. Integración [1 *]

Una actividad clave durante la construcción es la integración ción de rutinas construidas individualmente, clases, componentes y subsistemas en un solo sistema tem. Además, un sistema de software particular puede necesitar integrarse con otro software o sistemas de hardware

Preocupaciones relacionadas con la integración de la construcción. Incluir la planificación de la secuencia en la que se integrarán las redes, identificando qué se necesita vajilla, creando andamios para soportar versiones provisionales del software, determinando El grado de prueba y calidad del trabajo realizado componentes antes de que se integren, y

Page 73

determinar puntos en el proyecto en los que interino Se prueban versiones del software.

Los programas pueden integrarse por medio de el enfoque gradual o incremental. Por fases integración, también llamada integración "big bang", implica retrasar la integración del componente partes de software hasta todas las partes destinadas al lanzamiento En una versión están completos. Integración incremental se cree que ofrece muchas ventajas sobre el tráfico integración gradual gradual, por ejemplo, más fácil ubicación del error, monitoreo mejorado del progreso, entrega de producto anterior y cliente mejorado relaciones. En la integración incremental, el desarrollo ers escriben y prueban un programa en piezas pequeñas y luego combine las piezas una a la vez. Adicional infraestructura de prueba, como trozos, controladores y Objetos simulados, generalmente son necesarios para permitir la Integración mental. Construyendo e integrando una unidad a la vez (por ejemplo, una clase o un componente

4.2. Problemas de tiempo de ejecución orientado a objetos [1 *]

Los lenguajes orientados a objetos admiten una serie de mecanismos de tiempo de ejecución que incluyen polimorfismo y reflexion. Estos mecanismos de tiempo de ejecución aumentar la flexibilidad y adaptabilidad de los objetos programas orientados Polimorfismo es la habilidad de un lenguaje para soportar operaciones generales consabiendo hasta el tiempo de ejecución qué tipo de concreto objetos que incluirá el software. Porque el el programa no conoce los tipos exactos de objetos de antemano, el comportamiento exacto es disuasorio extraído en tiempo de ejecución (llamado enlace dinámico).

La reflexión es la capacidad de un programa para observar y modificar su propia estructura y comportamiento en la ejecución hora. La reflexión permite la inspección de clases, interfaces, campos y métodos en tiempo de ejecución consabiendo sus nombres en tiempo de compilación. También

nent), el proceso de construcción puede proporcionar tempra-
comentarios a desarrolladores y clientes. Otro
Las ventajas de la integración incremental incluyen
localización de errores más fácil, monitor de progreso mejorado
ing, unidades más probadas, etc.

4. Tecnologías de construcción

4.1. Diseño y uso de API

[3 *]

Una interfaz de programación de aplicaciones (API) es la
conjunto de firmas que se exportan y están disponibles para
los usuarios de una biblioteca o un marco para escribir su
aplicaciones. Además de las firmas, una API debería
siempre incluya declaraciones sobre el programa
efectos y / o comportamientos (es decir, su semántica).

El diseño de la API debería intentar facilitar la API
aprender y memorizar, conducir a un código legible, ser
difícil de usar mal, fácil de extender, completo,
y mantener la compatibilidad con versiones anteriores. Como
Las API generalmente duran más que sus implementaciones-
una biblioteca o marco ampliamente utilizado, se desea
que la API sea sencilla y se mantenga estable
facilitar el desarrollo y mantenimiento de la
aplicaciones de cliente.

El uso de API implica los procesos de selección
ing, aprendizaje, prueba, integración y posiblemente
API extendidas proporcionadas por una biblioteca o marco
trabajo (ver sección 3.6, Construcción con reutilización).

permite la creación de instancias en tiempo de ejecución de nuevos objetos y
invocación de métodos usando clase parametrizada
y nombres de métodos.

4.3. Parametrización y Genéricos

[4 *]

Tipos parametrizados , también conocidos como genéricos.
(Ada, Eiffel) y plantillas (C ++), habilitan el
definición de un tipo o clase sin especificar todo
los otros tipos que usa Los tipos no especificados son
suministrado como parámetros en el punto de uso. Param-
los tipos eterizados proporcionan una tercera vía (además de
herencia de clase y composición de objetos) para comp
plantean comportamientos en software orientado a objetos.

4.4. Afirmaciones, diseño por contrato y defensivo Programación

[1 *]

Una aserción es un predicado ejecutable que es
colocado en un programa, generalmente una rutina o macro
que permite verificaciones de tiempo de ejecución del programa. Asser-
Las opciones son especialmente útiles en las producciones de alta fiabilidad.
gramos Permiten a los programadores a más rápidamente
eliminar suposiciones de interfaz no coincidentes, errores
que se arrastran cuando se modifica el código, y así sucesivamente.
Las afirmaciones normalmente se compilan en el código en
tiempo de desarrollo y luego se compilan a partir de
código para que no degraden el rendimiento.

Page 74

El diseño por contrato es un enfoque de desarrollo
en el cual las precondiciones y postcondiciones son
incluido para cada rutina. Cuando las condiciones previas
y se utilizan condiciones posteriores, cada rutina o
Se dice que la clase forma un contrato con el resto de
el programa. Además, un contrato proporciona un
especificación precisa de la semántica de una rutina,
y así ayuda a la comprensión de su comportamiento.
El diseño por contrato está pensado para mejorar el
calidad de construcción de software.

La programación defensiva significa proteger un
rutina de ser interrumpida por entradas inválidas.
Las formas comunes de manejar entradas inválidas incluyen
comprobar los valores de todos los parámetros de entrada
y decidir cómo manejar las entradas malas. Asser-
a menudo se usan en programación defensiva para
Verifique los valores de entrada.

4.5. Manejo de errores, manejo de excepciones y Tolerancia a fallos

[1 *]

La forma en que se manejan los errores afecta el software
capacidad de cumplir con los requisitos relacionados con la con-
ness, robustez y otros atributos no funcionales
butes Las afirmaciones a veces se usan para verificar
por errores Otras técnicas de manejo de errores, como
como devolver un valor neutral, sustituyendo el siguiente
pieza de datos válidos, registrando un mensaje de advertencia,
devolver un código de error o apagar el software
vajilla: también se utilizan.

Se utilizan excepciones para detectar y procesar
errores o eventos excepcionales. La estructura básica

o que contengan sus efectos si la recuperación no es pos-
Sible. Las estrategias de tolerancia a fallas más comunes
incluyen copias de seguridad y reintentos, utilizando auxiliares
código, utilizando algoritmos de votación y reemplazando un
valor erróneo con un valor falso que tendrá
Un efecto benigno.

4.6. Modelos ejecutables

[5 *]

Los modelos ejecutables abstraen los detalles de
lenguajes de programación específicos y decisiones
sobre la organización del software. Diferente
de modelos de software tradicionales, una especificación
construido en un lenguaje de modelado ejecutable como
xUML (UML ejecutable) se puede implementar en
Varios entornos de software sin cambios.
Un compilador de modelo ejecutable (transformador) puede
convertir un modelo ejecutable en una implementación
utilizando un conjunto de decisiones sobre el hardware objetivo
y entorno de software. Así, construyendo
los modelos ejecutables pueden considerarse como una forma de
construyendo software ejecutable.
Estos modelos ejecutables son un soporte básico
El inicio de la arquitectura basada en modelos (MDA)
tive del Object Management Group (OMG). Un
El modelo ejecutable es una forma de especificar completamente
un modelo independiente de plataforma (PIM); un PIM es
un modelo de solución a un problema que no
confíe en cualquier tecnología de implementación. Entonces
un modelo específico de plataforma (PSM), que es un
modelo que contiene los detalles del implemen-
tation, se puede producir tejiendo juntos el

de una excepción es que una rutina usa *throw* para lanzar una excepción detectada y una excepción dling block *detectará* la excepción en un *try-catch* bloquear. El bloque try-catch puede procesar el error condición neosa en la rutina o puede regresar control a la rutina de llamada. Manejo de excepciones las políticas deben ser cuidadosamente diseñadas siguiendo ing principios comunes tales como incluir en el mensaje de excepción toda la información que condujo a la excepción, evitar bloques vacíos, sabiendo las excepciones que arroja el código de la biblioteca, tal vez construir un reportero de excepción centralizado, y estandarizar el uso de excepciones por parte del programa.

La tolerancia a fallas es una colección de técnicas. que aumentan la confiabilidad del software al detectar errores y luego recuperarse de ellos si es posible

PIM y la plataforma en la que se basa.

4.7. Construcción basada en estado y basada en tablas Tecnicas

[1 *]

Programación basada en estado o basada en autómatas programación, es una tecnología de programación usando máquinas de estados finitos para describir el programa comportamientos Los gráficos de transición de un estado. la máquina se usa en todas las etapas del desarrollo de software Opción (especificación, implementación, depuración- ging y documentación). La idea principal es construir programas de computadora de la misma manera Se realiza la automatización de los procesos tecnológicos. La programación basada en el estado generalmente se combina

Página 75

con programación orientada a objetos, formando un nuevo enfoque compuesto llamado *basado en el estado, programación orientada a objetos.*

Un método basado en tablas es un esquema que utiliza tablas para buscar información en lugar de usar declaraciones lógicas (como *if* y *case*). Utilizado en circunstancias apropiadas, código controlado por tabla es más simple que la lógica complicada y más fácil de modificar. Cuando se utilizan métodos basados en tablas, el el programador aborda dos problemas: qué información ción para almacenar en la mesa o mesas, y cómo efi- Acceso a la información de la tabla.

4.8. Configuración de tiempo de ejecución y Internacionalizacion

[1 *]

Para lograr una mayor flexibilidad, un programa es a menudo construido para soportar el tiempo de enlace tardío de su variedad ables La configuración de tiempo de ejecución es una técnica que enlaza valores variables y configuraciones de programa cuando el programa se está ejecutando, generalmente actualizando y leer archivos de configuración en un modo justo a tiempo.

La internacionalización es la actividad técnica. ity de preparar un programa, generalmente interactivo software, para soportar múltiples configuraciones regionales. El correlativo actividad de patrocinio, *localización,* es la actividad de Modificar un programa para soportar un local específico idioma. El software interactivo puede contener doz- ens o cientos de mensajes, pantallas de estado, ayuda mensajes, mensajes de error, etc. El diseño y los procesos de construcción deben acomodar problemas de cadena y juego de caracteres que incluyen se utilizará el conjunto de caracteres, qué tipo de cadenas se utilizan, cómo mantener las cadenas sin cambiando el código y traduciendo las cadenas en diferentes idiomas con un impacto mínimo en el código de procesamiento y la interfaz de usuario.

4.9. Procesamiento de entrada basado en gramática

[dieciséis*]

El procesamiento de entrada basado en la gramática implica sintaxis análisis o *análisis* del flujo de token de entrada. Eso implica la creación de una estructura de datos (llamada *árbol de análisis* o *árbol de sintaxis*) que representa la entrada datos. El recorrido transversal del árbol de análisis usu- variables definidas por el programador que pueblan el árbol. Después de construir el árbol de análisis, el programa lo usa como entrada para los procesos computacionales.

4.10. Primitivas de concurrencia

[7 *]

Una primitiva de sincronización es una programación abstracción proporcionada por un lenguaje de programación o el sistema operativo que facilita la concurrencia rency y sincronización. Concurrencia bien conocida Las primitivas de referencia incluyen semáforos, monitores, y mutexes.

Un semáforo es una variable protegida o abstracta tipo de datos que proporciona una abstracción simple pero útil ión para controlar el acceso a un recurso común por múltiples procesos o hilos en un concurrente entorno de programación

Un monitor es un tipo de datos abstracto que presenta un conjunto de operaciones definidas por el programador que son ejecutado con exclusión mutua. Un monitor con- tiene la declaración de variables compartidas y pro- procedimientos o funciones que operan en esas variedades ables La construcción del monitor garantiza que solo un proceso a la vez está activo dentro del monitor.

Un mutex (exclusión mutua) es una sincronización primitiva que otorga acceso exclusivo a un recurso compartido por un solo proceso o subproceso en un momento.

4.11. Middleware

[3 *] [6 *]

Middleware es una clasificación amplia para soft- Ware que proporciona servicios por encima de la operación capa del sistema aún debajo del programa de aplicación capa. Middleware puede proporcionar tiempo de ejecución contener- ers para componentes de software para proporcionar mensaje paso, persistencia y una ubicación transparente a través de una red. El middleware se puede ver como un conector entre los componentes que usan el middleware Medio moderno orientado a mensajes generalmente proporciona un bus de servicio empresarial (ESB), que admite la interacción orientada a servicios ción y comunicación entre múltiples programas aplicaciones de software.

Ally da la expresión que acaba de analizar. El analizador comprueba la tabla de símbolos para la presencia de

Page 76

4.12. Métodos de construcción para distribuidos Software

[7 *]

Un sistema distribuido es una colección de físicamente sistemas informáticos separados, posiblemente heterogéneos equipos conectados en red para proporcionar a los usuarios acceso a los diversos recursos que el sistema mantiene. La construcción de software distribuido es distinguido de la construcción de software tradicional ción por cuestiones como el paralelismo, la comunicación ción y tolerancia a fallas.

La programación distribuida generalmente cae en uno de varias categorías arquitectónicas básicas: cliente-servidor, arquitectura de 3 niveles, arquitectura de n niveles, dis- objetos tributarios, acoplamiento flojo o acoplamiento apretado (Ver sección 14.3 de las Fundaciones de Computación KA y la sección 3.2 del Software Design KA).

4.13. Construyendo sistemas heterogéneos

[6 *]

Los sistemas heterogéneos consisten en una variedad de unidades computacionales especializadas de diferentes tipos, tales como procesadores de señal digital (DSP), micro- controladores y procesadores periféricos. Estas las unidades computacionales son controladas independientemente y comunicarse entre sí. Incrustado Los sistemas son típicamente sistemas heterogéneos.

El diseño de sistemas heterogéneos puede requieren la combinación de varias especificaciones idiomas para diseñar diferentes partes de el sistema, en otras palabras, hardware / software codeign. Los temas clave incluyen multilenguaje validación, cosimulación e interfaz.

Durante el código de hardware / software, soft- desarrollo de software y desarrollo de hardware virtual Opción proceder simultáneamente a través de paso a paso descomposición. La parte de hardware suele ser simulado en arreglos de compuerta programables en campo (FPGA) o circuito integrado específico de la aplicación cuits (ASIC). La parte del software se traduce a Un lenguaje de programación de bajo nivel.

4.14. Análisis de rendimiento y ajuste

[1 *]

Eficiencia del código, determinada por la arquitectura, decisiones detalladas de diseño y estructura de datos y selección de algoritmo: influye en una ejecución Velocidad y tamaño. El análisis de rendimiento es la inversión Tigración del comportamiento de un programa utilizando información se recopila a medida que se ejecuta el programa, con el objetivo de identificar posibles puntos calientes en el pro- grama a mejorar.

Ajuste de código, que mejora el rendimiento en el nivel de código, es la práctica de modificar correcto código de manera que lo haga funcionar de manera más eficiente. El ajuste de código generalmente implica solo a pequeña escala cambios que afectan una sola clase, una sola rutina, o, más comúnmente, algunas líneas de código. Un rico El conjunto de técnicas de ajuste de código está disponible, incluyendo ing aquellos para ajustar expresiones lógicas, bucles, datos transformaciones, expresiones y rutinas. Utilizando un lenguaje de bajo nivel es otra tecnología común nique para mejorar algunos puntos calientes en un programa.

4.15. Estándares de plataforma

[6 *] [7 *]

Los estándares de la plataforma permiten a los programadores desarrollar aplicaciones portátiles que puedan ser ejecutables en entornos compatibles sin cambios Los estándares de la plataforma generalmente implican un conjunto de servicios estándar y API que son compatibles Las implementaciones de plataforma ible deben implementarse. Ejemplos típicos de estándares de plataforma son Java 2 Platform Enterprise Edition (J2EE) y el Estándar POSIX para sistemas operativos (portátil Interfaz del sistema operativo), que representa un conjunto de estándares implementados principalmente para Sistemas operativos basados en UNIX.

4.16. Prueba de primera programación

[1 *]

Programación de prueba primero (también conocida como Prueba- El desarrollo impulsado (TDD) es un desarrollo popular estilo de opción en el que los casos de prueba se escriben antes para escribir cualquier código. La primera prueba de programación puede generalmente detecta defectos antes y los corrige más fácilmente que los estilos de programación tradicionales. Además, escribir casos de prueba primero fuerza pro Gramers para pensar sobre los requisitos y el diseño. antes de codificar, exponiendo así los requisitos y problemas de diseño antes.

Page 77

5. Herramientas de construcción de software

5.3. Herramientas de prueba de unidad

5.1. Entornos de desarrollo [1 *]

Un entorno de desarrollo o desarrollo integrado. entorno de opciones (IDE), proporciona Instalaciones intensivas para programadores de software construcción integrando un conjunto de desarrollo herramientas. Las elecciones de los entornos de desarrollo. puede afectar la eficiencia y la calidad del software construcción.

Además de las funciones básicas de edición de código, Los IDE modernos a menudo ofrecen otras características como la Pilation y detección de errores desde el interior tor, integración con control de código fuente, construcción / herramientas de prueba / depuración, comprimidas o esquemáticas vistas de programas, transformaciones de código automatizadas y apoyo para refactorización.

5.2. Constructores de GUI [1 *]

Un generador de GUI (interfaz gráfica de usuario) es un herramienta de desarrollo de software que permite el desarrollador oper para crear y mantener GUI en un WYSI-Modo WYG (lo que ves es lo que obtienes). UNA El generador de GUI generalmente incluye un editor visual para que el desarrollador diseñe formularios y ventanas y administrar el diseño de los widgets arrastrando ging, droping y configuración de parámetros. Alguna GUI los constructores pueden generar automáticamente la fuente código correspondiente al diseño visual de la GUI.

Debido a que las aplicaciones GUI actuales generalmente siguen bajo el estilo basado en eventos (en el que el flujo de el programa está determinado por eventos y eventos manejo), las herramientas de creación de GUI generalmente proporcionan asistentes de generación de código, que automatizan tareas más repetitivas requeridas para el manejo de eventos. El código de soporte conecta widgets con el eventos salientes y entrantes que desencadenan la funciones que proporcionan la lógica de la aplicación.

Algunos IDEs modernos proporcionan GUI integrada constructores o complementos de constructor de GUI. También hay muchos constructores de GUI independientes.

[1 *] [2 *]

Las pruebas unitarias verifican el funcionamiento del software. módulos aislados de otros elementos de software que son comprobables por separado (por ejemplo, clases, rutinas, componentes). Las pruebas unitarias suelen ser auto-apareado Los desarrolladores pueden usar herramientas de prueba unitarias y marcos para ampliar y crear automatizados entorno de prueba Con herramientas de prueba de unidades y marcos, el desarrollador puede codificar criterios en la prueba para verificar la corrección de la unidad bajo vari-Ous conjuntos de datos. Cada prueba individual se implementa como un objeto, y un corredor de prueba ejecuta todas las pruebas. Durante la ejecución de la prueba, esos casos de prueba fallidos será marcado e informado automáticamente.

5.3. Perfiles, análisis de rendimiento y Herramientas de corte [1 *]

Las herramientas de análisis de rendimiento generalmente se utilizan para Soporte de ajuste de código. El más común per-Las herramientas de análisis de formance son herramientas de creación de perfiles. Un herramienta de perfiles de ejecución monitorea el código mientras se ejecuta y registra cuántas veces cada estado se ejecuta o cuánto tiempo dura el programa gasta en cada declaración o ruta de ejecución. Pro-presentar el código mientras se está ejecutando da una idea sobre cómo funciona el programa, dónde están los puntos calientes son, y donde los desarrolladores deberían enfocar esfuerzos de ajuste de código.

La segmentación de programas implica el cálculo de conjunto de declaraciones de programa (es decir, el segmento del programa) que puede afectar los valores de variables especificadas en algún punto de interés, que se conoce como Un criterio de corte. Se puede usar el corte de programa para localizar la fuente de errores, el programa sub-de pie y análisis de optimización. Programa las herramientas de corte calculan cortes de programa para varios lenguajes de programación usando estática o dinámica métodos de análisis

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

04				**02**	**06**	**08**
ell 20 norte en cC METRO [1 *]	**ile 2011 erv metro gm S** [2 *]	**ts et al. 2010 en lem do** [3 *]	**a. et al. 1994 metro a.m sol** [4 *]	**alcer 20 d B** [5 *] **ellor an METRO**	**r 20 tu d L** [6 *] **ll an tu norte**	**transmisión exterior atz y cd. 20 Ersch ilb S** [7 *]
1. Software Construcción Fundamentos						
c2, c3,						

1.1. Minimizando Complejidad	c7-c9, c24, c27, c28, c31, c32, c34								
1.2. Anticipando Cambio	c3 – c5, c24, c31, c32, c34								
1.3. Construyendo para Verificación	c8, c20–c23, c31, c34								
1.4. Reutilizar		c16							
1.5. Estándares en Construcción	c4								
2. Gestionar Construcción									
2.1. Construcción en Modelos de ciclo de vida	c2, c3, c27, c29								
2.2. Construcción Planificación	c3, c4, c21, c27 – c29								
2.3. Construcción Medición	c25, c28								
3. Práctico Consideraciones									
3.1. Construcción Diseño	c3, c5, c24								
3.2. Construcción Idiomas	c4								
3.3. Codificación	c5 – c19, c25 – c26								

Página 79

	04					**02**	**06**	**08**	
	ell 20 norte en cC METRO	**ile 2011 [1 *] erv metro gm S**	**ts et al. 2010 [2 *] en lem do**	**al et al. 1994 [3 *] metro a.m sol**	**[4 *]**	**alcer 20 d B [5 *] ellor an METRO**	**r 20 tu transmisión exterior d L [6 *] ll an tu norte**	**atz y col. 20 [7 *] Ersch ilb S**	
3.4. Construcción Pruebas	c22, c23								
3.5. Construcción para Reutilizar		c16							
3.6. Construcción con reutilización		c16							
3.7. Construcción Calidad	c8, c20 – c25								
3.8. Integración	c29								
4. construcción Tecnologías									
4.1. Diseño de API y Utilizar			c7						
4.2. Orientado a objetos Problemas de tiempo de ejecución	c6, c7								
4.3. Parametrización y genéricos				c1					

4.4. Aserciones, Diseño por contrato, y defensivo Programación	c8, c9							
4.5. Manejo de errores, Manejo de excepciones, y tolerancia a fallas	c3, c8							
4.6. Ejecutable Modelos					c1			
4.7. Basado en el estado y conducido por mesa Construcción Tecnicas	c18							
4.8. Tiempo de ejecución Configuración y Internacionalizacion	c3, c10							
4.9. Basada en gramática Procesamiento de entrada	c5					c8		

	04 ell 20 norte en cC METRO [1 *]	ile 2011 erv metro om S [2 *]	ts et al. 2010 en lem do	a[3 *] metro a.m sol	02 alcer 20 d B [5 *] ellor an METRO	06 r 20 tu transmisión exterior d L [6 *] ll an tu norte	08 atz y cl. 20 Ersch ilb S [7 *]
4.10. Concurrencia Primitivas							c6
4.11. Middleware			c1			c8	
4.12. Construcción Métodos para Software distribuido							c2
4.13. Construyendo Heterogéneo Sistemas						c9	
4.14. Actuación Análisis y afinación	c25, c26						
4.15. Plataforma Normas						c10	c1
4.16. Prueba primero Programación	c22						
5. Herramientas de construcción							
5.1. Desarrollo Ambientes	c30						
5.2. Constructores de GUI	c30						
5.3. Examen de la unidad Herramientas	c22	c8					
5.4. Perfilado, Actuación Análisis y Herramientas de corte	c25, c26						

Página 81

LECTURAS ADICIONALES

IEEE Std. Norma 1517-2010 para información Tecnología: vida del sistema y del software Procesos de ciclo: procesos de reutilización , IEEE, 2010 [8].

Esta norma especifica los procesos, actividades, y tareas que se aplicarán durante cada fase de la ciclo de vida del software para habilitar un producto de software para ser construido a partir de activos reutilizables. Cubre el concepto de desarrollo basado en la reutilización y el procesos de construccion para reutilizacion y construccion ción con reutilización.

IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Norma para sistemas y Ingeniería de software: ciclo de vida del software Procesos , IEEE, 2008 [9].

Este estándar define una serie de desarrollo de software procesos de opciones, incluida la construcción de software proceso de integración, proceso de integración de software y proceso de reutilización de software.

Referencias

[1 *] S. McConnell, *Código completo* , 2ª ed., Microsoft Press, 2004.

[2 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[3 *] P. Clements et al., *Software de documentación Arquitecturas: Vistas y más allá* , 2ª ed., Pearson Education, 2010.

[4 *] E. Gamma et al., *Patrones de diseño: elementos de software orientado a objetos reutilizables* , 1er ed., Addison-Wesley Professional, 1994.

[5 *] SJ Mellor y MJ Balcer, *ejecutable UML: una base para el modelo Arquitectura* , 1ª ed., Addison-Wesley, 2002.

[6 *] L. Null y J. Lobur, *Los fundamentos de Organización y arquitectura de computadoras* , 2ª ed., Jones and Bartlett Publishers, 2006

[7 *] A. Silberschatz, PB Galvin y G. Gagne, *Conceptos del sistema operativo* , 8ª ed., Wiley, 2008

[8] *IEEE Std. Norma 1517-2010 para Tecnología de la información: sistema y Procesos del ciclo de vida del software: reutilización Procesos* , IEEE, 2010.

[9] *IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Norma para sistemas y Ingeniería de software: ciclo de vida del software Procesos* , IEEE, 2008.

Page 82

CAPÍTULO 4

PRUEBAS DE SOFTWARE

SIGLAS

API	Interfaz del programa de aplicación
TDD	Desarrollo guiado por pruebas
TTCN3	Prueba y notación de control de prueba Versión 3
XP	Programación extrema

INTRODUCCIÓN

Las pruebas de software consisten en la verificación *dinámica* ción de que un programa proporciona comportamientos *esperados* en un conjunto *finito* de casos de prueba, adecuadamente *seleccionados*. El dominio de ejecución generalmente infinito.

En la definición anterior, las palabras en cursiva corresponden responder a problemas clave al describir el Software Área de conocimiento de prueba (KA):

- *Dinámico:* este término significa que las pruebas siempre implica ejecutar el programa en entradas seleccionadas Para ser precisos, la entrada el valor por sí solo no siempre es suficiente para especificar una prueba, ya que es compleja, no determinista el sistema puede reaccionar a la misma entrada con diferentes comportamientos, dependiendo del sistema estado. En este KA, sin embargo, el término "entrada" se mantendrá, con la conveniencia implícita Además, su significado también incluye una especi estado de entrada fied en aquellos casos para los que es importante. Las técnicas estáticas son diferentes. desde y complementario a las pruebas dinámicas. Las técnicas estáticas están cubiertas en el software Calidad KA. Vale la pena señalar que terminol- ogy no es uniforme entre diferentes comunidades algunas y algunos usan el término "prueba" también en referencia a técnicas estáticas.
- *Finito:* incluso en programas simples, muchas pruebas los casos son teóricamente posibles que agotan- Las pruebas tivas pueden requerir meses o años para ejecutar. Por eso, en la práctica, un completo conjunto de pruebas generalmente se puede considerar infi noche, y las pruebas se realizan en un subconjunto de todas las pruebas posibles, que están determinadas por el riesgo y criterios de priorización. Prueba siempre implica una compensación entre recursos limitados y horarios por un lado e inherentemente requisitos de prueba ilimitados por el otro.
- *Seleccionado: las* muchas técnicas de prueba propuestas las niques difieren esencialmente en cómo el conjunto de prueba está seleccionado, y los ingenieros de software deben ser consciente de que diferentes criterios de selección pueden producir grados muy diferentes de efectividad *mente*. Cómo identificar el más adecuado El criterio de selección en determinadas condiciones es un problema complejo en la práctica, análisis de riesgos técnicas y experiencia en ingeniería de software Tise se aplican.
- *Esperado:* debe ser posible, aunque no siempre fácil, decidir si lo observado los resultados de las pruebas del programa son aceptables o no; de lo contrario, el esfuerzo de prueba es de uso Menos. El comportamiento observado puede ser verificado contra las necesidades del usuario (comúnmente referido como prueba de validación), contra una especi fication (prueba para verificación), o, per- Haps, contra el comportamiento anticipado de requisitos o expectativas implícitas (ver Las pruebas de aceptación en el software requieren Ments KA).

En los últimos años, la vista de las pruebas de software ha madurado en uno constructivo. La prueba es ya no se ve como una actividad que comienza solo después la fase de codificación se completa con la limitada propósito de detectar fallas. Pruebas de software es, o debería ser, dominante en todo el Desarrollo y mantenimiento del ciclo de vida. En efecto, la planificación de las pruebas de software debe comenzar con el primeras etapas del proceso de requisitos de software,

Figura 4.1. Desglose de temas para el Software Testing KA

y los planes y procedimientos de prueba deben ser sistema-
desarrollada ática y continuamente, y posiblemente
bly refinado, a medida que avanza el desarrollo de software.
Estas actividades de planificación y diseño de pruebas
proporcionar información útil para diseñadores de software y
ayudar a resaltar posibles debilidades, como
descuidos / contradicciones de diseño u omisiones /
ambigüedades en la documentación.

Para muchas organizaciones, el enfoque de soft-
la calidad del material es una de prevención: obviamente es
mucho mejor para prevenir problemas que corregir
ellos. Las pruebas se pueden ver, entonces, como un medio para
proporcionar información sobre la funcionalidad
y atributos de calidad del software y también
para identificar fallas en aquellos casos donde el error
La prevención no ha sido efectiva. Es tal vez
obvio pero vale la pena reconocer que el software puede
todavía contienen fallas, incluso después de completar un
Amplia actividad de pruebas. Experto en fallas de software
retrasado después de la entrega se abordan por correctivo
mantenimiento. Los temas de mantenimiento de software son
cubierto en el mantenimiento de software KA.

En Software Quality KA (ver Software Qual-
ity Management Techniques), calidad de software
las técnicas de gestión se clasifican notablemente
en técnicas estáticas (sin ejecución de código) y
técnicas dinámicas (ejecución de código). Ambos gatos
Los egories son útiles. Este KA se centra en dinámica
técnicas

Las pruebas de software también están relacionadas con el software.
construcción (ver Pruebas de construcción en el
Software Construction KA). En particular, unidad
y las pruebas de integración están íntimamente relacionadas con
construcción de software, si no es parte de ella.

DESGLOSE DE TEMAS PARA PRUEBAS DE SOFTWARE

El desglose de temas para la Prueba de software
ing KA se muestra en la Figura 4.1. Un más detallado
el desglose se proporciona en la Matriz de temas
vs. Material de referencia al final de este KA.

El primer tema describe la prueba de software Fun-
Damentals. Cubre las definiciones básicas en el
campo de pruebas de software, la terminología básica
y cuestiones clave, y la relación de las pruebas de software
enviar con otras actividades.

El segundo tema, Niveles de prueba, consta de dos
subtemas (ortogonales): el primer subtema enumera los
niveles en los que la prueba de software grande es
tradicionalmente subdividido, y el segundo subtema

1. Fundamentos de pruebas de software

1.1. Terminología relacionada con pruebas

1.1.1. Definiciones de prueba y relacionadas Terminología

[1 *, c1, c2] [2 *, c8]

Definiciones de pruebas y términos relacionados con las pruebas
la nología se proporciona en las referencias citadas y
resumido de la siguiente manera.

1.1.2. Fallos vs. Fallos

[1 *, c1s5] [2 *, c11]

Se utilizan muchos términos en la ingeniería de software.
literatura para describir un mal funcionamiento: notablemente *culpa* ,
fracaso y *error,* entre otros. Esta terminología
ogy se define con precisión en [3, c2]. Es esencial
para distinguir claramente entre la *causa* de un mal
función (para la cual se usará el término falla)
aquí) y un efecto no deseado observado en el sistema
servicio prestado por tem (que se llamará
fracaso). De hecho, bien puede haber fallas en el
software que nunca se manifiesta como un fracaso

considera realizar pruebas para condiciones específicas o se refiere y se conoce como objetivos de prueba. No todos los tipos de pruebas se aplican a cada software. producto, ni todos los tipos posibles han sido listados.

El objetivo de prueba y el objetivo de prueba juntos determinar cómo se identifica el conjunto de prueba, ambos en cuanto a su consistencia, *la cantidad de pruebas suficiente para lograr el objetivo declarado* y a su composición, *qué casos de prueba deberían ser seleccionado para lograr el objetivo declarado* (aunque generalmente "para lograr el objetivo declarado- tive "permanece implícito y solo la primera parte de la se plantean dos preguntas en cursiva arriba). Criterios para abordar la primera pregunta se conocen como *prueba los criterios de adecuación* , mientras que los que abordan La segunda pregunta son los *criterios de selección de* la prueba.

Se han desarrollado varias técnicas de prueba. en las últimas décadas, y todavía hay otras nuevas siendo propuesto. Técnicas generalmente aceptadas están cubiertos en el tercer tema.

Las medidas relacionadas con la prueba se tratan en el cuarto tema, mientras que los problemas relacionados con Test Pro- los cess están cubiertos en el quinto. Finalmente el software Las herramientas de prueba se presentan en el tema seis.

uno (ver Limitaciones teóricas y prácticas de Pruebas en la sección 1.2, Temas clave). Así prueba ing puede revelar fallas, pero son las fallas las que pueden y debe ser eliminado [3]. El término más genérico *el defecto* se puede usar para referirse a una falla o fracaso, cuando la distinción no es importante [3].

Sin embargo, debe reconocerse que la causa de un fracaso no siempre puede ser inequívocamente identified. No existen criterios teóricos para definitivamente determinar, en general, la falla que causó un fracaso observado Se podría decir que fue el falla que tuvo que modificarse para eliminar la falla, pero otras modificaciones podrían haber funcionado solo también. Para evitar la ambigüedad, uno podría referirse a *errores que causan* fallas en lugar de fallas, es decir, esos conjuntos de entradas que provocan una falla en aparecer.

1.2. Cuestiones clave

1.2.1 Criterios de selección de prueba / adecuación de prueba Criterios (reglas de detención)

[1 *, c1s14, c6s6, c12s7]

Un criterio de selección de prueba es un medio para seleccionar casos de prueba o determinar que un conjunto de casos de prueba

es suficiente para un propósito específico. Prueba ade- Los criterios de calidad se pueden utilizar para decidir cuándo se realizarán o se han realizado pruebas de inteligencia [4] (ver Terminación en la sección 5.1, Práctico Consideraciones).

1.2.2. Prueba de efectividad / objetivos para Pruebas

[1 *, c11s4, c13s11]

La efectividad de la prueba se determina analizando Un conjunto de ejecuciones de programas. Selección de pruebas para ser ejecutado puede guiarse por diferentes objetivos: Es solo a la luz del objetivo perseguido que el Se puede evaluar la efectividad del conjunto de pruebas.

1.2.3 Prueba de detección de defectos

[1 *, c1s14]

Al probar el descubrimiento de defectos, una prueba exitosa es uno que hace que el sistema falle. Esto es bastante diferente de las pruebas para demostrar que el el software cumple con sus especificaciones u otro deseado propiedades, en cuyo caso la prueba es exitosa si no se observan fallas en casos de prueba realistas y entornos de prueba.

1.2.4. El problema de Oracle

[1 *, c1s9, c9s7]

Un oráculo es cualquier agente humano o mecánico que decide si un programa se comportó correctamente en una prueba dada y, en consecuencia, da como resultado un dict de "pasar" o "fallar". Existen muchas diferencias ent tipos de oráculos; por ejemplo, inequívoco especificaciones de requisitos, modelos de comportamiento, y anotaciones de código. Automatizacion de mecanizado Los oráculos pueden ser difíciles y costosos.

1.2.5 Limitaciones teóricas y prácticas de

a este respecto es el aforismo de Dijkstra que "pro- la prueba de gramo se puede utilizar para mostrar la presencia de errores, pero nunca para mostrar su ausencia "[5]. los La razón obvia de esto es que la prueba completa es No es factible en software realista. Debido a esto, las pruebas deben realizarse en función del riesgo [6, parte 1] y puede verse como una estrategia de gestión de riesgos.

1.2.6. El problema de los caminos inviables

[1 *, c4s7]

Las rutas inviables son rutas de flujo de control que no pueden ser ejercido por cualquier dato de entrada. Son un significado No puede ser un problema en las pruebas basadas en rutas, particularmente en derivación automatizada de entradas de prueba para ejercicio Controlar las rutas de flujo.

1.2.7. Testabilidad

[1 *, c17s2]

El término "comprobabilidad de software" tiene dos relacionados pero diferentes significados: por un lado, se refiere a la facilidad con que una cobertura de prueba dada criterio puede ser satisfecho; por otro lado, se define como la probabilidad, posiblemente medida estadísticamente, que un conjunto de casos de prueba expondrá una falla *si* el software es defectuoso. Ambos significados son importantes.

1.3. Relación de las pruebas con otras actividades

Las pruebas de software están relacionadas con, pero son diferentes de técnicas de gestión de calidad de software estático , pruebas de corrección, depuración y programa construcción. Sin embargo, es informativo prueba lateral desde el punto de vista del software analistas de calidad y certificadores.

- Pruebas versus software de calidad del software estático Técnicas de gestión (ver Calidad del software

Pruebas [1 *, c2s7]

La teoría de las pruebas advierte contra la atribución de una injustificada nivel de confianza fied a una serie de éxito pruebas Desafortunadamente, los resultados más establecidos de la teoría de prueba son negativas, en el sentido de que afirman qué pruebas nunca se pueden lograr en comparación con lo que se logra realmente La cita más famosa

Técnicas de gestión en el software Calidad KA [1 *, c12]).

• Pruebas versus pruebas de corrección y formales Verificación (ver la Ingeniería del Software Modelos y métodos KA [1 *, c17s2]).

• Pruebas versus depuración (ver Construcción Pruebas en la construcción de software KA y herramientas y técnicas de depuración en el Fundamentos de computación KA [1 *, c3s6]).

• Pruebas versus construcción del programa (ver Con- Pruebas de construcción en la construcción de software ción KA [1 *, c3s2]).

2. Niveles de prueba

Las pruebas de software generalmente se realizan en diferentes *Niveles* ent a lo largo del desarrollo y mantenimiento procesos de tenencia. Los niveles se pueden distinguir basado en el objeto de prueba, que se llama el *objetivo* , o con el propósito, que se llama *objetivo* (del nivel de prueba).

2.1. El objetivo de la prueba [1 *, c1s13] [2 *, c8s1]

El objetivo de la prueba puede variar: un solo módulo, un grupo de dichos módulos (relacionados por propósito, uso, comportamiento o estructura), o un sistema completo. Tres Las etapas de prueba se pueden distinguir: unidad, integración y sistema. Estas tres etapas de prueba no implica ningún modelo de proceso, ni ninguno de ellos se supone que es más importante que los otros dos.

2.1.1. Examen de la unidad [1 *, c3] [2 *, c8]

Las pruebas unitarias verifican el funcionamiento de forma aislada. de elementos de software que se pueden probar por separado. Dependiendo del contexto, estos podrían ser los subprogramas individuales o un componente más grande hecho de unidades altamente cohesivas. Típicamente, unidad la prueba ocurre con el acceso al código que se está probando y con el soporte de herramientas de depuración. El PROgrammers que escribieron el código típicamente, pero no siempre, realice pruebas unitarias.

2.1.2. Pruebas de integración [1 *, c7] [2 *, c8]

La prueba de integración es el proceso de verificar el interacciones entre componentes de software. Clasestrategias de prueba de integración sical, como topabajo y abajo, a menudo se usan con jerarquía Software estructurado químicamente.

Las estrategias de integración modernas y sistemáticas son típicamente impulsado por la arquitectura, que implica Integración incremental del software ponentes o subsistemas basados en identificados hilos funcionales Las pruebas de integración son a menudo un actividad continua en cada etapa de desarrollo durante el cual los ingenieros de software se abstraen perspectivas de nivel inferior y concentrarse en el perspectivas del nivel en el que son interejilla. Para otro software que no sea pequeño y simple, las estrategias de prueba de integración incremental son usuAlly prefirió poner todos los componentes juntos a la vez, lo que a menudo se llama "grande bang "pruebas.

2.1.3. Prueba de sistema [1 *, c8] [2 *, c8]

La prueba del sistema se refiere a la prueba de comportamiento de todo un sistema. Unidad efectiva y las pruebas de integración habrán identificado muchos de Los defectos del software. La prueba del sistema suele ser considerado apropiado para evaluar la no requisitos funcionales del sistema, tales como seguridad ridad, velocidad, precisión y fiabilidad (ver FuncRequisitos nacionales y no funcionales en el Requisitos de software KA y Software QualRequerimientos en la Calidad del Software KA). Interfaces externas a otras aplicaciones, utilidades, dispositivos de hardware o los entornos operativos También se suelen evaluar a este nivel.

2.2. Objetivos de la prueba [1 *, c1s7]

Las pruebas se realizan en vista de objetivos específicos bjetives, que se indican más o menos explícitamente y con diversos grados de precisión. Declarando Los objetivos de las pruebas en forma precisa y cuantitativa. los términos apoyan la medición y el control de la proceso de prueba

Las pruebas pueden estar dirigidas a verificar diferentes apoyos Erties. Los casos de prueba se pueden diseñar para verificar que las especificaciones funcionales se implementan correctamente mented, que se menciona de diversas maneras en la literatura prueba de conformidad, prueba de corrección ing, o pruebas funcionales. Sin embargo, varios otros las propiedades no funcionales también se pueden probar: incluyendo rendimiento, confiabilidad y usabilidad ity, entre muchos otros (ver Modelos y Calidad Características en la Calidad del Software KA).

Otros objetivos importantes para las pruebas incluyen pero no se limitan a la medición de confiabilidad,

Page 87

identificación de vulnerabilidades de seguridad, usabilidad evaluación y aceptación de software, para lo cual Se tomarían diferentes enfoques. Tenga en cuenta que, en general, los objetivos de la prueba varían con la prueba objetivo; diferentes propósitos se abordan en diferentes Niveles de prueba ent.

Los subtemas enumerados a continuación son los más a menudo citado en la literatura. Tenga en cuenta que algunos tipos de las pruebas son más apropiadas para personalizar paquetes de software: pruebas de instalación, para ejemplo, y otros para productos de consumo, como prueba beta

2.2.1. Pruebas de aceptación / calificación

[1 *, c1s7] [2 *, c8s4]

La prueba de aceptación / calificación determina si un sistema cumple sus criterios de aceptación, generalmente al verificar los comportamientos deseados del sistema en contra de los requisitos del cliente. El cliente Tomer o el representante de un cliente así especi vuela o realiza directamente actividades para verificar que se han cumplido sus requisitos, o en el caso de un producto de consumo, que la organización ha satisfecho los requisitos establecidos para la tar- obtener mercado. Esta actividad de prueba puede o no involucrar a los desarrolladores del sistema.

2.2.2. Prueba de instalación

[1 *, c12s2]

A menudo, después de completar el sistema y aceptarlo prueba, el software se verifica después de la instalación en el entorno objetivo La prueba de instalación puede ser visto como prueba del sistema realizada en el entorno operativo de configuración de hardware iones y otras restricciones operacionales. Instala- También se pueden verificar los procedimientos.

2.2.3. Pruebas alfa y beta

[1 *, c13s7, c16s6] [2 *, c8s4]

Antes de lanzar el software, a veces se administra a un pequeño grupo seleccionado de usuarios potenciales para uso de prueba (prueba *alfa*) y / o para un conjunto mayor de usuarios representativos (prueba *beta*). Estos usuarios Informar problemas con el producto. Alfa y beta las pruebas a menudo no están controladas y no siempre mencionado en un plan de prueba.

2.2.4. Logro y evaluación de confiabilidad

[1 *, c15] [2 *, c15s2]

Las pruebas mejoran la fiabilidad al identificar y corrigiendo fallas. Además, medidas estadísticas de fiabilidad puede derivarse generando aleatoriamente ing casos de prueba de acuerdo con el perfil operativo de el software (consulte Perfil operativo en la sección 3.5, Técnicas basadas en el uso). El último enfoque es llamado *prueba operacional* . Usando el crecimiento de confiabilidad modelos, ambos objetivos pueden perseguirse juntos [3] (ver Prueba de *L* ife, Evaluación de confiabilidad en la sección 4.1, Evaluación del programa bajo prueba).

2.2.5. Pruebas de regresión

[1 *, c8s11, c13s3]

Según [7], la prueba de regresión es la "selección sistema prueba de un sistema o componente para verificar que las modificaciones no han causado involuntariamente efectos y que el sistema o componente todavía cumple con los requisitos especificados ". práctica, el enfoque es mostrar que el software todavía pasa las pruebas previamente aprobadas en un conjunto de pruebas (de hecho, a veces también se le conoce como nonre- prueba de gression). Para el desarrollo incremental, El propósito de las pruebas de regresión es mostrar que el comportamiento del software no cambia por incremen- Tal cambios en el software, excepto en la medida en que debería. En algunos casos, se debe hacer una compensación entre la garantía dada por las pruebas de regresión cada vez que se realiza un cambio y los recursos requerido para realizar las pruebas de regresión, que puede llevar mucho tiempo debido a la gran Número de pruebas que pueden ejecutarse. Regresión las pruebas implican seleccionar, minimizar y / o priorizar un subconjunto de los casos de prueba en un conjunto de pruebas ing [8]. Las pruebas de regresión pueden ser con- conducido en cada uno de los niveles de prueba descritos en la sección sección 2.1, El objetivo de la prueba, y puede aplicarse a Pruebas funcionales y no funcionales.

2.2.6. Pruebas de rendimiento

[1 *, c8s6]

Las pruebas de rendimiento verifican que el software cumple los requisitos de rendimiento especificados y evalúa las características de rendimiento para instancia, capacidad y tiempo de respuesta.

Page 88

2.2.7. Pruebas de seguridad

[1 *, c8s3] [2 *, c11s4]

Las pruebas de seguridad se centran en la verificación de que El software está protegido de ataques externos. En en particular, las pruebas de seguridad verifican la confianza- Tialidad, integridad y disponibilidad de los sistemas. y sus datos. Por lo general, las pruebas de seguridad incluyen

2.2.12. Prueba de configuración

[1 *, c8s5]

En casos donde el software está construido para servir diferentes usuarios, las pruebas de configuración verifican el software bajo diferentes configuraciones especificadas.

2.2.13. Usabilidad e interfase de computadora humana

verificación contra el mal uso y abuso del software Ware o sistema (prueba negativa).

2.2.8. Pruebas de estrés

[1 *, c8s8]

Software de ejercicios de prueba de esfuerzo al máximo carga de diseño, así como más allá, con el objetivo de determinar los límites de comportamiento y probar mecanismos de defensa en sistemas críticos.

2.2.9. Pruebas consecutivas

[7]

El estándar IEEE / ISO / IEC 24765 define el regreso a prueba posterior como "prueba en la que dos o más Las variantes de un programa se ejecutan con el mismo entradas, las salidas se comparan y los errores son analizado en caso de discrepancias ".

2.2.10. Prueba de recuperación

[1 *, c14s2]

Las pruebas de recuperación tienen como objetivo verificar el software reiniciar las capacidades después de un bloqueo del sistema de la "desastre."

2.2.11. Prueba de interfaz

[2 *, c8s1.3] [9 *, c4s4.5]

Los defectos de interfaz son comunes en sistemas complejos Tems. La prueba de interfaz tiene como objetivo verificar si la interfaz de componentes correctamente para proporcionar el intercambio correcto de datos e información de control ción Por lo general, los casos de prueba se generan a partir de La especificación de la interfaz. Un objetivo específico de la prueba de interfaz es simular el uso de API por aplicaciones de usuario final. Esto involucra a los géneros- ción de parámetros de las llamadas API, la configuración de condiciones ambientales externas, y la defini ción de datos internos que afectan a la API.

Prueba de acción

[10 *, c6]

La tarea principal de usabilidad y computadora humana la prueba de interacción es evaluar qué tan fácil es para que los usuarios finales aprendan y usen el software. En en general, puede implicar probar las funciones del software opciones que soportan tareas del usuario, documentación que ayuda a los usuarios y la capacidad del sistema para recuperarse de errores de usuario (consulte Diseño de interfaz de usuario en Diseño de software KA).

3. Técnicas de prueba

Uno de los objetivos de las pruebas es detectar tantas fallas como sea posible. Muchas técnicas han sido desarrollado para hacer esto [6, parte 4]. Estas tecnicas intentar "romper" un programa siendo tan sys- temático como sea posible en la identificación de entradas que producir comportamientos representativos del programa; para instancia, al considerar las subclases de la entrada dominio, escenarios, estados y flujos de datos.

La clasificación de las técnicas de prueba pre presentado aquí se basa en cómo se generan las pruebas: de la intuición y experiencia del ingeniero de software rience, las especificaciones, la estructura del código, el fallas reales o imaginarias por descubrir, predecir uso, modelos o la naturaleza de la aplicación. Una categoría se ocupa del uso combinado de dos o más técnicas.

Algunas veces estas técnicas se clasifican como *caja blanca* (también llamada *caja de cristal*), si las pruebas son basado en información sobre cómo el software tiene diseñado o codificado, o como *recuadro negro* si la prueba los casos se basan solo en el comportamiento de entrada / salida de El software. La siguiente lista incluye aquellos técnicas de prueba que se usan comúnmente, pero algunos practicantes confían en algunas de las técnicas demás que otros.

Page 89

3.1. Basado en la intuición del ingeniero de software y experiencia

3.1.1. Ad hoc

Quizás la técnica más practicada es pruebas ad hoc: las pruebas se derivan basándose en habilidad, intuición y experiencia del ingeniero de software ence con programas similares. Las pruebas ad hoc pueden ser útil para identificar casos de pruebas que no son fáciles generado por técnicas más formalizadas.

3.1.2. Prueba exploratoria

Las pruebas exploratorias se definen como simultáneas aprendizaje, diseño de prueba y ejecución de prueba [6, parte 1]; es decir, las pruebas no están definidas de antemano en un plan de prueba establecido, pero son dinámicamente diseñado, ejecutado y modificado. El efectivo Las pruebas exploratorias dependen del software. conocimiento del ingeniero, que se puede derivar de diversas fuentes: comportamiento observado del producto en lugar de considerar todas las combinaciones posibles. La prueba por pares pertenece a la prueba combinatoria , que en general también incluye empresas de nivel superior binaciones que pares: estas técnicas son referidas como *t-sabio* , por el cual todas las combinaciones posibles de *t* variables de entrada se considera.

3.2.3. Análisis de valor límite

[1 *, c9s5]

Los casos de prueba se eligen en o cerca de los límites de el dominio de entrada de variables, con el subyacente justificación de que muchas fallas tienden a concentrarse cerca de los valores extremos de las entradas. Una extensión de Esta técnica es la prueba de robustez, en la que la prueba los casos también se eligen fuera del dominio de entrada de variables para probar la solidez del programa en el procesamiento Entradas inesperadas o erróneas.

3.2.4. Pruebas aleatorias

[1 *, c9s7]

durante las pruebas, familiaridad con la aplicación, la plataforma, el proceso de falla, el tipo de posposibles fallas y fallas, el riesgo asociado con un producto particular, y así sucesivamente.

3.2. Técnicas de entrada basadas en el dominio

3.2.1. Partición de equivalencia

[1 *, c9s4]

La partición de equivalencia implica la partición de dominio de entrada en una colección de subconjuntos (o equivalencia clases alentes) basadas en un criterio específico o relativo ción Este criterio o relación puede ser diferente. resultados computacionales, una relación basada en el control flujo o flujo de datos, o una distinción hecha entre entradas válidas que son aceptadas y procesadas por sistema y entradas no válidas, como valores fuera de rango ues, que no son aceptadas y deberían generar un mensaje de error o iniciar el procesamiento de error. Una repreconjunto representativo de pruebas (a veces solo una) es usu Aliado tomado de cada clase de equivalencia.

3.2.2. Prueba por pares

[1 *, c9s3]

Los casos de prueba se derivan combinando interesantes valores para cada par de un conjunto de variables de entrada

Las pruebas se generan exclusivamente al azar (no para ser confundido con pruebas estadísticas de la opera- Perfil nacional, como se describe en Perfil operativo en la sección 3.5). Esta forma de prueba cae bajo el encabezado de prueba de dominio de entrada desde la entrada el dominio debe ser conocido para poder elegir puntos aleatorios dentro de ella. Las pruebas aleatorias proporcionan un enfoque relativamente simple para la automatización de pruebas; recientemente, las formas mejoradas de pruebas aleatorias tienen propuesto en el que la entrada aleatoria sam- el pling está dirigido por otros criterios de selección de entrada [1] La prueba de fuzz o fuzzing es una forma especial de pruebas aleatorias destinadas a romper el software; eso se usa con mayor frecuencia para pruebas de seguridad.

3.3. Técnicas basadas en códigos

3.3.1 Control de criterios basados en el flujo

[1 *, c4]

Los criterios de cobertura basados en el flujo de control están dirigidos al cubrir todas las declaraciones, bloques de estado menciones o combinaciones específicas de declaraciones en un programa El más fuerte del flujo de control El criterio basado es la prueba de ruta, cuyo objetivo es ejecutar todas las rutas de flujo de control de entrada a salida en un Gráfico de flujo de control del programa. Ya que exhaustiva la prueba de ruta generalmente no es factible debido a

Página 90

bucles, otros criterios menos estrictos se centran en la cobertura borrado de rutas que limitan las iteraciones de bucle como cobertura de estado de cuenta, cobertura de sucursal y prueba de decisión / decisión. La adecuación de tales las pruebas se miden en porcentajes; por ejemplo, cuando todas las ramas se han ejecutado al menos una vez por las pruebas, el 100% de cobertura de sucursal tiene se ha logrado

3.3.2. Criterios basados en el flujo de datos

[1 *, c5]

En las pruebas basadas en el flujo de datos, el gráfico de flujo de control se anota con información sobre cómo el las variables del programa se definen, usan y eliminan (indefinido) El criterio más fuerte, todo definido rutas de uso, requiere que, para cada variable, cada segmento de ruta de flujo de control desde un definición de esa variable a un uso de esa definición es ejecutado. Para reducir el número de caminos se requieren estrategias más débiles, como todas las definiciones y todos los usos son empleados.

3.3.3 Modelos de referencia para código basado Pruebas

[1 *, c4]

Aunque no es una técnica en sí misma, el control la estructura de un programa puede representarse gráficamente resentido usando un diagrama de flujo para visualizar códigos técnicas de prueba basadas. Un diagrama de flujo es un gráfico dirigido, cuyos nodos y arcos corresponden responder a los elementos del programa (ver Gráficos y Árboles en los Fundamentos Matemáticos KA). Por ejemplo, los nodos pueden representar declaraciones o secuencias ininterrumpidas de enunciados y arcos

3.4.1. Error Adivinando

[1 *, c9s8]

En el error de adivinar, los casos de prueba son específicamente diseñado por ingenieros de software que intentan antici pate las fallas más plausibles en un programa dado. Una buena fuente de información es la historia de fallas descubiertas en proyectos anteriores, así como La experiencia del ingeniero de software.

3.4.2. Prueba de mutación

[1 *, c3s5]

Un mutante es una versión ligeramente modificada del programa bajo prueba, que difiere de un pequeño cambio sintáctico Cada caso de prueba ejercita ambos El programa original y todos los mutantes generados: si un caso de prueba tiene éxito en identificar la diferencia ferencia entre el programa y un mutante, el se dice que este último es "asesinado". Originalmente concebido como una técnica para evaluar conjuntos de pruebas (ver sección 4.2. Evaluación de las pruebas realizadas), muta- La prueba de acción también es un criterio de prueba en sí mismo: cualquiera de las pruebas se genera aleatoriamente hasta que sea suficiente los mutantes han sido asesinados, o las pruebas son específicamente diseñado para matar mutantes sobrevivientes. En lo ultimo caso, las pruebas de mutación también se pueden clasificar como Una técnica basada en código. El supuesto subyacente ción de la prueba de mutación, el efecto de acoplamiento, es que al buscar fallas sintácticas simples, Se encontrarán fallas más complejas pero reales. por la técnica para ser efectiva, una gran cantidad de los mutantes deben generarse automáticamente y ejecutado de manera sistemática [12].

3.5. Técnicas basadas en el uso

puede representar la transferencia de control entre nodos

3.4. Técnicas basadas en fallas

[1 *, c1s14]

Con diferentes grados de formalización, culpa-las técnicas de prueba basadas diseñan casos de prueba específicamente dirigido a revelar categorías de probable o fallas predefinidas. Para enfocar mejor el caso de prueba generación o selección, un *modelo de falla* puede ser introducido que clasifica los diferentes tipos de fallas

3.5.1. Perfil operacional

[1 *, c15s5]

En pruebas de evaluación de confiabilidad (también llamada prueba operativa), el entorno de prueba repro-reduce el entorno operativo del software Ware, o el *perfil operativo* , tan de cerca como posible. El objetivo es inferir de lo observado resultados de la prueba la fiabilidad futura del software cuando en uso real. Para hacer esto, se asignan entradas probabilidades, o perfiles, de acuerdo con sus frecuencias frecuencia de ocurrencia en la operación real. Ópera-Los perfiles nacionales se pueden utilizar durante las pruebas del sistema.

para guiar la derivación de casos de prueba que evaluarán el logro de objetivos de fiabilidad y ejercer el uso relativo y la criticidad de diferentes funciones similares a las que se encontrarán en El entorno operativo [3].

3.5.2. Heurística de observación del usuario

[10 *, c5, c7]

Los principios de usabilidad pueden proporcionar pautas para cubriendo problemas en el diseño del usuario inter-face [10 *, c1s4] (consulte Diseño de interfaz de usuario en Diseño de software KA). Heurística especializada, también llamados métodos de inspección de usabilidad, se aplican para la observación sistemática del uso del sistema bajo condiciones controladas para disuadir mina qué tan bien la gente puede usar el sistema y su interfaces La heurística de usabilidad incluye cognitiva tutoriales, análisis de reclamos, observaciones de campo, pensando en voz alta, e incluso enfoques indirectos como como cuestionarios de usuarios y entrevistas.

3.6. Técnicas de prueba basadas en modelos

Un modelo en este contexto es un resumen (formal) representación del software bajo prueba o de sus requisitos de software (ver Modelado en el Modelos y métodos de ingeniería de software KA). Las pruebas basadas en modelos se utilizan para validar los mentores, verificar su consistencia y generar pruebas casos centrados en los aspectos conductuales de la software. Los componentes clave de modelos basados las pruebas son [13]: la notación utilizada para representar el modelo del software o sus requisitos; trabajo-modelos de flujo o modelos similares; la estrategia de prueba o algoritmo utilizado para la generación de casos de prueba; la infraestructura de soporte para la ejecución de la prueba; y la evaluación de los resultados de la prueba en comparación con Resultados previstos. Debido a la complejidad de la técnicas, enfoques de prueba basados en modelos a menudo se utilizan junto con pruebas de automatización arneses Técnicas de prueba basadas en modelos. Incluya lo siguiente.

3.6.1. Tablas de decisiones

[1 *, c9s6]

Las tablas de decisiones representan relaciones lógicas. entre condiciones (aproximadamente, entradas) y acciones (aproximadamente, salidas). Los casos de prueba son sistemáticamente derivado al considerar todas las combinaciones posibles ción de condiciones y sus resultados correspondientes Tant acciones. Una técnica relacionada es *causa-efecto graficando* [1 *, c13s6].

3.6.2. Máquinas de estado finito

[1 *, c10]

Al modelar un programa como una máquina de estados finitos, Las pruebas se pueden seleccionar para cubrir los estados y transiciones.

3.6.3. Especificaciones formales

[1 *, c10s11] [2 *, c15]

Declarando las especificaciones en un lenguaje formal (ver Métodos formales en el ingeniero de software-ing Modelos y Métodos KA) permite automático derivación de casos de prueba funcionales y, en el Al mismo tiempo, proporciona un oráculo para comprobar la prueba resultados.

TTCN3 (Prueba y notación de control de prueba versión 3) es un lenguaje desarrollado para la prueba de escritura casos. La notación fue concebida para el específico necesidades de probar sistemas de telecomunicaciones, por lo que es particularmente adecuado para probar complejos complejos protocolos de comunicación requisitos

3.6.4. Modelos de flujo de trabajo

[2 *, c8s3.2, c19s3.1]

Los modelos de flujo de trabajo especifican una secuencia de activi-vínculos realizados por humanos y / o aplicaciones de software caciones, generalmente representados a través de gráficos notaciones Cada secuencia de acciones constituye un flujo de trabajo (también llamado escenario). Ambos tipos los flujos de trabajo cal y alternativos deben ser probados [6, parte 4]. Un enfoque especial en los roles en un trabajo la especificación de flujo está dirigida en el proceso comercial pruebas.

3.7. Técnicas basadas en la naturaleza de la Solicitud

Las técnicas anteriores se aplican a todo tipo de soft-mercancía. Técnicas adicionales para derivación de prueba y la ejecución se basan en la naturaleza del software mercancías que se prueban; por ejemplo,

Página 92

- software orientado a objetos
- software basado en componentes
- software basado en la web
- programas concurrentes
- software basado en protocolos
- sistemas en tiempo real
- sistemas críticos para la seguridad
- software orientado a servicios
- software de código abierto
- software incorporado

3.8. Selección y combinación de técnicas

3.8.1. Combinando funcional y estructural

[1 *, c9]

Técnicas de prueba basadas en modelos y en códigos. a menudo se contrastan como funcionales versus estructurales pruebas. Estos dos enfoques para probar la selección no deben verse como alternativas sino como complementos de hecho, usan diferentes fuentes de información y se ha demostrado que enciende diferentes tipos de problemas. Ellos pueden ser usado en combinación, dependiendo del presupuesto consideraciones

3.8.2. Determinista vs.Aleatorio

[1 *, c9s6]

Los casos de prueba se pueden seleccionar de forma determinista de acuerdo con una de muchas técnicas, o dominado de alguna distribución de insumos, como se suele hacer en las pruebas de confiabilidad. Sev Las comparaciones analíticas y empíricas tienen llevado a cabo para analizar las condiciones que hacer que un enfoque sea más efectivo que el otro.

4. Medidas relacionadas con la prueba

Algunas veces las técnicas de prueba se confunden con objetivos de prueba. Las técnicas de prueba pueden ser visto como ayudas que ayudan a asegurar el logro mento de los objetivos de la prueba [6, parte 4]. Por ejemplo, la cobertura de sucursales es una técnica de prueba popular. Lograr una medida de cobertura de sucursal especificada (p. ej., 95% de cobertura de sucursal) no debe ser el objetivo de la prueba per se: es una forma de mejora Las posibilidades de encontrar fallas al intentar ejercitar sistemáticamente cada rama del programa en cada punto de decisión. Para evitar tales errores entendiendo, se debe hacer una distinción clara entre medidas relacionadas con pruebas que proporcionan un evaluación del programa bajo prueba, basado en las salidas de prueba observadas y las medidas que evaluar la minuciosidad del conjunto de prueba. (Ver Medición de Ingeniería de Software en Soft-Ware Engineering Management KA para información sobre programas de medición. Ver software Medición de procesos y productos en Soft-Ware Engineering Process KA para obtener información sobre medidas)

La medición generalmente se considera fundamental. Tal como el análisis de calidad. La medición también puede ser utilizado para optimizar la planificación y ejecución de los exámenes. La gestión de pruebas puede usar varias medidas de proceso ent para monitorear el progreso. (Ver sección 5.1, Consideraciones prácticas, para una discussion de medidas del proceso de prueba util para fines de gestión.)

4.1. Evaluación del programa bajo prueba

4.1.1 Programa de medidas que ayudan en Planificación y diseño de pruebas

[9 *, c11]

Medidas basadas en el tamaño del software (por ejemplo, líneas de código fuente o tamaño funcional; ver Mea-Requisitos emergentes en el Software Requisitos-Ments KA) o en la estructura del programa se puede utilizar para guiar las pruebas. Las medidas estructurales también incluyen mediciones que determinan la frecuencia con qué módulos se llaman entre sí.

4.1.2. Tipos de fallas, clasificación y Estadística

[9 *, c4]

La literatura de pruebas es rica en clasificaciones y taxonomías de fallas. Para hacer las pruebas más efectivas Sin embargo, es importante saber qué tipos de fallas se puede encontrar en el software bajo prueba y el frecuencia relativa con la que tienen estas fallas ocurrió en el pasado. Esta información puede ser utilizada pleno en hacer predicciones de calidad, así como en mejora del proceso (ver Caracterización de defectos-ción en el Software Quality KA).

Página 93

4.1.3. Densidad de falla

[1 *, c13s4] [9 *, c4]

Un programa bajo prueba se puede evaluar contando

4.2.2 Siembra de fallas

[1 *, c2s5] [9 *, c6]

En la siembra de fallas, algunas fallas son introducidas artificialmente

fallas descubiertas como la relación entre el número de fallas encontradas y el tamaño del programa.

4.1.4. Prueba de vida, evaluación de confiabilidad

[1 *, c15] [9 *, c3]

Una estimación estadística de la fiabilidad del software. que se puede obtener observando fiabil- Si se logra, se puede usar para evaluar un software producto y decidir si las pruebas pueden o no ser detenido (ver sección 2.2, Logro de confiabilidad y evaluación).

4.1.5. Modelos de crecimiento de confiabilidad

[1 *, c15] [9 *, c8]

Los modelos de crecimiento de confiabilidad proporcionan una predicción de fiabilidad basada en fallas. Asumen, en general eral, que cuando las fallas que causaron lo observado las fallas han sido reparadas (aunque algunos modelos también acepta correcciones imperfectas), el producto estimado La confiabilidad de uct exhibe, en promedio, un aumento tendencia. Hay muchos crecimientos de confiabilidad publicados modelos. En particular, estos modelos se dividen en modelos de *conteo de fallas* y *tiempo entre fallas* .

4.2. Evaluación de las pruebas realizadas

4.2.1 Medidas de cobertura / minuciosidad

[9 *, c11]

Varios criterios de adecuación de la prueba requieren que la prueba casos ejercen sistemáticamente un conjunto de elementos identificado en el programa o en las especificaciones (Ver tema 3, Técnicas de prueba). Para evaluar el exhaustividad de las pruebas ejecutadas, ingeniería de software- neers pueden monitorear los elementos cubiertos para que pueden medir dinámicamente la relación entre elementos cubiertos y el número total. Para examen- ple, es posible medir el porcentaje de ramas cubiertas en el diagrama de flujo del programa o el porcentaje de requisitos funcionales ejercidos entre los que figuran en el documento de especificaciones. Los criterios de adecuación basados en el código requieren una adecuada instrumentación del programa bajo prueba.

introducido en un programa antes de la prueba. Cuando el se ejecutan pruebas, algunas de estas fallas sembradas ser revelado así como, posiblemente, algunas fallas que Ya estaban allí. En teoría, dependiendo de qué y cuántas de las fallas artificiales son descubiertas Se puede evaluar la efectividad de la prueba y se puede estimar el número restante de fallas genuinas apareado En la práctica, los estadísticos cuestionan la dis- contribución y representatividad de fallas sembradas en relación con fallas genuinas y el pequeño tamaño de muestra en el que se basan las extrapolaciones. Algunos tambien argumentan que esta técnica debe usarse con gran cuidado ya que la inserción de fallas en el software implica El riesgo obvio de dejarlos allí.

4.2.3 Puntuación de mutación

[1 *, c3s5]

En las pruebas de mutación (ver Pruebas de mutación en la sección sección 3.4, Técnicas basadas en fallas), la relación de mutantes muertos al número total de generados los mutantes pueden ser una medida de la efectividad de El conjunto de prueba ejecutado.

4.2.4 Comparación y efectividad relativa de diferentes técnicas

Se han realizado varios estudios para reducir la efectividad relativa de diferentes pruebas técnicas Es importante ser preciso en cuanto a propiedad contra la cual las técnicas están siendo probado; cuál es, por ejemplo, el significado exacto dado al término "efectividad"? Posible inter- las pretaciones incluyen la cantidad de pruebas necesarias para encontrar la primera falla, la razón del número de fallas encontradas a través de la prueba de todas las fallas encontradas durante y después de las pruebas, y cuánta fiabilidad ity fue mejorado. Análisis analítico y empírico parisones entre diferentes técnicas han sido realizado de acuerdo con cada una de las nociones de efectividad especificada anteriormente.

5. Proceso de prueba

Prueba de conceptos, estrategias, técnicas y medidas Las necesidades deben integrarse en un sistema definido y

proceso controlado El proceso de prueba admite prueba- ing actividades y proporciona orientación a los probadores y equipos de prueba, desde la planificación de la prueba hasta la evaluación, de tal manera que proporcione aseguramiento que los objetivos de la prueba se cumplirán de manera rentable- Tive camino.

5.1. Consideraciones prácticas

5.1.1 Actitudes / Programación sin ego

[1 * c16] [9 *, c15]

Un elemento importante de la prueba exitosa es un actitud colaborativa hacia las pruebas y la calidad actividades de aseguramiento. Los gerentes tienen un papel clave que desempeñar fomentando una recepción generalmente favorable hacia descubrimiento y corrección de fallas durante el software desarrollo y mantenimiento; por ejemplo, por

El artículo de prueba. La documentación de la prueba debe ser pro duced y continuamente actualizado al mismo nivel de calidad como otros tipos de documentación en Ingeniería de software. La documentación de prueba debe también estar bajo el control de la configuración del software gestión de la acción (ver la Configuración del software Gestión KA). Además, documentación de prueba incluye productos de trabajo que pueden proporcionar material para manuales de usuario y capacitación de usuarios.

5.1.5. Desarrollo guiado por pruebas

[1 *, c1s16]

El desarrollo basado en pruebas (TDD) se originó como uno de las prácticas principales de XP (programación extrema) y consiste en escribir pruebas unitarias antes de escribir el código a probar (ver Métodos ágiles en el Modelos de ingeniería de software y método KA).

superar la mentalidad del código individual propio
ership entre programadores y promoviendo un
ambiente colaborativo con responsabilidad del equipo
ity por anomalías en el código.

5.1.2. Guías de prueba

[1 *, c12s1] [9 *, c15s1]

Las fases de prueba pueden ser guiadas por varios objetivos: por ejemplo, las pruebas basadas en el riesgo utilizan riesgos del producto para priorizar y enfocar la estrategia de prueba egy, y las pruebas basadas en escenarios definen casos de prueba basado en escenarios de software especificados.

5.1.3. Prueba de gestión de procesos

[1 *, c12] [9 *, c15]

Actividades de prueba realizadas a diferentes niveles (ver tema 2, Niveles de prueba) deben organizarse juntos con personas, herramientas, políticas y medidas, en un proceso bien definido que es una parte integral de la ciclo vital.

5.1.4. Documentación de prueba y productos de trabajo

[1 *, c8s12] [9 *, c4s5]

La documentación es una parte integral de la formalización. ción del proceso de prueba [6, parte 3]. Documentos de prueba puede incluir, entre otros, el plan de prueba, prueba especificación de diseño, especificación de procedimiento de prueba, especificación de caso de prueba, registro de prueba e incidente de prueba informe. El software bajo prueba se documenta como

De esta manera, TDD desarrolla los casos de prueba como una solución rogate para una especificación de requisitos de software documento en lugar de como un cheque independiente que el software ha implementado correctamente el requisitos En lugar de una estrategia de prueba, TDD es una práctica que requiere que los desarrolladores de software definir y mantener pruebas unitarias; así también puede tener un impacto positivo en la elaboración de las necesidades del usuario y especificaciones de requisitos de software.

5.1.6. Equipo de prueba interno versus independiente

[1 *, c16]

Formalizar el proceso de prueba también puede implicar formalizando la organización del equipo de prueba. El equipo de evaluación puede estar compuesto por personal interno. miembros (es decir, en el equipo del proyecto, involucrados o no en la construcción de software), de miembros externos (con la esperanza de traer un imparcial, independiente perspectiva), o de miembros internos y externos Bers. Consideraciones de costo, calendario, vencimiento niveles de las organizaciones involucradas y criticidad de la aplicación puede guiar la decisión.

5.1.7. Estimación de costo / esfuerzo y proceso de prueba Medidas

[1 *, c18s3] [9 *, c5s7]

Varias medidas relacionadas con los recursos gastados. en las pruebas, así como en la búsqueda de fallas relativas efectividad de las diversas fases de prueba, se utilizan por los gerentes para controlar y mejorar las pruebas

Page 95

proceso. Estas medidas de prueba pueden cubrir tales aspectos como número de casos de prueba especificados, número ber de casos de prueba ejecutados, número de casos de prueba aprobado, y el número de casos de prueba falló, entre otros.

La evaluación de los informes de la fase de prueba puede realizarse combinado con análisis de causa raíz para evaluar la prueba efectividad del proceso en la búsqueda de fallas ya posible. Tal evaluación puede estar asociada con el análisis de riesgos. Por otra parte, los recursos que vale la pena gastar en las pruebas deben ser comp mensurate con el uso / criticidad de la aplicación ción: diferentes técnicas tienen diferentes costos y producir diferentes niveles de confianza en el producto confiabilidad.

5.1.8. Terminación

[9 *, c10s4]

Se debe tomar una decisión sobre cuánto examen ing es suficiente y cuando una etapa de prueba puede terminar. Nated Medidas de minuciosidad, como las logradas cobertura de código o cobertura funcional, así como estimaciones de densidad de fallas o de reli operacional capacidad, proporcionar apoyo útil pero no son suficientes cient en sí mismos. La decisión también implica consideraciones sobre los costos y riesgos incurridos por posibles fallas restantes, en oposición a los costos incurridos al continuar con la prueba (ver Prueba Criterios de selección / Criterios de adecuación de prueba en sección 1.2, cuestiones clave).

5.2. Actividades de prueba

Como se muestra en la siguiente descripción, exitoso La gestión de las actividades de prueba depende en gran medida en el programa de gestión de configuración de software realizado (consulte la Gestión de configuración de software- ment KA).

5.2.1. Planificación

[1 *, c12s1, c12s8]

Como todos los demás aspectos de la gestión de proyectos, Las actividades de prueba deben ser planificadas. Aspectos clave La planificación de la prueba incluye la coordinación de la persona nel, disponibilidad de instalaciones y equipos de prueba, creación y mantenimiento de todos los documentos relacionados con la prueba mentación y planificación para posibles indeseos Resultados capaces. Si hay más de una línea de base del se mantiene el software, entonces un plan importante Ninguna consideración es el tiempo y el esfuerzo necesarios para garantizar que el entorno de prueba esté configurado en configuración adecuada

5.2.2. Generación de casos de prueba

[1 *, c12s1, c12s3]

La generación de casos de prueba se basa en el nivel de pruebas a realizar y la prueba particular técnicas Los casos de prueba deben estar bajo el control control de la gestión de configuración de software y incluya los resultados esperados para cada prueba.

5.1.9. Prueba de reutilización y patrones de prueba [9 *, c2s5]

Para llevar a cabo pruebas o mantenimiento en una organización. De manera inteligente y rentable, los medios utilizados para probar cada parte del software debe reutilizarse sistemáticamente. Un repositorio de materiales de prueba. debe estar bajo el control del software gestión de la figuración para que cambie a soft- los requisitos o el diseño de las mercancías pueden reflejarse en cambios a las pruebas realizadas.

Las soluciones de prueba adoptadas para probar algunos tipos de aplicación en determinadas circunstancias, con las motivaciones detrás de las decisiones tomadas, formar un patrón de prueba que pueda ser documentado para su posterior reutilización en proyectos similares.

5.2.3. Prueba de desarrollo del entorno [1 *, c12s6]

El entorno utilizado para las pruebas debe ser compuesto compatible con el otro software de software adoptado herramientas de neering. Debería facilitar el desarrollo y control de casos de prueba, así como registro y recuperación de resultados esperados, scripts y otros materiales de prueba

5.2.4. Ejecución [1 *, c12s7]

La ejecución de las pruebas debe incorporar un principio básico. principio de experimentación científica: todo realizado durante la prueba debe realizarse y documentado con suficiente claridad que otra persona

Page 96

podría replicar los resultados. Por lo tanto, las pruebas deberían realizarse de acuerdo con lo documentado procedimientos que utilizan una versión claramente definida del software bajo prueba.

5.2.5. Evaluación de resultados de prueba [9 *, c15]

Los resultados de las pruebas deben evaluarse para determinar si la prueba ha sido o no exitoso. En la mayoría de los casos, "exitoso" significa que el software funcionó como se esperaba y lo hizo no tiene ningún resultado inesperado importante. No todos los resultados inesperados son necesariamente fallas pero a veces se determina que son simplemente ruido. Antes de que se pueda eliminar una falla, un análisis y Se necesita un esfuerzo de depuración para aislar, identificar y descríbelo. Cuando los resultados de la prueba son particularmente importante, una junta de revisión formal puede ser considerada vened para evaluarlos.

5.2.6. Informe de problemas / registro de prueba [1 *, c13s9]

Las actividades de prueba se pueden ingresar en una prueba iniciar sesión para identificar cuándo se realizó una prueba, quién realizó la prueba, qué configuración de software se utilizó y otra información de identificación relevante mation. Los resultados de la prueba inesperados o incorrectos en Edgen ser registrado en un sistema de informe de problemas, el datos para los cuales forma la base para la depuración posterior. Ging y solucionar los problemas que se observaron como fallas durante las pruebas. Además, las anomalías no clasificado como fallas podría documentarse en caso luego resultan ser más serios que los primeros pensamiento. Los informes de prueba también son entradas al función proceso de solicitud de gestión (ver Software Con- Control de figuración en la configuración del software Gestión KA).

5.2.7. Seguimiento de defectos [9 *, c9]

Los defectos pueden ser rastreados y analizados para determinar cuando fueron introducidos en el software, por qué fueron creados (por ejemplo, mal El software. Se utiliza la información de seguimiento de defectos para determinar qué aspectos de las pruebas de software y otros procesos necesitan mejoras y cómo enfoques anteriores efectivos han sido.

6. Herramientas de prueba de software

6.1. Soporte de herramientas de prueba [1 *, c12s11] [9 *, c5]

Las pruebas requieren muchas tareas intensivas en mano de obra. Ning numerosas ejecuciones de programas y manejo Una gran cantidad de información. Herramientas apropiadas puede aliviar la carga de la opera administrativa y tediosa y hacerlos menos propensos a errores. Sofista Las herramientas adaptadas pueden soportar el diseño de prueba y el caso de prueba generación, haciéndolo más efectivo.

6.1.1. Seleccionar herramientas [1 *, c12s11]

Orientación a gerentes y evaluadores sobre cómo seleccionar herramientas de prueba que serán más útiles para su organización nización y procesos es un tema muy importante, como la selección de herramientas afecta en gran medida la eficiencia de las pruebas y efectividad. La selección de herramientas depende de evidencia diversa, como opciones de desarrollo, objetivos de evaluación, facilidades de ejecución, etc. en Edgen general, puede que no haya una herramienta única que satisfará necesidades particulares, por lo que un conjunto de herramientas Podría ser una elección adecuada.

6.2. Categorías de herramientas

Clasificamos las herramientas disponibles según su función funcionalidad:

- Arneses de *prueba* (controladores, trozos) [1 *, c3s9] proporcionar un entorno controlado en el que se pueden iniciar pruebas y las salidas de prueba pueden estar registrado Para ejecutar partes de un pro- gram, drivers y stubs se proporcionan para simu- llamadas tardías y módulos llamados, respectivamente.
- Los *generadores de prueba* [1 *, c12s11] proporcionan asistencia tance en los casos de prueba de generación. El gen- la creación puede ser aleatoria, basada en rutas, modelos

requisitos definidos, declaración de variable incorrecta ción, pérdida de memoria, error de sintaxis de programación), y cuando pudieron haber sido observados por primera vez en a base, o una mezcla de los mismos.

• *Herramientas de captura / reproducción* [1 *, c12s11] autovolver a ejecutar o reproducir de forma matemática anteriormente pruebas ejecutadas que tienen entradas grabadas y salidas (p. ej., pantallas).

• *Oracle / comparadores de archivos / comprobación de afirmaciones las herramientas* [1 *, c9s7] ayudan a decidir si un El resultado de la prueba es exitoso o no.

• *Analizadores de cobertura e instructores* [1 *, c4] trabajar juntos. Los analizadores de cobertura evalúan cuál y cuántas entidades del programa gráfico de flujo se han ejercido entre todos los requeridos por la cobertura de prueba seleccionada criterio. El análisis se puede hacer gracias a instructores de programa que insertan la grabación sondas en el código.

• Los *trazadores* [1 *, c1s7] registran el historial de un rutas de ejecución del programa.

• Soporte de *herramientas de prueba de regresión* [1 *, c12s16] la ejecución de un conjunto de pruebas después de una sección del software ha sido modificado. También pueden ayuda para seleccionar un subconjunto de prueba de acuerdo con el cambio realizado.

• Soporte de *herramientas de evaluación de confiabilidad* [9 *, c8] análisis de resultados de prueba y visualización gráfica para evaluar la medición relacionada con la confiabilidad demanda de acuerdo con los modelos seleccionados.

Page 97

98

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	08				
	y 20				
	ath		**ile 2011**	**03**	
	Dep **[1 *]**		**erv** **[2 *]**	**un 20** **[9 *]**	**[10 *]**
	d T		**metro**	**K**	**ielsen 1993**
			gm		**norte**
			S		
	aik an norte				
1. Fundamentos de pruebas de software					
1.1. Terminología relacionada con pruebas					
1.1.1. Definiciones de prueba y Terminología relacionada	c1, c2		c8		
1.1.2. Fallos vs. Fallos	c1s5		c11		
1.2. Cuestiones clave					
1.2.1 Criterios de selección de prueba / Prueba de criterios de adecuación (Reglas de detención)	c1s14, c6s6, c12s7				
1.2.2. Prueba de efectividad / Objetivos para la prueba	c13s11, c11s4				
1.2.3 Prueba de defecto Identificación	c1s14				
1.2.4. El problema de Oracle	c1s9, c9s7				
1.2.5 Teórico y práctico Limitaciones de las pruebas	c2s7				
1.2.6. El problema de lo inviable Caminos	c4s7				
1.2.7. Testabilidad	c17s2				
1.3. Relación de las pruebas con Otras actividades					
1.3.1. Prueba vs. Estática Gestión de calidad de software Tecnicas	c12				
1.3.2. Prueba versus corrección Pruebas y Verificación Formal	c17s2				
1.3.3. Prueba versus depuración	c3s6				
1.3.4. Prueba versus programación	c3s2				
2. Niveles de prueba					
2.1. El objetivo de la prueba	c1s13		c8s1		
2.1.1. Examen de la unidad	c3		c8		
2.1.2. Pruebas de integración	c7		c8		
2.1.3. Prueba de sistema	c8		c8		

Page 99

	08				
	y 20				
	ath		**ile 2011**	**03**	
	Dep **[1 *]**		**erv** **[2 *]**	**un 20** **[9 *]**	**[10 *]**
	d T		**metro**	**K**	**ielsen 1993**
			gm		**norte**
			S		
	aik an norte				
2.2. Objetivos de la prueba	c1s7				
2.2.1. Aceptación / Calificación	c1s7		c8s4		
2.2.2. Prueba de instalación	c12s2				
	c13s7,				

2.2.3. Pruebas alfa y beta	c16s6	c8s4			
2.2.4. Logro de confiabilidad y evaluación	c15	c15s2			
2.2.5. Pruebas de regresión	c8s11, c13s3				
2.2.6. Pruebas de rendimiento	c8s6				
2.2.7. Pruebas de seguridad	c8s3	c11s4			
2.2.8. Pruebas de estrés	c8s8				
2.2.9. Pruebas consecutivas					
2.2.10. Prueba de recuperación	c14s2				
2.2.11. Prueba de interfaz		c8s1.3	c4s4.5		
2.2.12. Prueba de configuración	c8s5				
2.2.13. Usabilidad y humano Prueba de interacción informática				c6	
3. Técnicas de prueba					
3.1. Basado en el software La intuición del ingeniero y Experiencia					
3.1.1. Ad hoc					
3.1.2. Prueba exploratoria					
3.2. Basado en el dominio de entrada Tecnicas					
3.2.1. Partición de equivalencia	c9s4				
3.2.2. Prueba por pares	c9s3				
3.2.3. Análisis de valor límite	c9s5				
3.2.4. Pruebas aleatorias	c9s7				
3.3. Técnicas basadas en códigos					
3.3.1 Control basado en flujo Criterios	c4				

Página 100

	08				
	y 20				
	ath			**03**	
	Dep	**ile 2011**			
	[1 *] **d T**	**erv[2 *]** **metro**	**un 20[9 *]** **K**		**[10 *]** **ielsen 1993 norte**
		om			
	S				
	aik an norte				
3.3.2. Criterios basados en el flujo de datos	c5				
3.3.3 Modelos de referencia para Prueba basada en código	c4				
3.4. Técnicas basadas en fallas	c1s14				
3.4.1. Error Adivinando	c9s8				
3.4.2. Prueba de mutación	c3s5				
3.5. Técnicas basadas en el uso					
3.5.1. Perfil operacional	c15s5				
3.5.2. Observacion del usuario Heurística					c5, c7
3.6. Pruebas basadas en modelos Tecnicas					
3.6.1. Tabla de decisiones	c9s6				
3.6.2. Máquinas de estado finito	c10				
3.6.3. Prueba de formal Presupuesto	c10s11	c15			

3.7. Técnicas basadas en el Naturaleza de la aplicación					
3.8. Seleccionar y combinar Tecnicas					
3.8.1. Funcional y estructural	c9				
3.8.2. Determinista vs.Aleatorio	c9s6				
4. Medidas relacionadas con la prueba					
4.1. Evaluación del programa Bajo prueba					
4.1.1 Programa de mediciones Esa ayuda en la planificación y Pruebas de diseño				c11	
4.1.2. Tipos de fallas, clasificación, y estadísticas				c4	
4.1.3. Densidad de falla	c13s4			c4	
4.1.4. Prueba de vida, fiabilidad Evaluación	c15			c3	
4.1.5. Modelos de crecimiento de confiabilidad	c15			c8	

Page 101

	08				
	y 20				
	ath			**03**	
	Dep	**ile 2011**			
	[1 *] **d T**	**erv** **[2 *]** **metro**		**un 20** **[9 *]** **K**	**[10 *]** **ielsen 1993 norte**
		gm S			
	aik an norte				
4.2. Evaluación de las pruebas Realizado					
4.2.1 Cobertura / minuciosidad Medidas				c11	
4.2.2 Siembra de fallas	c2s5			c6	
4.2.3 Puntuación de mutación	c3s5				
4.2.4 Comparación y relativo Efectividad de diferentes Tecnicas					
5. Proceso de prueba					
5.1. Consideraciones prácticas					
5.1.1 Actitudes / Egoless Programación	c16			c15	
5.1.2. Guías de prueba	c12s1			c15s1	
5.1.3. Prueba de gestión de procesos	c12			c15	
5.1.4. Documentación de prueba y Productos del trabajo	c8s12			c4s5	
5.1.5. Desarrollo guiado por pruebas	c1s16				
5.1.6. Interna versus independiente Equipo de prueba	c16				
5.1.7. Estimación de costo / esfuerzo y Otras medidas de proceso	c18s3			c5s7	
5.1.8. Terminación				c10s4	
5.1.9. Prueba de reutilización y patrones				c2s5	
5.2. Actividades de prueba					
5.2.1. Planificación	c12s1 c12s8				
5.2.2. Generación de casos de prueba	c12s1				

5.2.3. Entorno de prueba Desarrollo	c12s3 c12s6				
5.2.4. Ejecución	c12s7				
5.2.5. Evaluación de resultados de prueba			c15		

Page 102

	08 **y 20** **ath** **Dep** **d T** **aik an norte** [1 *]	**ile 2011** **erv**[2 *] **metro** **om** **S**	**03** **un 20**[9 *] **K**	**[10 *]** **ielsen 1993** **norte**
5.2.6. Informe de problemas / prueba Iniciar sesión	c13s9			
5.2.7. Seguimiento de defectos			c9	
6. Herramientas de prueba de software				
6.1. Soporte de herramientas de prueba	c12s11		c5	
6.1.1. Seleccionar herramientas	c12s11			
6.2. Categorías de herramientas	c1s7, c3s9, c4, c9s7, c12s11, c12s16		c8	

Page 103

Referencias

[1 *] S. Naik y P. Tripathy, *Pruebas de software y garantía de calidad: teoría y Práctica* , Wiley-Spektrum, 2008.

[2 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[3] MR Lyu, ed., *Manual de software Ingeniería de Confiabilidad* , McGraw-Hill y IEEE Computer Society Press, 1996.

[4] H. Zhu, PAV Hall, y JHR May, "Cobertura de prueba de unidad de software y Adecuación ", *ACM Computing Surveys,* vol. 29, no. 4, diciembre de 1997, págs. 366–427.

[5] EW Dijkstra, “Notas sobre estructurado Programación ", Informe TH 70-WSE-03, Universidad Tecnológica, Eindhoven, 1970; http://www.cs.utexas.edu/users/EWD/ewd02xx / EWD249.PDF.

[6] *ISO / IEC / IEEE P29119-1 / DIS Proyecto de norma para Ingeniería de Software y Sistemas— Pruebas de software — Parte 1: Conceptos y Definiciones* , ISO / IEC / IEEE, 2012.

[7] *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010.

[8] S. Yoo y M. Harman, "Pruebas de regresión Minimización, selección y priorización: Una encuesta ", *Verificación de pruebas de software y Fiabilidad,* vol. 22, no. 2, marzo de 2012, pp. 67-120.

[9 *] SH Kan, *métricas y modelos en software Ingeniería de calidad* , 2ª ed., Addison-Wesley, 2002.

[10 *] J. Nielsen, *Ingeniería de usabilidad* , Morgan Kaufmann, 1993.

[11] TY Chen et al., “Pruebas aleatorias adaptativas: El ARTE de la diversidad de casos de prueba ", *Journal de Sistemas y Software,* vol. 83, no. 1 de enero 2010, págs. 60–66.

[12] Y. Jia y M. Harman, "Un análisis y Encuesta sobre el desarrollo de Prueba de mutación ”, *IEEE Trans. Software Ingeniería,* vol. 37, no. 5 de septiembre a octubre 2011, pp. 649-678.

[13] M. Utting y B. Legeard, *práctico. Pruebas basadas en modelos: un enfoque de herramientas* , Morgan Kaufmann, 2007.

CAPÍTULO 5

MANTENIMIENTO DEL SOFTWARE

SIGLAS

durante la etapa posterior a la entrega. Actividad previa a la entrega

SEÑOR	Solicitud de Modificación
PR	Informe de problema
SCM	Configuración de software administración
SLA	Acuerdo de nivel de servicio
SQA	Aseguramiento de la calidad del software
V&V	Verificación y validación

INTRODUCCIÓN

Los esfuerzos de desarrollo de software dan como resultado la entrega ery de un producto de software que satisface al usuario requisitos En consecuencia, el producto de software debe cambiar o evolucionar. Una vez en funcionamiento, defectos están descubiertos, los entornos operativos cambian, y superficie de requisitos de nuevos usuarios. El mainte- La fase financiera del ciclo de vida comienza después de una período de garantía o soporte posterior a la implementación entrega, pero las actividades de mantenimiento ocurren mucho más temprano.

El mantenimiento del software es una parte integral de un enfatice por qué hay necesidad de mantenimiento. ciclo de vida del software. Sin embargo, no ha recibido el mismo grado de atención que las otras fases tener. Históricamente, el desarrollo de software ha tenido un perfil mucho más alto que el mantenimiento de software en la mayoría de las organizaciones. Esto ahora está cambiando, ya que las organizaciones se esfuerzan por exprimir al máximo su inversión en desarrollo de software por keep- ing software que funcione el mayor tiempo posible. los El paradigma de código abierto ha traído más atención ión a la cuestión de mantener artefactos de software desarrollado por otros.

En esta *guía* , el mantenimiento del software está definido como la totalidad de las actividades requeridas para proporcionar soporte rentable para el software. Las actividades son realizado durante la etapa previa a la entrega, así como los lazos incluyen la planificación de operaciones posteriores a la entrega, mantenibilidad y determinación logística para actividades de transición [1 *, c6s9]. Fijar una entrega Las actividades incluyen modificación de software, capacitación, y operar o interactuar con una mesa de ayuda.

El área de conocimiento de mantenimiento de software (KA) está relacionado con todos los demás aspectos del software Ingenieria. Por lo tanto, esta descripción de KA es vinculado a todos los demás KA de ingeniería de software de La *guía* .

DESGLOSE DE TEMAS PARA MANTENIMIENTO DEL SOFTWARE

El desglose de temas para el Software Main- Tenance KA se muestra en la Figura 5.1.

1. Fundamentos de mantenimiento de software

Esta primera sección presenta los conceptos y terminología que forma una base subyacente para Comprender la función y el alcance del software mantenimiento. Los temas proporcionan definiciones y Las categorías de mantenimiento de software son críticas para entendiendo su significado subyacente.

1.1. Definiciones y terminología

[1 *, c3] [2 *, c1s2, c2s2]

Se define el propósito del mantenimiento del software. en el estándar internacional para mantenimiento de software nance: ISO / IEC / IEEE 14764 [1 *]. [1] En el contexto de ingeniería de software, el mantenimiento de software es esencialmente uno de los muchos procesos técnicos.

[1] En el propósito de concisión y facilidad de lectura ing, este estándar se conoce simplemente como IEEE 14764 en el texto posterior de este KA.

Figura 5.1 . Desglose de temas para el mantenimiento de software KA

El objetivo del mantenimiento del software es modificar el software existente mientras preserva su integridad. La norma internacional también establece la importancia de tener algo de mantenimiento actividades previas a la entrega final de software (actividades previas a la entrega). En particular, IEEE 14764 enfatiza la importancia de la entrega previa aspectos de mantenimiento: planificación, por ejemplo.

1.2. Naturaleza del mantenimiento

[2 *, c1s3]

El mantenimiento del software sostiene el producto de software uct a lo largo de su ciclo de vida (desde el desarrollo a las operaciones). Las solicitudes de modificación se registran y rastreado, el impacto de los cambios propuestos es determinado, el código y otros artefactos de software son modificado, se realizan pruebas y una nueva versión del producto de software es lanzado. Además, entrenar ing y soporte diario se proporcionan a los usuarios. los El responsable de *mantenimiento* se define como una organización que realiza actividades de mantenimiento. En este KA, el término a veces se referirá a individuos que formar esas actividades, contrastando con el desarrolladores

IEEE 14764 identifica las actividades principales de mantenimiento de software como implementación de procesos, análisis de problemas y modificaciones, modificaciones implementación, revisión / aceptación de mantenimiento, migración y jubilación. Estas actividades son discutido en la sección 3.2, Actividades de mantenimiento. Los mantenedores pueden aprender del desarrollo conocimiento de los usuarios del software. Contactar con los desarrolladores y la participación temprana de la

Page 106

mantenedor ayuda a reducir el mantenimiento general esfuerzo. En algunos casos, el desarrollador inicial no puede ser alcanzado o ha pasado a otras tareas, lo que crea un desafío adicional para main- tainers El mantenimiento debe tomar artefactos de software desde el desarrollo (por ejemplo, código o documentación) mentación) y apoyarlos de inmediato, luego evolucionar progresivamente / mantenerlos sobre un suave ciclo de vida de las mercancías.

1.3. Necesidad de mantenimiento

[2 *, c1s5]

Se necesita mantenimiento para garantizar que el software continúa satisfaciendo los requisitos del usuario. Mainte- nance es aplicable al software que se desarrolla utilizando cualquier modelo de ciclo de vida del software (por ejemplo, espiral o lineal). Los productos de software cambian debido a acciones de software correctivas y no correctivas. El mantenimiento debe realizarse para

- fallas correctas;
- mejorar el diseño;
- implementar mejoras;
- interfaz con otro software;
- adaptar programas para que diferentes hardware, software, características del sistema y telecomunicaciones se pueden usar instalaciones de cationes;
- migrar software heredado; y
- retirar el software.

Cinco características clave comprenden el mantenimiento actividades de er:

- mantener el control sobre el día del software funciones de hoy;
- mantener el control sobre el software modificación;

percepción del mantenimiento del software es que simplemente corrige fallas. Sin embargo, estudios y estudios Veys a lo largo de los años han indicado que las principales Más del 80 por ciento del mantenimiento del software es utilizado para acciones no correctivas [2 *, figura 4.1]. Agrupando mejoras y correcciones juntas en los informes de gestión contribuye a algunos errores concepciones sobre el alto costo de la corrección iones Comprender las categorías de software el mantenimiento ayuda a comprender la estructura de costos de mantenimiento de software. Además, entendiendo los factores que influyen en la mantenibilidad de El software puede ayudar a contener los costos. Algún ambiente factores mentales y su relación con el software los costos de mantenimiento incluyen lo siguiente:

- El entorno operativo se refiere al hardware. y software.
- El ambiente organizacional se refiere a la política. cies, competencia, proceso, producto y personal.

1.5. Evolución del software

[2 *, c3s5]

El mantenimiento del software en términos de evolución fue abordado por primera vez a finales de la década de 1960. Durante un período de veinte años, la investigación condujo a la formulación de ocho "Leyes de la evolución". Los hallazgos clave incluyen un propuesta de que el mantenimiento es un desarrollo evolutivo opción y que las decisiones de mantenimiento son ayudadas entendiendo lo que le sucede al software hora. Algunos afirman que el mantenimiento continúa desarrollo, excepto que hay una entrada adicional (o restricción), en otras palabras, soft grande existente el material nunca está completo y continúa evolucionando; a medida que evoluciona, se vuelve más complejo a menos que algunos Se toman medidas para reducir esta complejidad.

• perfeccionar las funciones existentes;
• identificación de amenazas de seguridad y reparación de vulnerabilidades de la ciudad; y
• prevenir el rendimiento del software de degradante a niveles inaceptables.

1.4. Mayoría de costos de mantenimiento
[2 *, c4s3, c5s5.2]

El mantenimiento consume una parte importante de las finanzas recursos sociales en un ciclo de vida del software. Una común

1.5. Categorías de mantenimiento
[1 *, c3, c6s2] [2 *, c3s3.1]

Tres categorías (tipos) de mantenimiento tienen definido: correctivo, adaptativo y perfecto tive [2 *, c4s3]. IEEE 14764 incluye un cuarto categoría preventiva.

• Mantenimiento correctivo: modificación reactiva cación (o reparaciones) de un producto de software

realizado después del parto para corregir el descubrimiento Ered problemas. Incluido en esta categoría es mantenimiento de emergencia, que es un modificación no programada realizada a tem- mantener en funcionamiento un producto de software de forma temporal pendiente de mantenimiento correctivo.
• Mantenimiento adaptativo: modificación de un producto de software realizado después de la entrega a mantener un producto de software utilizable en un cambio o entorno cambiante. Por ejemplo, el sistema operativo podría actualizarse y algunos cambios en el software pueden ser necesario.
• Mantenimiento perfecto: modificación de un producto de software después de la entrega para proporcionar mejoras para los usuarios, mejora de documentación del programa y recodificación a mejorar el rendimiento del software, mantener capacidad u otros atributos de software.
• Mantenimiento preventivo: modificación de un producto de software después de la entrega para detectar y corregir fallas latentes en el producto de software antes de que se conviertan en fallas operacionales.

IEEE 14764 clasifica adaptativo y perfecto mantenimiento como mejoras de mantenimiento. Eso También agrupa el correctivo y el preventivo Tive categorías de mantenimiento en una categoría de corrección egory, como se muestra en la Tabla 5.1.

Tabla 5.1. Categorías de mantenimiento de software

	Corrección	**Mejora**
Proactivo	Preventivo	Perfecto
Reactivo	Correctivo	Adaptado

2. Cuestiones clave en el mantenimiento del software

Se deben abordar una serie de cuestiones clave para Garantizar el mantenimiento efectivo del software. El mantenimiento del software proporciona técnicas únicas. desafíos de cal y de gestión para el software ingenieros, por ejemplo, tratando de encontrar una falla en software que contiene una gran cantidad de líneas de código que desarrolló otro ingeniero de software. Del mismo modo, competir con los desarrolladores de software por los recursos es una batalla constante. Planeando un lanzamiento futuro, que a menudo incluye la codificación del próximo lanzamiento al enviar parches de emergencia para la versión actual, también crea un desafío.

La siguiente sección presenta algunos de los Problemas de administración y de gestión relacionados con el software. mantenimiento. Se han agrupado bajo el siguientes encabezados de tema:

• problemas técnicos,
• asuntos Gerenciales,
• estimación de costos, y
• medición.

2.1. Problemas técnicos

2.1.1. Comprensión limitada
[2 *, c6]

La comprensión limitada se refiere a qué tan rápido un ingeniero de software puede entender dónde hacer un cambio o corrección en el software que él o ella No se desarrolló. La investigación indica que aproximadamente la mitad del esfuerzo total de mantenimiento se dedica a de pie el software a modificar. Por lo tanto, la El tema de la comprensión del software es de gran interés. est para los ingenieros de software. La comprensión es más difícil en la representación orientada al texto, en la fuente código, por ejemplo, donde a menudo es difícil crear la evolución del software a través de sus lanzamientos / versiones si los cambios no están documentados y si el los desarrolladores no están disponibles para explicarlo, que es A menudo el caso. Por lo tanto, los ingenieros de software pueden Tener una comprensión limitada del software; Hay mucho por hacer para remediar esto.

2.1.2. Pruebas
[1 *, c6s2.2.2] [2 *, c9]

El costo de repetir la prueba completa en un importante pieza de software es importante en términos de tiempo y dinero. Para asegurar que el pedido los informes de problemas son válidos, el responsable debe replicar o verificar problemas ejecutando el pruebas apropiadas Prueba de regresión (la selección nueva prueba de software o un componente para verificar si las modificaciones no han provocado efectos tendidos) es un concepto de prueba importante en mantenimiento. Además, encontrar tiempo para probar es a menudo difícil Coordinación de pruebas cuando es diferente miembros del equipo de mantenimiento están trabajando

en diferentes problemas al mismo tiempo sigue siendo un reto. Cuando el software realiza funciones críticas es posible que sea difícil ponerlo fuera de línea para probarlo. Las pruebas no se pueden ejecutar de la manera más significativa en el lugar completo: el sistema de producción. El software Testing KA proporciona información adicional y referencias sobre este asunto en su subtema sobre regresión prueba de sion.

2.1.3. Análisis de impacto

[1 *, c5s2.5] [2 *, c13s3]

El análisis de impacto describe cómo conducir, costo-efectivamente, un análisis completo del impacto de Un cambio en el software existente. Los mantenedores deben Poseer un conocimiento íntimo de los programas estructura y contenido. Ellos usan ese conocimiento para realizar un análisis de impacto, que identifica todos sistemas y productos de software afectados por un software solicitud de cambio de mercancías y desarrolla una estimación de Los recursos necesarios para lograr el cambio. Además, el riesgo de realizar el cambio es determinado. La solicitud de cambio, a veces llamada una solicitud de modificación (MR) y a menudo llamada informe de problema (PR), primero debe analizarse y traducido a términos de software. El análisis de impacto es realizado después de que una solicitud de cambio ingrese al Mejorar proceso de gestión de la configuración del hardware IEEE 14764 establece las tareas de análisis de impacto:

- analizar MR / PR;
- replicar o verificar el problema;
- desarrollar opciones para implementar el modificación;
- documentar el MR / PR, los resultados y el opciones de ejecución;
- obtener aprobación para la modificación seleccionada opción.

La gravedad de un problema a menudo se usa para decida cómo y cuándo se solucionará. Lo suave- El ingeniero de hardware identifica el equipo afectado ponentes Se proporcionan varias soluciones potenciales, seguido de una recomendación sobre el mejor Curso de acción.

Software diseñado teniendo en cuenta la mantenibilidad facilita enormemente el análisis de impacto. Más información se puede encontrar en la Configuración de software Gestión KA.

2.1.4. Mantenibilidad

[1 *, c6s8] [2 *, c12s5.5]

IEEE 14764 [1 *, c3s4] define mantenibilidad como la capacidad del producto de software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el medio ambiente, así como cambios en requisitos y especificaciones funcionales.

Como característica principal de la calidad del software, la mantenibilidad debe especificarse, revisarse y controlado durante el desarrollo de software activi- ataduras para reducir los costos de mantenimiento. Cuando hecho con éxito, la mantenibilidad del software mejorará. La mantenibilidad es a menudo difícil de lograr porque las subcaracterísticas son a menudo no es un enfoque importante durante el proceso de soft- desarrollo de artículos. Los desarrolladores son, típicamente, más preocupado con muchas otras actividades y con frecuencia propenso a ignorar al mantenedor requisitos Esto a su vez puede, y a menudo lo hace, resultar en una falta de documentación y prueba de software entornos, que es una causa principal de dificultades vínculos en la comprensión del programa y posteriores análisis de impacto. La presencia de sistemática y procesos maduros, técnicas y herramientas ayudan a Mejorar la capacidad de mantenimiento del software.

2.2. Asuntos Gerenciales

2.2.1. Alineamiento con Organizacional Objetivos

[2 *, c4]

Los objetivos organizacionales describen cómo demostrar estratifique el retorno de la inversión del software principal actividades de tenencia. El desarrollo inicial del software es generalmente basado en proyectos, con una escala de tiempo definida y presupuesto. El énfasis principal es entregar un producto que satisface las necesidades del usuario a tiempo y dentro del presupuesto. En contraste, el mantenimiento del software a menudo tiene la objetivo de extender la vida útil del software para el mayor tiempo posible Además, puede ser impulsado por la necesidad de satisfacer la demanda del usuario de actualizaciones de software y mejoras. En ambos casos, el retorno de la inversión es mucho menos clara, por lo que la vista en el nivel de alta gerencia es a menudo el de un importante actividad que consume recursos significativos sin claro beneficio cuantificable para la organización.

2.2.2. Dotación de personal

[2 *, c4s5, c10s4]

La dotación de personal se refiere a cómo atraer y mantenerse suave. personal de mantenimiento de vajilla. El mantenimiento no es [2 *, c9] visto como trabajo glamoroso. Como resultado, el software el personal de mantenimiento es visto frecuentemente

asignación de la responsabilidad de mantenimiento a un grupo individual o persona, independientemente de la organización estructura de la nación.

2.2.3. Outsourcing

[3 *]

como "ciudadanos de segunda clase" y, por lo tanto, moral sufre

2.2.3. Proceso

[1 *, c5] [2 *, c5]

El proceso del ciclo de vida del software es un conjunto de actividades, Métodos, prácticas y transformaciones que utilice para desarrollar y mantener software y su Productos Asociados. A nivel de proceso, software las actividades de mantenimiento tienen mucho en común con desarrollo de software (por ejemplo, software La gestión de la configuración es una actividad crucial en ambos). El mantenimiento también requiere varias actividades que no se encuentran en el desarrollo de software (ver sección 3.2 sobre actividades únicas para más detalles). Estas Las actividades presentan desafíos para la gerencia.

2.2.4. Aspectos organizacionales del mantenimiento

[1 *, c7s2.3] [2 *, c10]

Los aspectos organizacionales describen cómo iden- determinar qué organización y / o función será responsable del mantenimiento del software. los el equipo que desarrolla el software no es necesario asignado para mantener el software una vez que está Operacional.

Al decidir dónde se realiza el mantenimiento del software se ubicará la función, ingeniería de software las organizaciones pueden, por ejemplo, quedarse con el desarrollador original o ir a un main- permanente equipo específico de mantenimiento (o mantenedor). Teniendo en El equipo de mantenimiento permanente tiene muchos beneficios:

- permite la especialización;
- crea canales de comunicación;
- promueve una atmósfera colegial sin ego;
- reduce la dependencia de los individuos;
- permite verificaciones periódicas de auditoría.

Dado que hay muchos pros y contras para cada uno opción, la decisión debe tomarse caso por caso base de caso. Lo importante es la delegación o Software de tercerización y deslocalización mainte- Las finanzas se han convertido en una industria importante. Organiza- las operaciones subcontratan carteras enteras de software Ware, incluido el mantenimiento de software. Más a menudo, la opción de outsourcing se selecciona por menos software de misión crítica, como lo son las organizaciones dispuestos a perder el control del software utilizado en su negocio principal Uno de los mayores desafíos. para los subcontratistas es determinar el alcance de la servicios de mantenimiento requeridos, los términos de un servicio acuerdo de nivel de vicio, y los detalles contractuales. Los subcontratistas deberán invertir en un mantenimiento infraestructura y la mesa de ayuda en el sitio remoto debe tener personal con hablantes nativos. La subcontratación requiere una inversión inicial significativa ment y la configuración de un proceso de mantenimiento que requerirá automatización.

2.3. Estimación de costos de mantenimiento

Los ingenieros de software deben entender los diferentes categorías de mantenimiento de software, discutidas arriba, para abordar la cuestión de la estimación ing el costo del mantenimiento del software. Para el plan Para propósitos, la estimación de costos es un importante aspecto de la planificación para el mantenimiento del software.

2.3.1. Estimación de costos

[2 *, c7s2.4]

La sección 2.1.3 describe cómo se iden los análisis de impacto tifica todos los sistemas y productos de software afectados por una solicitud de cambio de software y desarrolla un Estimación de los recursos necesarios para lograr ese cambio

Las estimaciones de costos de mantenimiento se ven afectadas por muchos factores técnicos y no técnicos. IEEE 14764 afirma que "los dos más populares enfoques para estimar recursos para software mantenimiento son el uso de modelos paramétricos y el uso de la experiencia "[1 *, c7s4.1]. Una combinación nación de estos dos también se puede utilizar.

Page 110

2.3.2. Modelos paramétricos

[2 *, c12s5.6]

Modelado de costos paramétricos (modelos matemáticos) se ha aplicado al mantenimiento de software. De sig- importancia es que los datos históricos del pasado son necesarios para utilizar y calibrar Los modelos matemáticos. Atributos del controlador de costos afectar las estimaciones.

2.3.3. Experiencia

[2 *, c12s5.5]

Experiencia, en forma de juicio experto, a menudo se usa para estimar el esfuerzo de mantenimiento. Claramente, el mejor enfoque para el mantenimiento es mación es combinar datos históricos y experimentación ence. El costo para realizar una modificación (en términos de número de personas y cantidad de tiempo) es entonces derivado. Datos históricos de estimación de mantenimiento debe proporcionarse como resultado de una medición modelo de calidad sugiere medidas que son específicas para mantenimiento de software. Medidas para subchar- Las características de mantenimiento incluyen lo siguiente: ing [4 *, pág. 60]:

- Analizabilidad: medidas del mantenedor esfuerzo o recursos gastados en intentar cualquiera para diagnosticar deficiencias o causas de falla o para identificar partes a modificar.
- Capacidad de cambio: medidas del mantenedor esfuerzo asociado con la implementación de una especi modificación modificada
- Estabilidad: medidas del comportamiento inesperado. ior del software, incluido el encontrado durante las pruebas
- Testabilidad: medidas del mantenedor y esfuerzo de los usuarios al intentar probar el modificado software.
- Otras medidas que usan los mantenedores incluyen
- tamaño del software,
- complejidad del software,

programa.

2.4. Medición de mantenimiento de software

[1 *, c6s5] [2 *, c12]

Entidades relacionadas con el mantenimiento de software, cuyos atributos pueden estar sujetos a medición, incluye proceso, recurso y producto [2 *, c12s3.1].

Existen varias medidas de software que pueden derivarse de los atributos del software, El proceso de mantenimiento, y el personal, incluyendo- tamaño, complejidad, calidad, comprensibilidad, mantenibilidad y esfuerzo. Medidas de complejidad de software también se puede obtener usando disponible herramientas comerciales Estas medidas constituyen un buen punto de partida para la medida del mantenedor programa de ment. Discusión del proceso de software. y la medición del producto también se presenta en el Proceso de Ingeniería de Software KA. El tema de un programa de medición de software se describe en la Gerencia de Ingeniería de Software KA.

2.4.1 Medidas específicas

[2 *, c12]

El mantenedor debe determinar qué medidas son apropiados para una organización específica basada en el propio contexto de esa organización. El software

- comprensibilidad, y
- mantenibilidad.

Proporcionando esfuerzo de mantenimiento de software, por categorías, para diferentes aplicaciones proporciona información comercial a los usuarios y sus organizaciones iones También puede permitir la comparación de soft- mantenimiento de los perfiles internos dentro de un organización.

3. Proceso de mantenimiento

Además del programa estándar de ingeniería de software ceses y actividades descritas en IEEE 14764, hay una serie de actividades que son exclusivas de mantenedores.

3.1. Procesos de mantenimiento

[1 *, c5] [2 *, c5] [5, s5.5]

Los procesos de mantenimiento proporcionan las actividades necesarias. y entradas / salidas detalladas para esas actividades como descrito en IEEE 14764. El programa de mantenimiento Las actividades de cess de IEEE 14764 se muestran en la Figura 5.2. Las actividades de mantenimiento de software incluyen

- implementación del proceso,
- análisis de problemas y modificaciones,
- implementación de modificación,

Página 111

- revisión / aceptación de mantenimiento,
- migración, y
- retiro de software.

Figura 5.2. Proceso de mantenimiento de software

Otros modelos de proceso de mantenimiento incluyen:

- arreglo rapido,
- espiral,
- Osborne's,
- mejora iterativa, y
- Orientado a la reutilización.

Recientemente, metodologías ágiles, que promueven procesos ligeros, también se han adaptado a

las actividades y tareas son responsabilidad de mantenedor Como ya se señaló, muchos mantenimientos las actividades son similares a las del desarrollo de software ment. Los mantenedores realizan análisis, diseño, codificación ing, pruebas y documentación. Deben rastrear requisitos en sus actividades, tal como se hace en desarrollo, y actualice la documentación como las líneas de base cambian. IEEE 14764 recomienda que cuando un mantenedor usa un proceso de desarrollo, debe adaptarse para satisfacer necesidades específicas [1 *, c5s3.2.2]. Sin embargo, para el mantenimiento del software, Algunas actividades implican procesos exclusivos de soft- mantenimiento de vajilla.

3.2.1. Actividades unicas

[1 *, c3s10, c6s9, c7s2, c7s3] [2 *, c6, c7]

Hay una serie de procesos, actividades y prácticas exclusivas del mantenimiento de software:

- Comprensión del programa: actividades necesarias para obtener un conocimiento general de lo que es un software producto y cómo funcionan las piezas juntas.
- Transición: un control y coordinación secuencia de actividades durante las cuales el software se transfiere progresivamente desde el desarrollo Oper para el mantenedor.
- Solicitud de modificación aceptación / rechazo: modificaciones solicitando trabajo más allá de un cer- El tamaño / esfuerzo / complejidad del tain puede ser rechazado por mantenedores y redirigido a un desarrollador.
- Mesa de ayuda de mantenimiento: un usuario final y mantenimiento coordinado función de apoyo que desencadena la evaluación, priorización, y el costo de las solicitudes de modificación.

tenencia Este requisito surge de la siempre-
demanda creciente para un cambio rápido de
servicios de arrendamiento. Mejora del software.
proceso de mantenimiento es apoyado por especialistas
capacidad de mantenimiento de software modelos de madurez
(ver [6] y [7], que se anotan brevemente en el
Lecturas adicionales sección).

3.2. Actividades de mantenimiento
[1 *, c5, c6s8.2, c7s3.3]

El proceso de mantenimiento contiene las actividades.
y tareas necesarias para modificar un software existente
producto de productos mientras se preserva su integridad. Estas

- Análisis de impacto: una técnica para identificar áreas.
  impactado por un cambio potencial;
- Acuerdos de nivel de servicio de mantenimiento
  (SLAs) y licencias y contratos de mantenimiento
  tratados: acuerdos contractuales que describen
  los servicios y objetivos de calidad.

3.2.2. Actividades de apoyo
[1 *, c4s1, c5, c6s7] [2 *, c9]

Los mantenedores también pueden realizar actividades de apoyo,
como documentación, configuración de software
gestión, verificación y validación, problema
resolución, garantía de calidad del software, revisiones,

y auditorias. Otra importante actividad de apoyo.
consiste en capacitar a los mantenedores y usuarios.

3.2.3. Actividades de planificación de mantenimiento
[1 *, c7s3]

Una actividad importante para el mantenimiento del software es
planificación, y los mantenedores deben abordar los problemas
asociado con una serie de perspectivas de planificación
tives, incluyendo

- planificación empresarial (nivel organizacional),
- planificación de mantenimiento (nivel de transición),
- planificación de lanzamiento / versión (nivel de software), y
- planificación de solicitud de cambio de software individual
  (nivel de solicitud).

En el nivel de solicitud individual, la planificación es
llevado a cabo durante el análisis de impacto (ver sección
2.1.3, Análisis de impacto). El lanzamiento / versión
La actividad de planificación requiere que el mantenedor:

- recopilar las fechas de disponibilidad del individuo
  peticiones,
- acordar con los usuarios sobre el contenido de posteriores
  lanzamientos / versiones,
- identificar conflictos potenciales y desarrollar
  alternativas,
- evaluar el riesgo de una liberación dada y desarrollar
  un plan de retroceso en caso de problemas
  levantarse y
- informar a todos los interesados.

Mientras que los proyectos de desarrollo de software pueden
normalmente duran de unos meses a unos pocos años,
la fase de mantenimiento suele durar muchos
años. Hacer estimaciones de recursos es un elemento clave
ment de planificación de mantenimiento. Software principal
la planificación de la tenencia debe comenzar con la decisión
para desarrollar un nuevo producto de software y debería
considerar objetivos de calidad. Un documento conceptual
debe desarrollarse, seguido de un mantenimiento
plan. El concepto de mantenimiento para cada software.
El producto debe documentarse en el plan [1 *,
c7s2] y debe abordar el

- alcance del mantenimiento del software,
- adaptación del mantenimiento del software
  proceso,
- identificación del mantenimiento del software
  organización, y
- estimación de costos de mantenimiento de software.

El siguiente paso es desarrollar un correspondiente
plan de mantenimiento de software. Este plan debería ser
preparado durante el desarrollo de software y debe
especificar cómo los usuarios solicitarán modificaciones de software
o informar problemas. Mantenimiento del software
la planificación se aborda en IEEE 14764. Proporciona
pautas para un plan de mantenimiento. Finalmente a las
el nivel más alto, la organización de mantenimiento
tendrá que realizar actividades de planificación empresarial
(cursos presupuestarios, financieros y humanos) solo
como todas las otras divisiones de la organización.
La gestión se discute en el capítulo Relacionado
Disciplinas de la Ingeniería del Software.

3.2.4. Gestión de configuración de software
[1 *, c5s1.2.3] [2 *, c11]

IEEE 14764 describe la configuración del software
La gestión como elemento crítico del mantenimiento
proceso financiero Gestión de configuración de software
los procedimientos de mentores deberían prever la verificación
ción, validación y auditoría de cada paso requerido
para identificar, autorizar, implementar y liberar el
producto de software

No es suficiente simplemente rastrear modificaciones
Peticiones o informes de problemas. El software
El producto y cualquier cambio realizado en él debe ser
trolleado Este control se establece mediante la implementación
Inforzar y hacer cumplir una configuración de software aprobada
proceso de gestión de raciones (SCM). El software
Configuration Management KA proporciona detalles
de SCM y analiza el proceso por el cual soft-
Se envían, evalúan, solicitan cambios de mercancías
y aprobado SCM para mantenimiento de software es
diferente de SCM para el desarrollo de software en
el número de pequeños cambios que deben ser con-
controlado por software operativo. El SCM pro-
cess se implementa desarrollando y siguiendo
un plan de gestión de configuración de software y
procedimientos de operación. Los mantenedores participan en
Tableros de control de configuración para determinar el
contenido del próximo lanzamiento / versión.

3.2.5. Calidad de software
[1 *, c6s5, c6s7, c6s8] [2 *, c12s5.3]

No es suficiente simplemente esperar que haya aumentado la calidad resultará del mantenimiento de soft-
mercancía. Los mantenedores deben tener un software calificado
programa de ity. Debe ser planificado y procesado
debe implementarse para apoyar el mantenimiento
proceso. Las actividades y técnicas para Soft-
Garantía de calidad de las mercancías (SQA), V&V, revisiones,
y las auditorías deben seleccionarse en concierto con todos
los otros procesos para alcanzar el nivel deseado
de calidad. También se recomienda que el
tainer adapta los procesos de desarrollo de software,
técnicas y entregas (por ejemplo, pruebas
documentación) y resultados de la prueba. Más detalles pueden
se encuentra en el Software Quality KA.

4. Técnicas de mantenimiento.

Este tema presenta algunos de los
Técnicas aceptadas utilizadas en el mantenimiento de software.

4.1. Comprensión del programa
[2 *, c6, c14s5]

Los programadores pasan mucho tiempo leyendo y
comprender los programas para implementar
cambios Los navegadores de código son herramientas clave para programas
comprensión y se utilizan para organizar y pres
código fuente ent. Documentación clara y concisa
También puede ayudar en la comprensión del programa.

4.2. Reingeniería
[2 *, c7]

La reingeniería se define como el examen y
alteración del software para reconstituirlo en un nuevo
formulario e incluye la implementación posterior
ción de la nueva forma. A menudo no se lleva a cabo para
mejorar la mantenibilidad pero para reemplazar el envejecimiento de los sistemas
software acy. La refactorización es una tecnología de reingeniería
nique que apunta a reorganizar un programa sin
cambiando su comportamiento. Busca mejorar un pro-
estructura de gramo y su mantenibilidad. Refactorizador
Las técnicas de ing pueden usarse durante cambios menores.

4.3. Ingeniería inversa
[1 *, c6s2] [2 *, c7, c14s5]

La ingeniería inversa es el proceso de análisis.
software para identificar los componentes del software
y sus interrelaciones y para crear representaciones
presentaciones del software en otra forma o en
mayores niveles de abstracción. Ingeniería inversa-
ing es pasivo; no cambia el software
resultar en un nuevo software. Ingeniería inversa-
Los esfuerzos producen gráficos de llamadas y control de flujo
gráficos del código fuente. Un tipo de reversa
La ingeniería es la redocumentación. Otro tipo es
recuperación de diseño Finalmente, ingeniería inversa de datos.
ing, donde se recuperan esquemas lógicos de
bases de datos físicas, ha crecido en importancia sobre
Los últimos años. Las herramientas son clave para la ingeniería inversa
tareas relacionadas y relacionadas, como la redocumentación
ción y recuperación del diseño.

4.4. Migración
[1 *, c5s5]

Durante la vida del software, es posible que deba modificarse
Fied para correr en diferentes entornos. A fin de que
migrarlo a un nuevo entorno, el mantenedor
necesita determinar las acciones necesarias para acomodar
terminar la migración, y luego desarrollar y documentar
Marco de procesos necesarios para efectuar la migración en
Un plan de migración que cubra la migración requiere:
ments, herramientas de migración, conversión de producto
y datos, ejecución, verificación y soporte.
La migración de software también puede implicar una serie de
actividades adicionales como

- notificación de intención: una declaración de por qué
  el viejo ambiente ya no se debe soportar
  portado, seguido de una descripción de la nueva
  entorno y su fecha de disponibilidad;
- operaciones paralelas: poner a disposición el
  de los sistemas antiguos y nuevos para que el usuario
  experimenta una transición suave a lo nuevo
  ambiente;
- notificación de finalización: cuando la programación
  se completa la migración de Uled, se envía una notificación
  enviado a todos los interesados;

- revisión postoperatoria: una evaluación de par-
  operación alélica y el impacto de cambiar a
  el nuevo ambiente;
- segmentadores de programas, que seleccionan solo partes de un
  programa afectado por un cambio;
- analizadores estáticos, que permiten una visión general

• archivo de datos: almacenar los datos del software antiguo.

4.5. Jubilación

[1 *, c5s6]

Una vez que el software ha llegado al final de su uso, vida plena, debe ser retirado. Un análisis debería ser realizado para ayudar a hacer la jubilación decisión. Este análisis debe incluirse en el plan de jubilación, que cubre la jubilación requerida mentos, impacto, reemplazo, cronograma y esfuerzo. La accesibilidad de las copias de archivo de datos también puede ser incluido Retirar software implica un número de actividades similares a la migración.

5. Herramientas de mantenimiento de software

[1 *, c6s4] [2 *, c14]

Este tema abarca herramientas que son particularmente importante en el mantenimiento del software donde exista El software está siendo modificado. Ejemplos relacionados la comprensión del programa ing incluye

ing y resúmenes del contenido de un programa;

• analizadores dinámicos, que permiten tainer para rastrear la ruta de ejecución de un programa;

• analizadores de flujo de datos, que permiten Tainer para rastrear todos los flujos de datos posibles de un programa;

• referencias cruzadas, que generan índices de componentes del programa; y

• analizadores de dependencia, que ayudan a mantener Los analizan y entienden la interrelación. se envía entre componentes de un programa.

Las herramientas de ingeniería inversa ayudan al proceso mediante trabajando hacia atrás desde un producto existente a crear artefactos como especificaciones y diseño descripciones, que luego se pueden transformar en generar un nuevo producto a partir de uno antiguo. Principal- los maestros también usan prueba de software, configuración de software gestión de la acción, documentación del software y herramientas de medida de software.

---

115 de 1189.

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	06	**03**	
		g 20	
		un	**08**
	C 14764 20	**Alaska**	
	/ES [1 *] DECIR	**d T[2 *]**	**eed 20 [3 *]**
	O		**norte**
	/ES	**b an**	**S**
		segundo	
	IEEE	**Gru**	
1. Mantenimiento de software Fundamentos			
1.1. Definiciones y terminología	c3	c1s2, c2s2	
1.2. Naturaleza del mantenimiento		c1s3	

1.3. Necesidad de mantenimiento		c1s5	
1.4. Mayoría de costos de mantenimiento		c4s3, c5s5.2	
1.5. Evolución del software		c3s5	
1.6. Categorías de mantenimiento	c3, c6s2	c3s3.1, c4s3	
2. Cuestiones clave en el software Mantenimiento			
2.1. Problemas técnicos			
2.1.1. Comprensión limitada		c6	
2.1.2. Pruebas	c6s2.2.2	c9	
2.1.3. Análisis de impacto	c5s2.5	c13s3	
2.1.4. Mantenibilidad	c6s8, c3s4	c12s5.5	
2.2. Asuntos Gerenciales			
2.2.1. Alineación con Objetivos organizacionales		c4	
2.2.2. Dotación de personal		c4s5, c10s4	
2.2.3. Proceso	c5	c5	
2.2.4. Aspectos organizacionales de Mantenimiento	c7s.2.3	c10	
2.2.5. Outsourcing / Offshoring			todos
2.3. Estimación de costos de mantenimiento			
2.3.1. Estimación de costos	c7s4.1	c7s2.4	

Page 116

	06	**03**	
	C 14764 20 /ES[1 *] DECIRO /ES IEEE	**g 20 un Alaska d T[2 *] b an segundo Gru**	**08 eed[3 *] 20 norte S**
2.3.2. Modelos paramétricos		c12s5.6	
2.3.3. Experiencia		c12s5.5	
2.4. Mantenimiento del software Medición	c6s5	c12, c12s3.1	
2.4.1 Medidas específicas		c12	
3. Proceso de mantenimiento			
3.1. Procesos de mantenimiento	c5	c5	
3.2. Actividades de mantenimiento	c5, c5s3.2.2, c6s8.2, c7s3.3		
3.2.1. Actividades unicas	c3s10, c6s9, c7s2, c7s3	c6, c7	
3.2.2. Actividades de apoyo	c4s1, c5, c6s7	c9	
3.2.3. Planificación de mantenimiento Ocupaciones	c7s2, c7s.3		

3.2.4. Configuración de software administración	c5s1.2.3	c11
3.2.5. Calidad de software	c6s5, c6s7, c6s8	c12s5.3
4. Técnicas de mantenimiento.		
4.1. Comprensión del programa		c6, c14s5
4.2. Reingeniería		c7
4.3. Ingeniería inversa	c6s2	c7, c14s5
4.4. Migración	c5s5	
4.5. Jubilación	c5s6	
5. Herramientas de mantenimiento de software	c6s4	c14

Página 117

LECTURAS ADICIONALES

A. April y A. Abran, *Mantenimiento de software Gestión: evaluación y continua Mejora* [6].

Este libro explora el dominio del software pequeño. procesos de mantenimiento (S3M). Proporciona carretera mapas para mejorar el programa de mantenimiento de software ceses en organizaciones. Describe un software mantenimiento modelo de madurez específico organizado por niveles que permiten la evaluación comparativa y mejora continua. Metas para cada práctica clave se proporcionan áreas de datos y el modelo de proceso previo sented está totalmente alineado con la arquitectura y marco de normas internacionales ISO12207, ISO14764 e ISO15504 y madurez popular modelos como ITIL, CoBIT, CMMI y CM3.

M. Kajko-Mattsson, “Hacia un negocio Modelo de mantenimiento ", IEEE Int'l Conf. Mantenimiento de software [7].

Este documento presenta una visión general de la Correc- Modelo de Madurez de Mantenimiento (CM3). En A diferencia de otros modelos de proceso, CM3 es un modelo cializado, totalmente dedicado al correctivo mantenimiento de software. Ve mantenimiento en términos de las actividades a realizar y sus orden, en términos de la información utilizada por estos actividades, objetivos, reglas y motivaciones para su ejecución, y niveles y roles organizacionales involucrado en varias etapas de un correctivo típico proceso de mantenimiento

Referencias

[1 *] *IEEE Std. 14764-2006 (también conocido como ISO / IEC 14764: 2006) Norma para software Ingeniería: ciclo de vida del software Procesos — Mantenimiento* , IEEE, 2006.

[2 *] P. Grubb y AA Takang, *Software Mantenimiento: Conceptos y práctica* , 2do. ed., World Scientific Publishing, 2003.

[3 *] HM Sneed, "Ofreciendo Software Mantenimiento como servicio offshore ”, *Proc. IEEE Int'l Conf. Mantenimiento del software* (ICSM 08), IEEE, 2008, págs. 1-5.

[4 *] JW Moore, *La hoja de ruta hacia el software Ingeniería: una guía basada en estándares* , Wiley-IEEE Computer Society Press, 2006.

[5] *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010.

[6] A. April y A. Abran, *software Gestión de mantenimiento: evaluación y mejora continua* , Wiley-IEEE Computer Society Press, 2008.

[7] M. Kajko-Mattsson, “Hacia un negocio Modelo de mantenimiento, " *Proc. Conf. Internacional Mantenimiento de software* , IEEE, 2001, pp. 500-509.

CAPÍTULO 6

GESTIÓN DE CONFIGURACIÓN DE SOFTWARE

SIGLAS

CCB	Tablero de control de configuración
CM	Gestión de la configuración
FCA	Auditoría de configuración funcional
PCA	Auditoría de configuración física
SCCB	Control de configuración de software Tablero
LIC	Elemento de configuración de software
SCM	Configuración de software administración
SCMP	Configuración de software Plan de gestión
SCR	Solicitud de cambio de software
SCSA	Estado de configuración del software Contabilidad
SDD	Documento de diseño de software
SEI / CMMI	Instituto de Ingeniería de Software Modelo de Capacidad de Madurez Integración
SQA	Aseguramiento de la calidad del software
SRS	Requisito de software Especificación

INTRODUCCIÓN

Un sistema se puede definir como la combinación de elementos interactivos organizados para lograr uno o propósitos más declarados [1]. La configuración de un sistema es la característica funcional y física tics de hardware o software según lo establecido en técnicas documentación cal o lograda en un producto [1]; eso También se puede considerar como una colección de versiones de hardware, firmware o elementos de software combinado de acuerdo con procedimientos de construcción específicos

para servir a un propósito particular. Configuración man- el envejecimiento (CM), entonces, es la disciplina de la identificación ing la configuración de un sistema en puntos distintos a tiempo para el control sistemático cambios en la configuración y mantenimiento La integridad y trazabilidad de la configuración a lo largo del ciclo de vida del sistema. Es formalmente definido como

> Una disciplina de aplicación técnica y administrativa. dirección y vigilancia istrativa para: iden- tificar y documentar el funcional y físico características de un elemento de configuración, controlar los cambios a esas características, registrar e informar el procesamiento de cambios y estado de implementación y verificar cumplimiento Con los requisitos especificados. [1]

Gestión de configuración de software (SCM) es un proceso de ciclo de vida de software de soporte que beneficia la gestión de proyectos, desarrollo y actividades de mantenimiento, actividades de aseguramiento de la calidad vínculos, así como los clientes y usuarios del final producto.

Los conceptos de gestión de la configuración. se aplica a todos los elementos a controlar, aunque hay algunas diferencias en la implementación entre CM de hardware y CM de software.

SCM está estrechamente relacionado con la calidad del software actividad de aseguramiento de la itidad (SQA). Como se define en el Área de conocimiento de calidad de software (KA), SQA los procesos aseguran que el software productos y procesos en el ciclo de vida del proyecto. cumplir con los requisitos especificados por plan ning, promulgar y realizar un conjunto de actividades para proporcionar la confianza adecuada de que la calidad es siendo incorporado en el software. Las actividades de SCM ayudan para lograr estos objetivos de SQA. En algún proyecto contextos ect, los requisitos específicos de SQA prescriben específicos actividades de SCM.

Figura 6.1. Desglose de temas para el KA de gestión de configuración de software

Las actividades de SCM son gestión y planificación. Ning del proceso SCM, configuración de software identificación, control de configuración de software, contabilidad del estado de la configuración del software, software auditoría de configuración de software y lanzamiento de software gestión y entrega.

El software de gestión de configuración KA está relacionado con todos los otros KAs, ya que el objeto de gestión de la configuración es el pro artefacto duplicado y utilizado en todo el software de ingeniería proceso de neering.

DESGLOSE DE TEMAS PARA CONFIGURACIÓN DE SOFTWARE ADMINISTRACIÓN

El desglose de temas para la configuración del software uration Management KA se muestra en la Figura 6.1.

1. Gestión del proceso SCM

SCM controla la evolución e integridad de un producto identificando sus elementos; gestionando y cambio de control; y verificar, grabar y informar sobre la información de configuración. Desde el perspectiva del ingeniero de software, SCM facilita actividades de desarrollo e implementación de cambios corbatas. Una implementación SCM exitosa requiere cuidadosa planificación y gestión. Esto a su vez, requiere una comprensión de la organización contexto y las restricciones impuestas a diseño e implementación del proceso SCM.

1.1. Contexto Organizacional para SCM

[2 *, c6, ann. D] [3 *, introducción] [4 *, c29]

Para planificar un proceso SCM para un proyecto, es necesario sary entender el contexto organizacional y Las relaciones entre los elementos organizativos. SCM interactúa con varias otras actividades o elementos organizacionales

Los elementos organizativos responsables de la procesos de soporte de ingeniería de software pueden ser estructurado de varias maneras. Aunque la responsabilidad la capacidad para realizar ciertas tareas de SCM podría ser asignado a otras partes de la organización (como la organización de desarrollo), la respuesta general La posibilidad de SCM a menudo recae en una organización distinta elemento nacional o individuo designado.

El software se desarrolla con frecuencia como parte de un sistema más grande que contiene hardware y firmware elementos. En este caso, las actividades de SCM tienen lugar en paralelo con hardware y firmware CM activ- ities y debe ser coherente con el nivel de sistema CM. Tenga en cuenta que el firmware contiene hardware y software; por lo tanto, tanto hardware como software Los conceptos de CM son aplicables.

SCM podría interactuar con el de una organización actividad de aseguramiento de la calidad en temas como gestión de registros y artículos no conformes. En cuanto a lo primero, algunos artículos bajo SCM el control también podría estar sujeto a registros de proyectos disposiciones del aseguramiento de la calidad de la organización programa. La gestión de elementos no conformes es usu- alia la responsabilidad del aseguramiento de la calidad ingeniería emitida por los distintos estándares de organizaciones (ver Apéndice B sobre estándares).

1.3. Planificación para SCM

[2 *, c6, ann. D, ann. E] [3 *, c23] [4 *, c29]

La planificación de un proceso SCM para un determinado el proyecto debe ser consistente con la organización contexto nacional, restricciones aplicables, comunicación orientación aceptada monly, y la naturaleza de la proyecto (por ejemplo, tamaño, seguridad crítica y seguridad). Las principales actividades cubiertas son soft- identificación de la configuración del software, software

actividad; sin embargo, SCM podría ayudar con el seguimiento de
Informes e informes sobre elementos de configuración de software
cayendo en esta categoría.

Quizás la relación más cercana es con el
organización de desarrollo y mantenimiento de software
NIZACIONES Es dentro de este contexto que muchos de
las tareas de control de configuración de software son con-
canalizado Con frecuencia, las mismas herramientas soportan el desarrollo
Opción, mantenimiento y propósitos de SCM.

1.2. Restricciones y orientación para la SCM Proceso

[2 *, c6, ann. D, ann. E] [3 *, c2, c5] [5 *, c19s2.2]

Restricciones que afectan y orientación para el SCM
El proceso proviene de varias fuentes. Poli
cies y procedimientos establecidos en empresas u otros
los niveles organizacionales pueden influir o prescribir
El diseño e implementación del programa SCM
cess para un proyecto dado. Además, el contrato
entre el adquiriente y el proveedor podría
contienen disposiciones que afectan el proceso de SCM. por
ejemplo, ciertas auditorías de configuración pueden ser
requerido, o podría especificarse que ciertos artículos
ser colocado debajo de CM. Cuando los productos de software a
desarrollarse tienen el potencial de afectar al público
seguridad, los organismos reguladores externos pueden imponer
restricciones Finalmente, la vida particular del software
proceso del ciclo elegido para un proyecto de software y
el nivel de formalismo seleccionado para implementar el
software afecta el diseño e implementación de
El proceso SCM.

Orientación para diseñar e implementar un
El proceso SCM también se puede obtener de "best
práctica ", como se refleja en las normas sobre software
control de figuración, estado de configuración del software
contabilidad, auditoría de configuración de software y
gestión y entrega de versiones de software. En
Además, cuestiones como la organización y la respuesta
posibilidades, recursos y horarios, selección de herramientas
e implementación, proveedor y subcontratista
control y control de interfaz son típicamente con-
cluido. Los resultados de la actividad de planificación son
registrado en un Plan SCM (SCMP), que es típico
Sujeto a revisión y auditoría de SQA.

Las estrategias de ramificación y fusión deben ser
cuidadosamente planificado y comunicado, ya que
impactar muchas actividades de SCM. Desde un stand de SCM
punto, una rama se define como un conjunto de fuente en evolución
versiones de archivo [1]. La fusión consiste en combinar
diferentes cambios en el mismo archivo [1]. Este typi-
ocurre cuando más de una persona cambia un
elemento de configuración. Hay muchas ramificaciones y
fusionar estrategias de uso común (ver más
Sección de lecturas para discusión adicional).

El modelo de ciclo de vida de desarrollo de software.
(consulte Modelos de ciclo de vida del software en el software
Proceso de ingeniería KA) también afecta a SCM
actividades, y la planificación SCM debería tomar esto
en cuenta. Por ejemplo, integración continua
es una práctica común en muchos desarrollos de software
enfoques de ment. Típicamente se caracteriza por
frecuentes ciclos de construcción-prueba-implementación. Actividades de SCM
debe planificarse en consecuencia.

1.3.1. Organización y responsabilidades de SCM

[2 *, ann. Ds5, ann. Ds6] [3 *, c10-11] [4 *, introducción, c29]

Para evitar confusiones sobre quién realizará
actividades o tareas SCM dadas, organizacionales

Page 121

los roles que deben participar en el proceso SCM necesitan
ser claramente identificado Responsabilidades específicas
para determinadas actividades o tareas de SCM también deben ser
asignado a entidades organizativas, ya sea por título
o por elemento organizativo. El autor general
ity y los canales de informes para SCM también deben ser
identificado, aunque esto podría lograrse
en la gestión del proyecto o garantía de calidad
etapa de planificación.

1.3.2. Recursos y horarios de SCM

[2 *, ann. Ds8] [3 *, c23]

La planificación de SCM identifica al personal y las herramientas
involucrado en la realización de actividades y tareas de SCM.
Aborda las preguntas de programación estableciendo
secuencias necesarias de tareas SCM e identificar-
ing sus relaciones con los cronogramas del proyecto
e hitos establecidos en la gestión del proyecto
etapa de planificación del ment. Cualquier requisito de entrenamiento
necesario para implementar los planes y capacitar
También se especifican nuevos miembros del personal.

1.3.3. Selección e implementación de herramientas

[3 *, c26s2, c26s6] [4 *, c29s5]

En cuanto a cualquier área de ingeniería de software, el

- Futuro: ¿cuál es el plan para el uso de las herramientas en
  ¿el futuro?
- Cambio: ¿qué tan adaptables son las herramientas?
- Ramificación y fusión: son las capacidades de las herramientas
  Habilidades compatibles con la rama planificada
  estrategias de fusión e integración?
- Integración: hacer las diversas herramientas SCM inte-
  rejilla entre ellos? Con otras herramientas en
  uso en la organización?
- Migración: ¿puede el repositorio mantenido por
  la herramienta de control de versiones se transfiere a otra
  herramienta de control de versiones mientras se mantiene
  historia completa de los elementos de configuración que
  contiene?

SCM generalmente requiere un conjunto de herramientas, como
opuesto a una sola herramienta. Tales conjuntos de herramientas son algunos
tiempos referidos como bancos de trabajo. En tal caso
texto, otra consideración importante en el plan
Ning para la selección de herramientas es determinar si el SCM
el banco de trabajo estará *abierto* (en otras palabras, herramientas
de diferentes proveedores se utilizarán en diferentes
actividades ent del proceso SCM) o *integradas*
(donde los elementos del banco de trabajo están diseñados
trabajar juntos).

El tamaño de la organización y el tipo de
los proyectos involucrados también pueden afectar la selección de herramientas

selección e implementación de herramientas SCM debe ser cuidadosamente planeado Las siguientes preguntas Se deben considerar las siguientes opciones:

- Organización: lo que motiva la adquisición de herramientas. ción desde una perspectiva organizacional?
- Herramientas: ¿podemos usar herramientas comerciales o desarrollarlos nosotros mismos?
- Medio ambiente: cuáles son las limitaciones impuesto por la organización y sus técnicas contexto cal?
- Legado: cómo utilizarán (o no) los proyectos nuevas herramientas?
- Financiación: ¿quién pagará por las herramientas? adquisición, mantenimiento, capacitación y personalización?
- Alcance: cómo se implementarán las nuevas herramientas: por ejemplo, a través de toda la organización o solo en proyectos específicos?
- Propiedad: quién es responsable de la introducción duction de nuevas herramientas?

(Consulte el tema 7, Gestión de configuración de software-Herramientas de ment).

1.3.4. Control de proveedores / subcontratistas

[2 *, c13] [3 *, c13s9, c14s2]

Un proyecto de software puede adquirir o hacer uso de productos de software comprados, como compiladores u otras herramientas. La planificación de SCM considera si y cómo se tomarán estos elementos en la configuración control de la acción (por ejemplo, integrado en el proyecto) ect bibliotecas) y cómo serán los cambios o actualizaciones evaluado y gestionado.

Consideraciones similares se aplican a los subcontratados software. Cuando se utiliza software subcontratado, tanto los requisitos de SCM que se impondrán al proceso SCM del subcontratista como parte de subcontrato y los medios para monitorear es necesario que se establezca Este último incluye consideración de qué información SCM debe ser disponible para monitoreo de cumplimiento efectivo.

Page 122

1.3.5. Control de interfaz

[2 *, c12] [3 *, c24s4]

Cuando un elemento de software interactuará con otro elemento de software o hardware, un cambio a cualquier artículo puede afectar al otro. Planeando para El proceso SCM considera cómo la interfaz Se identificarán los elementos y cómo los cambios en el Los artículos serán gestionados y comunicados. los El rol de SCM puede ser parte de un nivel de sistema más grande proceso para la especificación y control de la interfaz; puede involucrar especificaciones de interfaz, interfaz planes de control y documentos de control de interfaz. En este caso, planificación SCM para control de interfaz tiene lugar dentro del contexto del sistema proceso de nivel.

1.4. Plan SCM

[2 *, ann. D] [3 *, c23] [4 *, c29s1]

Los resultados de la planificación de SCM para un proyecto se registran en una gestión de configuración de software plan de ment (SCMP), un "documento vivo" que sirve como referencia para el proceso SCM. Es mantenido (es decir, actualizado y aprobado) como necesario durante el ciclo de vida del software. En imple- Al mencionar el SCMP, generalmente es necesario desarrollar una serie de subordinados más detallados procedimientos que definen cómo los requisitos específicos se llevará a cabo durante las actividades del día a día por ejemplo, qué estrategias de ramificación serán utilizado y con qué frecuencia se producen compilaciones y Se ejecutan pruebas acopladas de todo tipo.

Orientación sobre la creación y mantenimiento de un SCMP, basado en la información producida por la actividad de planificación, está disponible en un número de fuentes, como [2 *]. Esta referencia proporciona requisitos para que la información sea contenida en un SCMP; también define y describe seis gatos documentos de información SCM que se incluirán en un SCMP:

- Horarios SCM (coordinación con otros actividades del proyecto)
- Recursos SCM (herramientas, recursos físicos, y recursos humanos)
- Mantenimiento SCMP.

1.5. Vigilancia de la configuración del software administración

[3 *, c11s3]

Después de que se haya implementado el proceso SCM, puede ser necesario cierto grado de vigilancia para garantizar que las disposiciones del SCMP sean Realizado correctamente. Es probable que haya requisitos específicos de SQA para garantizar el cumplimiento con procesos y procedimientos SCM especificados. La persona responsable de SCM asegura que aquellos con la responsabilidad asignada realizan las tareas SCM definidas correctamente. El software autoridad de aseguramiento de la calidad, como parte de un cumplimiento actividad de auditoría previa, también podría realizar esto vigilancia.

El uso de herramientas SCM integradas con proceso capacidad de control puede hacer la vigilancia Tarea más fácil. Algunas herramientas facilitan el proceso de comunicación aplicación a la vez que proporciona flexibilidad para el soft- ingeniero de almacén para adaptar procedimientos. Otras herramientas hacer cumplir el proceso, dejando al ingeniero de software con menos flexibilidad Requisitos de vigilancia y el nivel de flexibilidad que se debe proporcionar a ingeniero de software son consideraciones importantes en la selección de herramientas.

1.5.1. Medidas y medidas de SCM

[3 *, c9s2, c25s2 – s3]

Las medidas de SCM pueden diseñarse para proporcionar información específica sobre el producto en evolución o para proporcionar información sobre el funcionamiento de la SCM proceso. Un objetivo relacionado de monitorear el SCM proceso es descubrir oportunidades para el proceso

- Introducción (propósito, alcance, términos utilizados)
- Gestión de SCM (organización, respuesta-responsabilidades, autoridades, políticas aplicables, directivas y procedimientos)
- Actividades SCM (identificación de configuración, control de configuración, etc.)

mejora. Mediciones de procesos SCM proporcionar un buen medio para monitorear el efecto Actividad de SCM de forma continua. Estas medidas son útiles para caracterizar ing el estado actual del proceso, así como en proporcionando una base para hacer comparaciones sobre hora. El análisis de las mediciones puede producir

ideas que conducen a cambios en el proceso y correcciones actualizaciones de patrocinio al SCMP.

Bibliotecas de software y varias herramientas SCM las capacidades proporcionan fuentes para extraer información mación sobre las características de la SCM proceso (además de proporcionar proyecto y personal-información de gestión). Por ejemplo, información sobre el tiempo requerido para lograr varios tipos de cambios serían útiles en una evaluación ción de los criterios para determinar qué niveles de la autoridad es óptima para autorizar ciertos tipos de cambios y para estimar cambios futuros.

Se debe tener cuidado para mantener el enfoque de la vigilancia sobre los conocimientos que se pueden obtener de las medidas, no en las medidas sí mismos. Discusión del proceso de software y La medición del producto se presenta en el Soft-Proceso de Ingeniería de mercancías KA. Mea de software los programas de aseguramiento se describen en el Software Dirección de Ingeniería KA.

1.5.2. Auditorías en proceso de SCM

[3 *, c1s1]

Se pueden realizar auditorías durante el software proceso de ingeniería para investigar el estado actual tus de elementos específicos de la configuración o para evaluar la implementación del proceso SCM. La auditoría en proceso de SCM proporciona una mal mecanismo para monitorear aspectos seleccionados del proceso y puede coordinarse con el Función SQA (ver tema 5, Configuración del software-Auditoría).

2. Identificación de la configuración del software

[2 *, c8] [4 *, c29s1.1]

La identificación de la configuración del software identifica elementos a controlar, establece identificación esquemas para los artículos y sus versiones, y establece las herramientas y técnicas que se utilizarán en Adquisición y gestión de artículos controlados. Estas actividades proporcionan la base para el otro SCM ocupaciones.

2.1. Identificación de elementos a controlar

[2 *, c8s2.2] [4 *, c29s1.1]

Uno de los primeros pasos para controlar el cambio es identificando los elementos de software a controlar. Esto implica comprender la configuración del software uración dentro del contexto de la configuración del sistema ración, selección de elementos de configuración de software, desarrollar una estrategia para etiquetar elementos de software y describiendo sus relaciones, e identificando tanto las líneas de base que se utilizarán como el procedimiento para una adquisición de referencia de los artículos.

2.1.1. Configuración de software

[1, c3]

La configuración del software es funcional y física. ical características de hardware o software como conjunto adelante en la documentación técnica o logrado en un producto. Se puede ver como parte de un configuración del sistema.

2.1.2. Elemento de configuración de software

[4 *, c29s1.1]

Un elemento de configuración (CI) es un elemento o agregado gation de hardware o software o ambos que es diseñado para ser administrado como una sola entidad. Un suave El elemento de configuración de software (SCI) es una entidad de software que se ha establecido como un elemento de configuración [1] El SCM generalmente controla una variedad de artículos Además del código en sí. Artículos de software con el potencial para convertirse en LIC incluye planes, especificaciones caciones y documentación de diseño, material de prueba riales, herramientas de software, código fuente y código ejecutable, bibliotecas de códigos, datos y diccionarios de datos, y documentación para instalación, mantenimiento, operaciones y uso de software.

Seleccionar LIC es un proceso importante en que se debe lograr un equilibrio entre pro-Vidriando una visibilidad adecuada para el control del proyecto. plantea y proporciona un número manejable de Artículos controlados.

2.1.3. Elemento de configuración de software Relaciones

[3 *, c7s4]

Relaciones estructurales entre los seleccionados. Las LME y sus partes constituyentes afectan a otros Actividades o tareas de SCM, como software construir o analizar el impacto de la propuesta cambios Seguimiento adecuado de estas relaciones. También es importante para apoyar la trazabilidad. El diseño del esquema de identificación de LIC

Figura 6.2. Adquisicion de Articulos

debe considerar la necesidad de mapear los elementos identificados a la estructura del software, así como a la necesidad de Apoyar la evolución de los elementos de software y sus relaciones

2.1.4. Versión del software [1, c3] [4 *, c29s3]

Los elementos de software evolucionan como un proyecto de software cede Una versión de un elemento de software es un identi instancia fied de un artículo. Se puede considerar como un estado de un elemento en evolución. Una variante es una versión de un programa resultante de la aplicación de soft- diversidad de artículos.

2.1.5. Base [1, c3]

Una línea base de software es una versión aprobada formalmente sion de un elemento de configuración (independientemente que se designa formalmente y se fija en un determinado tiempo durante el ciclo de vida del elemento de configuración. El término también se usa para referirse a un verso particular sion de un elemento de configuración de software que tiene acordado. En cualquier caso, la línea base puede solo se cambiará mediante un cambio formal procedimientos de control. Una línea de base, junto con todos los cambios aprobados a la línea de base, representa el configuración aprobada actual.

Las líneas de base comúnmente utilizadas incluyen funcional, asignado, desarrollo y producto

Líneas de base La línea base funcional corresponde a los requisitos del sistema revisados. El alo- la línea de base correspondiente corresponde a la revisada especificación de requisitos de software y software especificación de requisitos de interfaz de software. los línea de base de desarrollo representa la evolución configuración de software en momentos seleccionados durante El ciclo de vida del software. Cambiar autoridad para esta línea de base generalmente descansa principalmente con el organización de desarrollo pero puede ser compartida con otras organizaciones (por ejemplo, SCM o Prueba). La línea de base del producto corresponde a la producto de software completo entregado para sys- Integración tem. Las líneas de base que se utilizarán para un proyecto dado, junto con los niveles asociados de autoridad necesaria para la aprobación del cambio, son típicos identificado en el SCMP.

2.1.6. Adquisición de elementos de configuración de software [3 *, c18]

Los elementos de configuración del software se colocan debajo de Control SCM en diferentes momentos; es decir, son incorporado a una línea base particular en un momento particular punto principal en el ciclo de vida del software. El desencadenante evento es la finalización de alguna forma de formal tarea de aceptación, como una revisión formal. Figura 6.2 caracteriza el crecimiento de los artículos de línea base como El ciclo de vida continúa. Esta cifra se basa en el modelo de cascada solo con fines ilustrativos; los subíndices utilizados en la figura indican versiones

de los artículos en evolución. La solicitud de cambio de software (SCR) se describe en la sección 3.1.

Al adquirir una LME, su origen e integración inicial Se debe establecer la ciudad. Siguiendo el acervo ción de una LME, los cambios al artículo deben ser formal aprobado según corresponda para el LIC y la línea de base involucrada, como se define en el SCMP. Después de la aprobación, el artículo se incorpora a la línea de base del software de acuerdo con el apropiado

que cambios hacer, la autoridad para aprobar ing ciertos cambios, soporte para la implementación ción de esos cambios, y el concepto de formal desviaciones de los requisitos del proyecto, así como exenciones de ellos. Información derivada de estos las actividades son útiles para medir el tráfico de cambio y roturas, así como aspectos de la reelaboración.

3.1. Solicitar, evaluar y aprobar

procedimiento.

2.2. Biblioteca de software

[3 *, c1s3] [4 *, c29s1.2]

Una biblioteca de software es una colección controlada de software y documentación relacionada diseñada para ayuda en el desarrollo, uso o mantenimiento de software [1] También es instrumental en el lanzamiento de software actividades de gestión y entrega. Varios tipos de podrían usarse bibliotecas, cada una correspondiente a la nivel de madurez particular del elemento de software. por ejemplo, una biblioteca en funcionamiento podría soportar la modificación y una biblioteca de soporte de proyectos podría soportar pruebas ing, mientras que una biblioteca maestra podría usarse para modificar Productos lavados. Un nivel apropiado de SCM control (línea de base asociada y nivel de autoridad para cambio) está asociado con cada biblioteca. Seguridad, en términos de control de acceso y facilidades de respaldo vínculos, es un aspecto clave de la gestión de la biblioteca.

Las herramientas utilizadas para cada biblioteca deben admitir el control SCM necesita esa biblioteca, tanto en términos de control de LIC y control de acceso a la biblioteca. En el nivel de la biblioteca de trabajo, esto es una capacidad de gestión de código que sirve para desarrollar ers, mantenedores y SCM. Se centra en el hombre. envejecimiento de las versiones de elementos de software mientras se supervisa y controla portar las actividades de múltiples desarrolladores. A mayores niveles de control, el acceso es más restringido y SCM es el usuario principal.

Estas bibliotecas también son una fuente importante. de información para mediciones de trabajo y Progreso.

3. Control de configuración de software

[2 *, c9] [4 *, c29s2]

El control de la configuración del software se refiere con la gestión de cambios durante el software ciclo vital. Cubre el proceso para determinar

Cambios de software

[2 *, c9s2.4] [4 *, c29s2]

El primer paso en la gestión de cambios controlados elementos determina qué cambios hacer. los proceso de solicitud de cambio de software (ver un típico flujo de un proceso de solicitud de cambio en la Figura 6.3) proporciona procedimientos formales para enviar y registrar solicitudes de cambio, evaluar el potencial costo real e impacto de un cambio propuesto, y aceptar, modificar, diferir o rechazar El cambio propuesto. Una solicitud de cambio (CR) es una modificación para ampliar o reducir el alcance del proyecto; modificar políticas, procesos, planes o procedimientos; modificar costos o presupuestos; o revisar horarios [1] Solicitudes de cambios en la configuración del software Los artículos pueden ser originados por cualquier persona en cualquier punto en el ciclo de vida del software y puede incluir Una solución sugerida y prioridad solicitada. Uno fuente de un CR es el inicio de la corrección acción en respuesta a informes de problemas. Sin importar de la fuente, el tipo de cambio (por ejemplo, defecto o mejora) generalmente se registra en el Software CR (SCR).

Esto proporciona una oportunidad para rastrear defectos y medición de actividad de cambio de recolección mientras se administra cambio. Una vez que se recibe un SCR, Una evaluación técnica (también conocida como impacto análisis) se realiza para determinar el alcance de las modificaciones que serían necesarias deberían Se aceptará la solicitud de cambio. Un buen sub- posición de las relaciones entre software (y, posiblemente, hardware) es importante para esta tarea. Finalmente, una autoridad establecida —com mensurate con la línea de base afectada, la LME involucrados y la naturaleza del cambio evaluar los aspectos técnicos y gerenciales de la solicitud de cambio y aceptar, modificar, rechazar o diferir el cambio propuesto.

Page 126

Figura 6.3. Flujo de un proceso de control de cambios

3.1.1. Tablero de control de configuración de software

[2 *, c9s2.2] [3 *, c11s1] [4 *, c29s2]

La autoridad para aceptar o rechazar propuestas los cambios recaen en una entidad típicamente conocida como

Decisiones de CCB y proceso de cambio de informes información. Un enlace entre la capacidad de esta herramienta y el sistema de notificación de problemas puede facilitar el seguimiento de soluciones para problemas reportados.

Tablero de control de configuración (CCB). En menor proyectos, esta autoridad en realidad puede residir con el líder o un individuo asignado en lugar de un Junta de varias personas. Puede haber múltiples niveles de autoridad de cambio dependiendo de una variedad de criterios, como la criticidad del elemento involucrado, la naturaleza del cambio (por ejemplo, impacto en presupuesto y calendario), o el proyecto actual punto en el ciclo de vida. La composición de la Los CCB utilizados para un sistema dado varían según en estos criterios (un representante de SCM siempre estar presente). Todos los interesados, apropiados a nivel del CCB, están representados. Cuando El alcance de la autoridad de un CCB es estrictamente blanda Ware, se conoce como una configuración de software Junta de Control (SCCB). Las actividades del CCB normalmente están sujetos a auditorías de calidad de software revisión.

3.1.2. Proceso de solicitud de cambio de software [3 *, c1s4, c8s4]

Un programa de solicitud de cambio de software (SCR) efectivo El proceso requiere el uso de herramientas de apoyo y proceso procedimientos para originar solicitudes de cambio, hacer cumplir ing el flujo del proceso de cambio, capturando

3.2. Implementación de cambios de software [4 *, c29]

Los SCR aprobados se implementan utilizando el procedimientos de software definidos de acuerdo con los requisitos de horario aplicables. Desde un número de SCR aprobados podrían implementarse simultáneamente, es necesario proporcionar un medio para rastrear en qué SCR se incorporan versiones de software particulares y líneas de base. Como parte del cierre del proceso de cambio, comp los cambios realizados pueden someterse a auditorías de configuración y verificación de calidad del software, esto incluye asegurando que solo se hayan realizado cambios aprobados hecho. El proceso de solicitud de cambio de software descrito anteriormente normalmente documentará el información de aprobación de SCM (y otra) para el cambio.

Los cambios pueden ser compatibles con el código fuente ver- herramientas de control de sion. Estas herramientas permiten un equipo de ingenieros de software, o un solo ingeniero de software, para rastrear y documentar cambios en el código fuente. Estas herramientas proporcionan un repositorio único para almacenar código fuente, puede evitar más de un soft- ingeniero de hardware de editar el mismo módulo en al mismo tiempo, y registre todos los cambios realizados en

Page 127

código fuente. Los ingenieros de software verifican los módulos fuera del repositorio, hacer cambios, documentar los cambios y luego guarde los módulos editados en el repositorio Si es necesario, los cambios también pueden ser descartado, restaurando una línea de base anterior. Más herramientas poderosas pueden soportar el desarrollo paralelo y entornos distribuidos geográficamente.
Estas herramientas pueden manifestarse como separadas, aplicaciones especializadas bajo el control de un grupo independiente de SCM. También pueden aparecer como parte integrada de la ingeniería de software ambiente. Finalmente, pueden ser tan elementales como un sistema de control de cambio rudimentario proporcionado con un sistema operativo

3.3. Desviaciones y exenciones [1, c3]

Las restricciones impuestas a un ingeniero de software esfuerzo o las especificaciones producidas durante el las actividades de desarrollo pueden contener disposiciones que no puede satisfacerse en el punto designado en el ciclo de vida Una desviación es un autor escrito rización, otorgada antes de la fabricación de un elemento, para apartarse de una actuación particular o requisito de diseño para un número específico de unidades o un período de tiempo específico. Una renuncia es un escrito diez autorizaciones para aceptar un elemento de configuración otro elemento designado que se encuentra, durante la producción o después de haber sido sometido a inspección, para apartarse de los requisitos especificados pero nunca sin embargo, se considera adecuado para su uso tal cual o después retrabajo por un método aprobado. En estos casos, un proceso formal se utiliza para obtener la aprobación de desviaciones o exenciones de las disposiciones.

4. Contabilidad del estado de la configuración del software

sistema de información, el estado de configuración informa- mación que se gestionará para la configuración en evolución Se deben identificar, recopilar y mantener las opciones. Se necesita diversa información y medidas para apoyar el proceso de SCM y cumplir con los requisitos figuración del estado que informa las necesidades de la gerencia, ingeniería de software y otras actividades relacionadas. Los tipos de información disponibles incluyen el identificación de configuración aprobada, así como la identificación y el estado actual de implementación Tus cambios, desviaciones y exenciones.

Es necesaria alguna forma de soporte automatizado de herramientas sario para lograr la recopilación de datos SCSA y tareas de informes; esto podría ser una capacidad de base de datos ity, una herramienta independiente, o una capacidad de un mayor, herramienta integrada entorno.

4.2. Informes de estado de configuración de software [2 *, c10s2.4] [3 *, c1s5, c9s1, c17]

La información reportada puede ser utilizada por varios elementos organizativos y de proyecto, incluidos el equipo de desarrollo, el equipo de mantenimiento, gestión de proyectos y actividades de calidad de software corbatas. Los informes pueden tomar la forma de consultas ad hoc Riesgos para responder preguntas específicas o el periódico producción de informes prediseñados. Alguna información ción producida por la actividad contable del estado durante el curso del ciclo de vida podría convertirse registros de aseguramiento de calidad.

Además de informar el estado actual de la configuración, la información obtenida por el SCSA puede servir como base de varias medidas ments. Los ejemplos incluyen el número de cambio solicitudes por LIC y el tiempo promedio necesario para implementar una solicitud de cambio.

[2 *, c10]

Contabilidad del estado de la configuración del software (SCSA) es un elemento de configuración de gestión de configuración. El registro y la presentación de informes de información es necesario gestionar una configuración de manera efectiva

4.1. Información del estado de la configuración del software [2 *, c10s2.1]

La actividad SCSA diseña y opera un sistema tem para la captura y reporte de los necesarios información a medida que avanza el ciclo de vida. Como en cualquier

5. Auditoría de configuración de software [2 *, c11]

Una auditoría de software es un examen independiente. ción de un producto de trabajo o conjunto de productos de trabajo para evaluar el cumplimiento de especificaciones, estándares, acuerdos contractuales u otros criterios [1].

Las auditorías se llevan a cabo de acuerdo con un bien definido proceso que consiste en varios roles de auditor y responsabilidades. En consecuencia, cada auditoría debe ser cuidadosamente planeado Una auditoría puede requerir un número Ber de individuos para realizar una variedad de tareas en un período de tiempo bastante corto Herramientas para apoyar

La planificación y la realización de una auditoría pueden ser muy importantes facilitar el proceso

La auditoría de configuración de software determina la medida en que un artículo cumple con los requisitos Características funcionales y físicas. Informal Se pueden realizar auditorías de este tipo en puntos clave en el ciclo de vida Dos tipos de auditorías formales podrían y ser requerido por el contrato vigente (para examen ple, en contratos que cubren software crítico): el Auditoría de configuración funcional (FCA) y la Auditoría de configuración física (PCA). Exitoso la finalización de estas auditorías puede ser un requisito previo para el establecimiento de la línea de base del producto.

5.1. Auditoría de configuración funcional de software [2 *, c11s2.1]

El propósito del software FCA es asegurar que el elemento de software auditado es consistente con su especificaciones de gobierno. La salida del soft- actividades de verificación y validación de artículos (ver Verificación y Validación en el Software Quality KA) es una entrada clave para esta auditoría.

5.2. Auditoría de configuración física de software [2 *, c11s2.2]

El propósito de la configuración física del software La auditoría de la nación (PCA) es para asegurar que el diseño y la documentación de referencia es coherente con el producto de software as-built.

5.3. Auditorías en proceso de una línea de base de software [2 *, c11s2.3]

Como se mencionó anteriormente, se pueden realizar auditorías durante el proceso de desarrollo para investigar El estado actual de elementos específicos de la figuración. En este caso, se podría aplicar una auditoría a los elementos de referencia muestreados para garantizar que la forma es consistente con las especificaciones o para asegurar que la documentación en evolución continúe ser coherente con el elemento de referencia en desarrollo.

6. Gestión de versiones de software y Entrega [2 *, c14] [3 *, c8s2]

En este contexto, el *lanzamiento se* refiere a la distribución ción de un elemento de configuración de software fuera la actividad de desarrollo; esto incluye interna lanzamientos así como distribución a clientes. Cuando diferentes versiones de un elemento de software están disponibles para la entrega (como versiones para diferentes plataformas formas o versiones con diferentes capacidades), es frecuentemente necesario para recrear versiones específicas y empacar los materiales correctos para la entrega de la versión. La biblioteca de software es un elemento clave. en la realización de tareas de liberación y entrega.

6.1. Edificio de software [4 *, c29s4]

La creación de software es la actividad de combinar el versiones correctas de elementos de configuración de software, utilizando los datos de configuración apropiados, en un programa ejecutable para entrega a un cliente o otro destinatario, como la actividad de prueba. por sistemas con hardware o firmware, el ejecutable programa se entrega a la actividad de construcción del sistema ity. Las instrucciones de compilación aseguran que la compilación adecuada los pasos se toman en la secuencia correcta. Adicionalmente para crear software para nuevas versiones, generalmente es también es necesario para que SCM tenga la capacidad de reproducir versiones anteriores para recuperación, prueba, mantenimiento o propósitos de liberación adicionales.

El software se construye utilizando versiones particulares de herramientas de apoyo, como compiladores (ver Conceptos básicos de la apiladora en los fundamentos de informática KA). Puede ser necesario reconstruir una copia exacta de un elemento de configuración de software creado anteriormente. En este caso, herramientas de soporte y compilación asociada las instrucciones deben estar bajo control SCM para garantizar la disponibilidad de las versiones correctas de herramientas.

La capacidad de una herramienta es útil para seleccionar el versiones correctas de elementos de software para un objetivo determinado entorno y para automatizar el proceso de construir el software a partir de las versiones seleccionadas y datos de configuración apropiados. Para proyectos con desarrollo paralelo o distribuido ronments, esta capacidad de herramienta es necesaria. Más entornos de ingeniería de software proporcionan esto capacidad. Estas herramientas varían en complejidad de exigir al ingeniero de software que aprenda un lenguaje de secuencia de comandos especializado para gráficos enfoques que ocultan gran parte de la complejidad de una instalación de construcción "inteligente".

El proceso de construcción y los productos a menudo son sub- Objeto a la verificación de la calidad del software. Salidas de

el proceso de compilación podría ser necesario para futuras referencias
ence y puede convertirse en registros de garantía de calidad.

6.2. Gestión de versiones de software
[4 *, c29s3.2]

La gestión de versiones de software abarca el
identificación, empaque y entrega de
elementos de un producto, por ejemplo, un ejecutor
programa capaz, documentación, notas de la versión y
datos de configuración Dado que el producto cambia
puede ocurrir de manera continua, una preocupación para
la gestión de versiones determina cuándo emitir
Un lanzamiento. La gravedad de los problemas abordados.
por la liberación y las mediciones de la falla den-
Las situaciones de lanzamientos anteriores afectan esta decisión. Los
La tarea de embalaje debe identificar qué artículos del producto
deben ser entregados y luego seleccione el correcto
variantes de esos elementos, dada la aplicación prevista
cación del producto. El documento informativo
se conoce el contenido físico de una versión
como documento de descripción de versión . La liberación
las notas generalmente describen nuevas capacidades, conocidas
problemas y requisitos de plataforma necesarios
para el correcto funcionamiento del producto. El paquete a ser
lanzado también contiene instalación o actualización
instrucciones. Esto último puede ser complicado por el
hecho de que algunos usuarios actuales pueden tener versiones
que son varios lanzamientos antiguos. En algunos casos, liberar
la administración puede ser requerida para rastrear
distribución del producto a varios clientes
o sistemas de destino, por ejemplo, en un caso donde
el proveedor debía notificar a un cliente de
Problemas recientemente reportados. Finalmente, un mecanismo
para asegurar la integridad del artículo lanzado puede ser
implementado, por ejemplo, lanzando un digital
firma con ella.

Se necesita una capacidad de herramienta para soportar
Estas funciones de administración de versiones. Es uso
Ful para tener una conexión con la capacidad de la herramienta
Apoyar el proceso de solicitud de cambio para
mapear los contenidos de lanzamiento a los SCR que han sido
recibido. Esta capacidad de herramienta también podría mantener
información sobre varias plataformas de destino y en
Varios entornos de clientes.

7. Herramientas de gestión de configuración de software
[3 *, c26s1] [4 *, c8s2]

Cuando se discute la gestión de la configuración del software,
herramientas de ayuda, es útil clasificarlas. SCM
las herramientas se pueden dividir en tres clases en términos
del alcance en el que brindan apoyo: indi-
soporte visual, soporte relacionado con proyectos y apoyo
apoyo a todo el proceso del pany.

Las herramientas de soporte individual son apropiadas y
típicamente suficiente para organizaciones pequeñas o
grupos de desarrollo sin variantes de sus
Los productos de software u otros SCM complejos requieren:
ments. Incluyen:

- Herramientas de control de versiones: seguimiento, documento y almacenar elementos de configuración individuales como código fuente y documentación externa.
- Construir herramientas de manejo: en su forma más simple, tales herramientas compilan y vinculan un ejecutable versión del software Más avanzado las herramientas de construcción extraen la última versión de el software de control de versiones, realice Controles de ity, ejecutar pruebas de regresión y producir diversas formas de informes, entre otras tareas.
- Herramientas de control de cambios: apoyan principalmente control de solicitudes de cambio y eventos notificación (por ejemplo, estado de solicitud de cambio cambios, hitos alcanzados).

Las herramientas de soporte relacionadas con el proyecto son principalmente soporte
gestión del espacio de trabajo para equipos de desarrollo
e integradores; son típicamente capaces de suplir
Puesto distribuido entornos de desarrollo. Tal
las herramientas son apropiadas para organizaciones medianas y grandes
Zations con variantes de sus productos de software
y desarrollo paralelo pero sin certificación
requisitos

Las herramientas de soporte de procesos de toda la empresa pueden tipificar-
Automatizar adecuadamente partes de un pro-
cess, brindando soporte para la gestión del flujo de trabajo
nenciones, roles y responsabilidades. Ellos pueden
para manejar muchos elementos, datos y ciclos de vida. Tal
Las herramientas se suman al soporte relacionado con el proyecto al apoyar
un proceso de desarrollo más formal, que incluye
requisitos de certificación

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

-2012 03 06

ile 2011

	[2 *] **IEEE 828**	**culo 20** **H**	**oore 20** **METRO**	**erv[4 *]** **metro** **om** **S**
1. Gestión de la SCM **Proceso**				
1.1. Contexto Organizacional para SCM	c6, ann.D	Introducción		c29
1.2. Restricciones y Orientación para el proceso SCM	c6, ann.D, Ana	c2	c19s2.2	introducción c29
1.3. Planificación para SCM	c6, ann.D, Ana	c23		c29
1.3.1. Organización SCM y Responsabilidades	ann.Ds5–6	c10–11		introducción c29
1.3.2. Recursos de SCM y Horarios	ann.Ds8	c23		
1.3.3. Selección de herramientas y Implementación		c26s2; s6		c29s5
1.3.4. Vendedor / Subcontratista Controlar	c13	c13s9 – c14s2		
1.3.5. Control de interfaz	c12	c24s4		
1.4. Plan SCM	ann.D	c23		c29s1
1.5. Vigilancia de software Gestión de la configuración		c11s3		
1.5.1. Medidas SCM y Medición		c9s2; c25s2 – s3		
1.5.2. Auditorías en proceso de SCM		c1s1		
2. Configuración del software Identificación				c29s1.1
2.1. Identificando artículos para ser Revisado	c8s2.2			c29s1.1
2.1.1. Configuración de software				
2.1.2. Configuración de software ít				c29s1.1
2.1.3. Configuración de software Relaciones de artículos		c7s4		
2.1.4. Versión del software				c29s3

Página 131

	-2012 **[2 *]** **IEEE 828**	**03** **culo 20** **H** [3 *]	**06** **[5 *]** **oore 20** **METRO**	**ile 2011** **erv[4 *]** **metro** **om** **S**
2.1.5. Base				
2.1.6. Adquisición de software Artículos de configuración		c18		
2.2. Biblioteca de software		c1s3		c29s1.2
3. Configuración del software Controlar	c9			c29s2
3.1. Solicitar, evaluar y Aprobar cambios de software	c9s2.4			c29s2
3.1.1. Configuración de software Tabla de control	c9s2.2	c11s1		c29s2

3.1.2. Cambio de software Proceso de solicitud		c1s4, c8s4	
3.2. Implementando Software Cambios			c29
3.3. Desviaciones y exenciones			
4. Configuración del software Contabilidad de estado	c10		
4.1. Configuración de software Informacion de estado	c10s2.1		
4.2. Configuración de software Informes de estado	c10s2.4	c1s5, c9s1, c17	
5. Configuración del software Revisión de cuentas	c11		
5.1. Software funcional Auditoría de configuración	c11s2.1		
5.2. Software físico Auditoría de configuración	c11s2.2		
5.3. Auditorías en proceso de un Línea base de software	c11s2.3		
6. Lanzamiento de software Gestión y entrega	c14	c8s2	c29s3
6.1. Edificio de software			c29s4
6.2. Lanzamiento de software administración			c29s3.2
7. Configuración del software Herramientas administrativas		c26s1	

Page 132

LECTURAS ADICIONALES

Stephen P. Berczuk y Brad Appleton, *Gestión de configuración de software Patrones: trabajo en equipo efectivo, práctico Integración* [6].

Este libro expresa prácticas útiles de SCM y estrategias como patrones. Los patrones se pueden implementar Se utilizan varias herramientas, pero se expresan de una manera agnóstica de herramientas.

"CMMI para el Desarrollo", Versión 1.3, pp. 137-147 [7].

Este modelo presenta una colección de las mejores prácticas. consejos para ayudar a las organizaciones de desarrollo de software mejorar sus procesos. En el nivel de madurez 2, se sugiere actividades de gestión de configuración.

Referencias

[1] *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010.

[2 *] *IEEE Std. 828-2012, estándar para Gestión de configuración en sistemas y Ingeniería de Software* , IEEE, 2012.

[3 *] AMJ Hass, *Gestión de la configuración Principios y prácticas* , 1ª ed., Addison-Wesley, 2003.

[4 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[5 *] JW Moore, *La hoja de ruta hacia el software Ingeniería: una guía basada en estándares* , Wiley-IEEE Computer Society Press, 2006.

[6] SP Berczuk y B. Appleton, *Software Patrones de gestión de configuración: Trabajo en equipo efectivo, integración práctica* , Addison-Wesley Professional, 2003.

[7] Equipo de producto CMMI, "CMMI para Desarrollo, Versión 1.3, "Software Instituto de Ingeniería, 2010; http: // resources.sei.cmu.edu/library/asset-view. cfm? assetID = 9661 .

CAPÍTULO 7

GESTIÓN DE INGENIERÍA DE SOFTWARE

SIGLAS

PMBOK ® Guía	*Guía para la gestión de proyectos Cuerpo de conocimientos*
SDLC	Ciclo de vida del desarrollo de programas
SEM	Gerencia de Ingeniería de Software
SQA	Aseguramiento de la calidad del software
SWX	*Extensión de software para PMBOK ® Guía*
WBS	Estructura de desglose del trabajo

INTRODUCCIÓN

La gestión de la ingeniería de software se puede definir como la aplicación de actividades de gestión ning, coordinar, medir, monitorear, controlar trolling e informes [1] para garantizar que el software los productos y servicios de ingeniería de software son entregado de manera eficiente, efectiva y en beneficio de las partes interesadas. La disciplina relacionada de gestionar El ment es un elemento importante de todo el conocimiento áreas (KA), pero por supuesto es más relevante para este KA que a otros KAs. La medición también es un aspecto importante de todos los KAs; el tema de la medida Los programas de ment se presentan en este KA.

En cierto sentido, debería ser posible gestionar un proyecto de ingeniería de software de la misma manera Se gestionan otros esfuerzos complejos. Sin embargo, hay aspectos específicos de proyectos de software y procesos del ciclo de vida del software que complican gestión eficaz, incluidos estos:

- Los clientes a menudo no saben lo que se necesita o Lo que es factible.
- Los clientes a menudo carecen de aprecio por el plexidades inherentes a la ingeniería de software, particularmente con respecto al impacto del cambio requisitos de ing.
- Es probable que aumente la comprensión y condiciones cambiantes generarán nuevas o requisitos de software modificados.
- Como resultado de los requisitos cambiantes, soft- El hardware a menudo se construye usando un proceso iterativo en lugar de como una secuencia de tareas cerradas.
- La ingeniería de software necesariamente incorpora- Valora la creatividad y la disciplina. Mantenimiento un equilibrio apropiado entre los dos es a veces difícil
- El grado de novedad y complejidad es a menudo alto
- A menudo hay una rápida tasa de cambio en el tecnología subyacente

Actividades de gestión de ingeniería de software. ocurrir en tres niveles: organizacional e infra- gestión de estructuras, gestión de proyectos, y gestión del programa de medición. Los dos últimos están cubiertos en detalle en este KA descripción. Sin embargo, esto no es para disminuir el importancia de la organización y la infraestructura asuntos Gerenciales. En general se acuerda que gerentes de ingeniería organizacional de software debe estar familiarizado con la gestión del proyecto conocimiento y medición de software descrito en este KA. También deberían poseer algunos conocimientos de dominio objetivo. Del mismo modo, es También es útil si los gerentes de proyectos complejos y programas en los que el software es un componente de la arquitectura del sistema es consciente de las diferencias eces que los procesos de software introducen en los proyectos Gestión de ect y medición de proyectos.

1 Los términos Iniciación, Planificación, Ejecución, La supervisión y el control, y el cierre se utilizan para describir grupos de procesos en la *Guía PMBOK ®* y *SWX* .

Page 134

Figura 7.1. Desglose de temas para la gestión de ingeniería de software KA

Otros aspectos de la gestión organizacional.
ejercer un impacto en la ingeniería de software (para
ejemplo, políticas y procedimientos organizacionales
que proporcionan el marco en el que el software
se realizan proyectos de ingeniería). Estas poli-
Es posible que sea necesario ajustar las medidas y los procedimientos
los requisitos para el desarrollo efectivo de software
ment y mantenimiento. Además, una serie de
las políticas específicas de ingeniería de software pueden
necesita estar en su lugar o establecido para que sea efectivo
gestión de ingeniería de software en la organización
nivel nacional Por ejemplo, las políticas suelen ser
necesario para establecer una organización específica en toda la organización
procesos o procedimientos para ingeniería de software
tareas como diseño de software, construcción de software
ción, estimación, monitoreo e informes. Tal
las políticas son importantes para un efectivo a largo plazo
gestión de proyectos de ingeniería de software
a través de una organización (por ejemplo, establecer
Una base consistente para analizar los proyectos pasados
rendimiento ect e implementa mejoras).

Otro aspecto importante de la organización.
gestión es políticas de gestión de personal
y procedimientos para la contratación, capacitación y mentoría
ing personal para el desarrollo profesional, no solo en
el nivel del proyecto, pero también a largo plazo
dentro una organización. Ingeniería de software per-
sonnel puede presentar entrenamiento o personal único
desafíos de gestión (por ejemplo, mantener
moneda en un contexto donde la tecnología subyacente
la nología sufre cambios rápidos y continuos).

La gestión de la comunicación también es frecuente.
mencionado como un aspecto pasado por alto pero importante
de la organización de las personas en un campo donde
comprensión precisa de las necesidades del usuario, software
requisitos y diseños de software son necesarios.
Además, la gestión de cartera, que pro-
presenta una vista general, no solo de las curvas de software
actualmente en desarrollo en varios proyectos y
programas (proyectos integrados), pero también de software
artículos planificados y actualmente en uso en una organización
ción, es deseable. Además, la reutilización del software es una clave

135

factor para mantener y mejorar la productividad
y competitividad. La reutilización efectiva requiere un
gestión: un principio básico de cualquier verdadero motor
disciplina de neering (ver Medición en inglés)

visión estratégica que refleja las ventajas y desventajas de la reutilización.

Además de comprender los aspectos de gestión que está influenciada de manera única por soft- proyectos de software, los ingenieros de software deberían tener algún conocimiento de los aspectos más generales de gestión que se discuten en este KA (incluso en los primeros años después de la graduación).

Atributos de la cultura organizacional y el comportamiento ior, más gestión de otras áreas funcionales de la empresa, tienen una influencia, aunque indi directamente, en la ingeniería de software de una organización procesos.

Amplia información sobre software la gestión de proyectos se puede encontrar en la *Guía al Cuerpo de Conocimientos de Gestión de Proyectos (Guía PMBOK ®)* y la *extensión de software para la guía PMBOK ®* (*SWX*) [1] [2]. Cada uno de estos Las guías incluyen diez KA de gestión de proyectos: gestión de integración de proyectos, alcance del proyecto gestión, gestión del tiempo del proyecto, proyecto gestión de costos, gestión de calidad de proyectos, proyecto de gestión de recursos humanos, proyecto gestión de comunicaciones, gestión de riesgos de proyectos gestión, gestión de adquisiciones de proyectos y gestión de partes interesadas del proyecto. Cada KA tiene relevancia directa para esta Ingeniería de Software Gestión KA.

También se proporciona información adicional en el otras referencias y lecturas adicionales para este KA.

Este Software Engineering Management KA consiste en el programa de gestión de proyectos de software ceses en los primeros cinco temas en la Figura 7.1 (Inicia- definición de alcance y alcance, plan de proyecto de software ning, implementación de proyectos de software, revisión y Evaluación, Cierre), más Ingeniería de Software Medición en el sexto tema y software Herramientas de gestión de ingeniería en el séptimo tema. Mientras que la gestión y la medida del proyecto la gestión del ment es a menudo considerada como separados, y de hecho cada uno posee muchos atributos únicos, la estrecha relación ha llevado a tratamiento combinado en este KA.

Desafortunadamente, una percepción común de lo blando la industria del software es que los productos de software se entregan llegó tarde, por encima del presupuesto, de mala calidad y con Funcionalidad incompleta. Medición informada Neering Foundations KA): puede ayudar a mejorar La percepción y la realidad. En esencia, el hombre agente sin medida (cualitativa y cuantitativo) sugiere una falta de disciplina, y medición sin gestión sugiere una falta de propósito o contexto. Gestión eficaz requiere una combinación de medición y experiencia.

Se adoptan las siguientes definiciones de trabajo aquí:

- La *gestión* es un sistema de procesos y controles necesarios para lograr lo estratégico objetivos establecidos por la organización.
- *Medición se* refiere a la asignación de val- ues y etiquetas para el trabajo de ingeniería de software productos, procesos y recursos más el modelos que se derivan de ellos, ya sea estos modelos se desarrollan utilizando estadísticas u otras técnicas [3 *, c7, c8].

La gestión de proyectos de ingeniería de software. secciones de este KA hacen un uso extensivo de la sección de medición de ingeniería de software.

Este KA está estrechamente relacionado con otros en el *SWEBOK Guide* , y leyendo el siguiente KA las descripciones en conjunto con este serán particularmente útil:

- La Fundación de Ingeniería KA describe algunos conceptos generales de medición que son directamente aplicables al Software Engi- sección de medición de este KA. Además, los conceptos y técnicas. presentado en la sección de Análisis Estadístico de los Fundamentos de Ingeniería KA aplican directamente a muchos temas en este KA.
- Los requisitos de software que describe KA algunas de las actividades que deberían realizarse formado durante la iniciación y alcance defi- fase inicial del proyecto.
- La gestión de la configuración del software KA se ocupa de identificación, control, estado contabilidad y auditoría de software figuraciones junto con el lanzamiento de software gestión y entrega y configuración de software herramientas de gestión de raciones.

Page 136

- El proceso de ingeniería de software KA describe los modelos de ciclo de vida del software y el relaciones entre procesos y trabajo productos
- La calidad del software KA enfatiza la calidad ity como objetivo de gestión y como objetivo de muchas actividades de ingeniería de software.
- La Ingeniería de Software Economía KA discute cómo hacer software relacionado decisiones en un contexto empresarial.

DESGLOSE DE TEMAS PARA INGENIERÍA DE SOFTWARE ADMINISTRACIÓN

Porque la mayoría del ciclo de vida de desarrollo de software- implementación de programas de medición en organizaciones de ingeniería de software;
- Herramientas de gestión de ingeniería de software, que describe la selección y uso de herramientas para gestionar un proyecto de ingeniería de software.

1. Iniciación y definición del alcance

El enfoque de estas actividades está en la disuasión efectiva minería de requisitos de software usando vari- nuestros métodos de obtención y la evaluación de viabilidad del proyecto desde una variedad de puntos de vista. Una vez que se ha establecido la viabilidad del proyecto, el las tareas restantes dentro de esta sección son las especificaciones ficación de requisitos y selección de la provisión eses de revisión y revisión de requisitos.

los modelos requieren actividades similares que pueden ser ejecutables recortado de diferentes maneras, el desglose de temas está basado en actividades. Ese desglose se muestra en Figura 7.1. Los elementos de la ruptura de nivel superior abajo se muestra en esa figura son las actividades que generalmente se realizan cuando se desarrolla un software se está gestionando un proyecto independiente de El modelo de ciclo de vida de desarrollo de software (ver Modelos de ciclo de vida del software en el software Proceso de Ingeniería KA) que ha sido elegido para Un proyecto específico. No hay intención en este descanso abajo para recomendar un modelo de ciclo de vida específico. El desglose implica solo lo que sucede y no implica cuándo, cómo o cuántas veces Cada actividad ocurre. Los siete temas son:

- Iniciación y definición del alcance, que tratan con la decisión de embarcarse en un software proyecto de ingeniería;
- Planificación de proyectos de software, que aborda las actividades emprendidas para prepararse para un éxito cessful proyecto de ingeniería de software de la perspectiva de gestión;
- Implementación de proyectos de software, que trata con ingeniería de software generalmente aceptada actividades de manejo que ocurren durante el ejecución de un proyecto de ingeniería de software;
- Revisión y evaluación, que se ocupan de asegurando que técnico, horario, costo y las actividades de ingeniería de calidad son satisfactorias;
- Cierre, que aborda las actividades. logrado para completar un proyecto;
- Medición de ingeniería de software, que se ocupa del desarrollo efectivo y

1.1. Determinación y Negociación de Requisitos

[3 *, c3]

Conjunto de requisitos de determinación y negociación los límites visibles para el conjunto de tareas son llevado a cabo (ver los requisitos de software KA). Las actividades incluyen obtención de requisitos, análisis sis, especificación y validación. Métodos y Se deben seleccionar y aplicar técnicas, tomando tener en cuenta las diversas perspectivas de las partes interesadas. Esto lleva a la determinación del alcance del proyecto en para cumplir objetivos y satisfacer limitaciones.

1.2. Análisis de viabilidad

[4 *, c4]

El propósito del análisis de factibilidad es desarrollar un Descripción clara de los objetivos y la evaluación del proyecto. tomió enfoques alternativos para determinar si el proyecto propuesto es la mejor alternativa tive dadas las limitaciones de la tecnología, los recursos, finanzas y consideraciones sociales / políticas. Un proyecto inicial y declaración del alcance del producto, proyecto entregables, restricciones de duración del proyecto y un Se debe preparar una estimación de los recursos necesarios.

Los recursos incluyen un número suficiente de personas que tienen las habilidades, instalaciones, infraestructura y soporte (ya sea internamente o externamente). El análisis de viabilidad a menudo requiere estimaciones aproximadas de esfuerzo y costo basado sobre métodos apropiados (ver sección 2.3, Esfuerzo, Horario y estimación de costos).

1.3. Proceso para la revisión y revisión de Requisitos

[3 *, c3]

Dada la inevitabilidad del cambio, las partes interesadas debe acordar los medios por los cuales los requisitos y el alcance deben ser revisados y revisados (para ejemplo, procedimientos de gestión de cambios, iteración ciclo retrospectivo). Esto implica claramente ese alcance y requisitos no serán "establecidos en piedra "pero puede y debe ser revisado en el momento predefinido puntos minados a medida que se desarrolla el proyecto (por ejemplo, en el momento en que se crean las prioridades de la cartera de pedidos en revisiones de hitos). Si se aceptan cambios, luego alguna forma de análisis de trazabilidad y riesgo el análisis debe usarse para determinar el impacto de esos cambios (ver sección 2.5, Gestión de riesgos) ment y Control de configuración de software en el Software Configuration Management KA).

Un enfoque de cambio gestionado también puede formar el base para la evaluación del éxito durante el cierre de un ciclo incremental o un proyecto completo, basado sobre los cambios que se han producido en el camino (ver tema 5, cierre).

2. Planificación de proyectos de software

El primer paso en la planificación de proyectos de software debe ser la selección de un desarrollo de software apropiado modelo de ciclo de vida y tal vez adaptarlo

2.1. Planificación de procesos

[3 *, c3, c4, c5] [5 *, c1]

Modificación del ciclo de vida del desarrollo de software (SDLC) Els abarcan un continuo de predictivo a adaptativo (consulte Modelos de ciclo de vida del software en el software Proceso de Ingeniería KA). Los SDLC predictivos son caracterizado por el desarrollo de software suave detallado requisitos de software, planificación detallada del proyecto y planificación mínima para la iteración entre el desarrollo fases de ment. Los SDLC adaptativos están diseñados para ajustar los requisitos de software emergentes y predictivo de planes. Un altamente pre SDLC adictivo ejecuta los primeros cinco procesos listado en la Figura 7.1 en una secuencia lineal con revisiones a fases anteriores solo según sea necesario. Adap- Los SDLC activos se caracterizan por un desarrollo iterativo ciclos de opciones. SDLC en el rango medio de El continuo SDLC produce incrementos de funciones cionalidad en un horario previamente planificado (en el lado predictivo del continuo) o como producto Efectos de los ciclos de desarrollo frecuentemente actualizados (en el lado adaptativo del continuo).

Los SDLC conocidos incluyen la cascada, modelos incrementales y espirales más varias formas de desarrollo de software ágil [2] [3 *, c2].

Métodos relevantes (ver el Ingeniero de Software- del Modelos y métodos KA) y las herramientas deben ser seleccionado como parte de la planificación. Herramientas automatizadas que se utilizará durante todo el proyecto también debe

basado en el alcance del proyecto, requisitos de software, y una evaluación de riesgos. Otros factores a considerar ered incluye la naturaleza del dominio de la aplicación, complejidad funcional y técnica, y soft- requisitos de calidad del software (consulte Calidad del software Requisitos en el Software Quality KA).

En todos los SDLC, la evaluación de riesgos debe ser un elemento de la planificación inicial del proyecto y el "riesgo perfil "del proyecto debe ser discutido y aceptado por todos los interesados relevantes. Software procesos de gestión de calidad (ver Software Procesos de gestión de calidad en el software La calidad KA) debe determinarse como parte de la proceso de planificación y resultado en procedimientos y responsabilidades para el aseguramiento de la calidad del software verificación y validación, revisiones y auditorías (Ver el Software Quality KA). Procesos y responsabilidades para la revisión y revisión en curso del plan del proyecto y los planes relacionados también deben estar claramente establecido y acordado.

ser planeado y adquirido. Las herramientas pueden incluir herramientas para la programación de proyectos, software requerido ments, diseño de software, construcción de software, mantenimiento de software, configuración de software gestión, proceso de ingeniería de software, software calidad de loza, y otros. Mientras que muchos de estos las herramientas deben seleccionarse principalmente en función de consideraciones técnicas discutidas en otros KAs, algunos de ellos están estrechamente relacionados con la gestión Consideraciones sobre el tema discutidas en este capítulo.

2.2. Determinar los entregables

[3 *, c4, c5, c6]

Los productos de trabajo de cada actividad del proyecto (para ejemplo, documentación de diseño de arquitectura de software ments, informes de inspección, software probado) deben ser identificado y caracterizado Oportunidades para reutilizar componentes de software de proyectos anteriores ects o para utilizar productos de software estándar

Page 138

debe ser evaluado Adquisición de software y uso de terceros para desarrollar entregables debe planificarse y seleccionarse a los proveedores (ver sección 3.2, Adquisición de software y proveedor Gestión de contratos).

2.3. Esfuerzo, cronograma y estimación de costos

[3 *, c6]

El rango estimado de esfuerzo requerido para un proyecto ect, o partes de un proyecto, se pueden determinar usando Un modelo de estimación calibrado basado en el historial datos de tamaño y esfuerzo (cuando estén disponibles) y otros métodos relevantes como el juicio de expertos y analogía. Se pueden establecer dependencias de tareas y oportunidades potenciales para completar tareas concurrente y secuencialmente se puede identificar y documentado usando un diagrama de Gantt, para examen ple. Para proyectos predictivos de SDLC, lo esperado calendario de tareas con horas de inicio proyectadas, duraciones y tiempos finales normalmente se producen durante planificación. Para proyectos SDLC adaptativos, un exceso de toda estimación de esfuerzo y cronograma es típicamente desarrollado a partir de la comprensión inicial de la requisitos o, alternativamente, restricciones en se puede especificar el esfuerzo general y el cronograma y utilizado para determinar una estimación inicial del número ber de ciclos iterativos y estimaciones de esfuerzo y otros recursos asignados a cada ciclo.

Recursos necesarios (por ejemplo, personas y herramientas) pueden traducirse en estimaciones de costos La estimación inicial de esfuerzo, cronograma y costo es una actividad iterativa que debe ser negociada y revisado entre los interesados afectados hasta que Se llega a sensus sobre los recursos y el tiempo disponible para completar el proyecto.

2.4. Asignación de recursos

[3 *, c5, c10, c11]

Se deben asignar equipos, instalaciones y personas. atiende a las tareas identificadas, incluida la asignación cación de responsabilidades para completar la variación ous elementos de un proyecto y el proyecto general. así como por cuestiones relacionadas con el personal (para examen ple, productividad de individuos y equipos, equipo dinámica y estructuras de equipo).

2.5. Gestión de riesgos

[3 *, c9] [5 *, c5]

El riesgo y la incertidumbre están relacionados pero son distintos. cepts. La incertidumbre resulta de la falta de información. ción El riesgo se caracteriza por la probabilidad de un evento que resultará en un impacto negativo más un caracterización del impacto negativo en un proyecto ect. El riesgo es a menudo el resultado de la incertidumbre. los Lo contrario del riesgo es la oportunidad, que es una característica denominado por la probabilidad de que un evento que tiene un resultado positivo puede ocurrir.

La gestión del riesgo implica la identificación del riesgo. factores y análisis de la probabilidad y potencia impacto social de cada factor de riesgo, priorización de factores de riesgo y desarrollo de mitigación de riesgos estrategias para reducir la probabilidad y minimizar el impacto negativo si un factor de riesgo se convierte en un problema. Métodos de evaluación de riesgos (por ejemplo, juicio experto, datos históricos, árboles de decisión, y simulaciones de procesos) a veces se pueden usar para identificar y evaluar los factores de riesgo.

Las condiciones de abandono del proyecto también pueden ser determinado en este punto en discusión con todos partes interesadas relevantes. Aspectos únicos de software de riesgo, como la tendencia de los ingenieros de software a agregar funciones innecesarias, o los riesgos relacionados con soft- La naturaleza intangible de las mercancías puede influir en el riesgo gestión de un proyecto de software. Atención particular se debe pagar la gestión de riesgos relacionado con los requisitos de calidad del software, como seguridad o protección (consulte la KA de calidad del software). La gestión de riesgos debe hacerse no solo en el inicio de un proyecto, pero también a intervalos periódicos vals a lo largo del ciclo de vida del proyecto.

2.6. Gestión de la calidad

[3 *, c4] [4 *, c24]

Los requisitos de calidad del software deben ser identi

Una matriz que muestra quién es responsable de responsable, consultado e informado sobre cada una de las tareas pueden ser utilizadas. Recurso la asignación se basa y está limitada por disponibilidad de recursos y su uso óptimo, como

fied, tal vez tanto en cuantitativa como cualitativa términos, para un proyecto de software y los asociados productos del trabajo. Umbrales para calidades aceptables Se deben establecer medidas de ity para cada software requisito de calidad basado en las necesidades de los interesados

y expectativas Procedimientos relacionados con Software Quality Assurance (SQA) en curso y mejora de calidad a lo largo del desarrollo proceso, y para la verificación y validación de el producto de software entregable, también debe ser especificado durante la planificación de calidad (por ejemplo, revisiones técnicas e inspecciones o demostraciones funciones de funcionalidad completada; ver el software Calidad KA).

2.7. Gestión del plan [3 *, c4]

Para proyectos de software, donde el cambio es una expectation, los planes deben ser gestionados. Administrar el El plan del proyecto debe ser planificado. Planes y procesos seleccionados para el desarrollo de software debe ser sistemáticamente monitoreado, revisado, reportado y, cuando sea apropiado, revisado. Planes asociado con procesos de apoyo (para examen- ple, documentación, configuración de software gestión y resolución de problemas) también debe ser gestionado Informes, monitoreo y control un proyecto debe caber dentro del SDLC seleccionado y las realidades del proyecto; los planes deben tener en cuenta para los diversos artefactos que se utilizarán para envejecer el proyecto.

3. Implementación de proyectos de software

Durante la promulgación de proyectos de software (también Procesos como ejecución del proyecto) se implementan planes y Se promulgan los procesos incorporados en los planes. En todo momento, debe haber un enfoque en la adhesión a los procesos SDLC seleccionados, con un expectativa primordial de que la adherencia conducirá a la satisfactoria satisfacción de los interesados requiere mentos y logro de los objetivos del proyecto Tives. Fundamental para la promulgación son los continuos actividades de manejo de monitoreo, control- ling e informes.

3.1. Implementación de Planes [4 *, c2]

Las actividades del proyecto deben llevarse a cabo de acuerdo bailar con el plan del proyecto y los planes de apoyo. Recursos (por ejemplo, personal, tecnología, y financiación) se utilizan y productos de trabajo (para ejemplo, diseño de software, código de software y se generan casos de prueba de software).

3.2. Adquisición de software y contrato de proveedor administración [3 *, c3, c4]

Adquisición de software y contrato de proveedor la gestión se refiere a cuestiones relacionadas con contratación con clientes del desarrollo de software organización de opciones que adquieren el producto entregable productos de trabajo y con proveedores que suministran productos o servicios a la ingeniería de software organización.

Esto puede implicar la selección de tipos apropiados de contratos, como precio fijo, tiempo y material als, costo más tarifa fija, o costo más tarifa de incentivo. Acuerdos con clientes y proveedores tipi Especifique con precisión el alcance del trabajo y la entrega ables e incluyen cláusulas como sanciones por retraso entrega o no entrega y propiedad intelectual acuerdos que especifican lo que el proveedor o proveedor los alicates están proporcionando y lo que el comprador está pagando ing para, además de lo que será entregado y propiedad por el adquirente. Para software desarrollado por proveedores (tanto internos como externos al soft- organización de desarrollo de mercancías), acuerdos com Indique solo los requisitos de calidad del software para aceptación del software entregado.

Después de que se haya establecido el acuerdo, Ejecución del proyecto de conformidad con los términos del acuerdo debe ser gestionado (ver capítulo 12 de SWX, Gestión de Adquisiciones de Software, para más información sobre este tema [2]).

3.3. Implementación del proceso de medición [3 *, c7]

El proceso de medición debe ser promulgado durante ing el proyecto de software para asegurar que relevante y se recopilan datos útiles (ver secciones 6.2, Planifique el proceso de medición y 6.3, realice El proceso de medición).

3.4. Monitorear proceso [3 *, c8]

Adhesión al plan del proyecto y afines los planes deben evaluarse continuamente y al

intervalos predeterminados. Además, salidas y comp
Se deben evaluar los criterios de cumplimiento para cada tarea
Los entregables deben evaluarse en términos de su
características requeridas (por ejemplo, a través de inspec-
o demostrando la funcionalidad de trabajo).
Gastos de esfuerzo, cumplimiento del cronograma y costos
hasta la fecha se debe analizar, y el uso de recursos
examinado. El perfil de riesgo del proyecto (ver sección
2.5, Gestión de riesgos) debe revisarse, y
adherencia a los requisitos de calidad del software eval-
uado (consulte Requisitos de calidad del software en
Calidad de software KA).

Los datos de medición deben analizarse (ver Sta-
Análisis estadístico en las fundaciones de ingeniería
KA). Análisis de varianza basado en la desviación de
real de los resultados esperados y los valores deben
ser determinado. Esto puede incluir sobrecostos,
deslizamiento del horario u otras medidas similares.
Identificación atípica y análisis de calidad y
Se deben realizar otros datos de medición (para
ejemplo, análisis de defectos; ver Calidad del software
Medición en la Calidad del Software KA). Riesgo
las exposiciones deben recalcularse (ver sección 2.5,
Gestión de riesgos). Estas actividades pueden permitir
detección de problemas e identificación de excepciones
basado en umbrales que se han excedido.
Los resultados deben informarse cuando los umbrales
han sido excedidos, o según sea necesario.

3.5. Proceso de control

[3 *, c7, c8]

Los resultados de las actividades de monitoreo del proyecto.
proporcionar la base sobre la cual se pueden tomar decisiones.
En su caso, y cuando la probabilidad y
impacto de los factores de riesgo se entienden, los cambios pueden
hacerse al proyecto. Esto puede tomar la forma de
acción correctiva (por ejemplo, volver a probar ciertas
componentes de software); puede implicar incorporación
calificar acciones adicionales (por ejemplo, decidir
utilizar prototipos para ayudar en la necesidad de software:
validación de ments; ver Prototipos en el software
Requisitos KA); y / o puede implicar una revisión
del plan del proyecto y otros documentos del proyecto
(por ejemplo, los requisitos de software especifi-
cación) para acomodar eventos imprevistos y
sus implicaciones

En algunos casos, el proceso de control puede
conducir al abandono del proyecto. En todos los casos,
control de configuración de software y control de software
los procedimientos de gestión de la figuración deben ser
adherido a (consulte el Manual de configuración de software-
agement KA), las decisiones deben documentarse
y comunicado a todas las partes relevantes, planes
debe revisarse y revisarse cuando sea necesario,
y datos relevantes registrados (ver sección 6.3, Per-
formar el proceso de medición).

3.6. Informes

[3 *, c11]

En momentos especificados y acordados, avance a
se debe informar la fecha, tanto dentro de la organización
nización (por ejemplo, a una dirección de proyecto de dirección
mittee) y partes interesadas externas (para examen
ple, clientes o usuarios). Los informes deben centrarse en
las necesidades de información del público objetivo como
opuesto al estado detallado que informa dentro del
equipo de proyecto.

4. Revisión y evaluación

En momentos predefinidos y según sea necesario, el programa general
Resistencia hacia el logro de los objetivos establecidos
y satisfacción de las partes interesadas (usuario y cliente)
Los requisitos deben ser evaluados. Similar,
evaluaciones de la efectividad del software
proceso, el personal involucrado y las herramientas y
los métodos empleados también deben llevarse a cabo
ularly y según lo determinen las circunstancias.

4.1. Determinación de la satisfacción de los requisitos

[4 *, c8]

Poder lograr la satisfacción de los interesados es
un objetivo principal de la ingeniería de software
gerente, el progreso hacia este objetivo debería
ser evaluado periódicamente El progreso debe ser
evaluado en el logro del proyecto principal milla-
piedras (por ejemplo, finalización de software
arquitectura de diseño o terminación de un software
revisión técnica de mercancías), o al finalizar
un ciclo de desarrollo iterativo que resulta en
Un incremento de producto. Desviaciones del software
los requisitos deben ser identificados y apropiados
Se deben tomar acciones.

Como en la actividad del proceso de control anterior (ver sección
ción 3.5, Proceso de control), configuración de software

control y gestión de configuración de software
se deben seguir los procedimientos (ver el Software
Configuration Management KA), decisiones docu-
Ment y comunicado a todas las partes relevantes,
planes revisados y revisados cuando sea necesario, y
Datos relevantes registrados (ver sección 6.3, Realizar
El proceso de medición).

4.2. Revisión y evaluación del desempeño

[3 *, c8, c10]

Revisiones periódicas del desempeño del proyecto

5.2. Actividades de cierre

[2, s3.7, s4.8]

Una vez confirmado el cierre, el archivo de
los materiales del proyecto deben realizarse en
de acuerdo con el método acordado por las partes interesadas
ods, ubicación y duración, posiblemente incluyendo
destrucción de información sensible, software,
y el medio en el que las copias son residentes.
La base de datos de medición de la organización debe
ser actualizado con datos relevantes del proyecto. Un proyecto,
análisis retrospectivo de fase o iteración debe

sonnel puede proporcionar información sobre la probabilidad de adhesión a los planes y procesos, así como posibles áreas de dificultad (por ejemplo, equipo conflictos de miembros). Los diversos métodos, herramientas y las técnicas empleadas deben evaluarse para su efectividad y adecuación, y la proceso utilizado por el proyecto también debe ser evaluado de forma sistemática y periódica evance, utilidad y eficacia en el contexto del proyecto. En su caso, se deben hacer cambios y gestionado

5. Cierre

Un proyecto completo, una fase principal de un proyecto, o un ciclo de desarrollo iterativo alcanza el cierre seguro cuando todos los planes y procesos han sido promulgada y completada. Los criterios para el proyecto, fase, o el éxito de la iteración debe ser evaluado. Una vez establecido el cierre, archivo, retrospectiva tive, y las actividades de mejora de procesos pueden ser realizado.

5.1. Determinación de cierre

[1, s3.7, s4.6]

El cierre se produce cuando las tareas especificadas para un proyecto, una fase o una iteración se han completado logro completo y satisfactorio de la competencia Se han confirmado los criterios de cumplimiento. Software los requisitos pueden confirmarse como satisfechos o no, y el grado de logro de los objetivos puede ser determinado. Los procesos de cierre deben involucrar partes interesadas relevantes y resultado en documentación de la aceptación de los interesados relevantes; cualquier conocido Los problemas deben ser documentados.

ser emprendido para que los problemas, problemas, riesgos, y las oportunidades encontradas pueden ser analizadas (ver tema 4, Revisión y evaluación). Lecciones aprendido debe ser extraído del proyecto y alimentado en aprendizaje organizacional y mejora esfuerzos.

6. Medición de ingeniería de software

La importancia de la medición y su papel en mejores prácticas de gestión e ingeniería es ampliamente reconocido (ver Medición en el Fundamentos de Ingeniería KA). Medida efectiva la garantía se ha convertido en uno de los pilares de madurez organizacional. La medida puede ser aplicado a organizaciones, proyectos, procesos y productos del trabajo. En esta sección, el foco está en aplicación de medidas a nivel de proyecto ects, procesos y productos de trabajo.

Esta sección sigue el IEEE 15939: 2008 estándar [6], que describe un proceso para definir Las actividades y tareas necesarias para implementar un proceso de medición de software. El estándar también incluye un modelo de información de medición.

6.1. Establecer y mantener la medición Compromiso

[7 *, c1, c2] [2]

- Requisitos para la medición. Cada comida esfuerzo de aseguramiento debe ser guiado por objetivos organizacionales e impulsados por un conjunto de los requisitos de medición establecidos por

[2] Tenga en cuenta que estos dos capítulos pueden ser descargado de forma gratuita desde www.psmsc.com/PSMBook.asp.

Page 142

la organización y el proyecto (para examen ple, un objetivo organizacional podría ser "Primero en el mercado con nuevos productos").
- Alcance de la medición . La organización unidad a la cual cada requerimiento de medición se aplicará debe establecerse. Esta puede consistir en un área funcional, una sola proyecto, un solo sitio o una empresa completa. El alcance temporal de la medición. el esfuerzo también debe considerarse porque series temporales de algunas mediciones pueden ser necesario; por ejemplo, para calibrar estima- modelos de sección (ver sección 2.3, Esfuerzo, Programado) Ule, y Estimación de Costos).
- Compromiso del equipo con la medición. los el compromiso debe establecerse formalmente, comunicado y apoyado por recursos (ver siguiente artículo).
- Recursos para la medición. Una organización El compromiso de la nación con la medición es un factor esencial para el éxito, como lo demuestra la asignación de recursos para implementar ing el proceso de medición. Asignación los recursos incluyen la asignación de responsabilidades ity para las diversas tareas de la medición proceso (como analista y bibliotecario). Ade- priorizado. Luego un subconjunto de objetivos a ser abordado puede ser seleccionado, documentado, comp comunicado y revisado por las partes interesadas.
- Seleccionar medidas. Las medidas del candidato deben ser seleccionado, con enlaces claros a la información Necesidades Se deben seleccionar medidas basado en las prioridades de la información necesidades y otros criterios como el costo de la col Lección, grado de interrupción del proceso durante recogida, facilidad de obtención precisa, con- datos consistentes y facilidad de análisis e informes En g. Porque las características de calidad interna (ver Modelos y características de calidad en la calidad del software KA) a menudo no se consideran contenido en el software contractualmente vinculante requisitos, es importante tener en cuenta aumentando la calidad interna del software para proporcionar un indicador temprano de posibles problemas eso puede afectar a los interesados externos.
- Definir recopilación de datos, análisis e informes. procedimientos de ing. Esto abarca la colección procedimientos y horarios, almacenamiento, verificación ción, análisis, informes y configuración gestión de datos.
- Seleccionar criterios para evaluar la información. productos Los criterios para la evaluación son influ-

Financiamiento, capacitación, herramientas y apoyo para realizar el proceso también debe ser asignado.

6.2. Planificar el proceso de medición

[7 *, c1, c2]

- Caracterizar la unidad organizativa. los unidad organizativa proporciona el contexto para medida, por lo que el contexto organizacional debe hacerse explícito, incluido el limitaciones que impone la organización El proceso de medición. La caracterización se puede establecer en términos de organización procesos, dominios de aplicación, tecnología, interfaces organizacionales y organizacionales estructura.
- Identificar las necesidades de información. Información las necesidades se basan en los objetivos, restricciones, riesgos y problemas de la organización unidad. Pueden derivarse de negocios, organizacional, regulatorio y / o producto objetivos Deben identificarse y accedido por los objetivos técnicos y comerciales tives de la unidad organizativa. Información los productos incluyen aquellos asociados con el producto que se produce, así como aquellos asociado con los procesos que se utilizan para gestionar y medir el proyecto.
- Proporcionar recursos para tareas de medición. los plan de medición debe ser revisado y aprobado por las partes interesadas apropiadas para incluir todos los procedimientos de recopilación de datos; almacenamiento, análisis y procedimientos de reporte; evaluación criterios; horarios; y responsabilidades Crite- ria para revisar estos artefactos debería tener establecido en la unidad organizativa nivel o superior y debe usarse como base para estas revisiones Tales criterios deben tomar en consideración experiencia previa, disponible capacidad de recursos y posibles interrupciones a proyectos cuando los cambios de la práctica actual Se proponen tices. La aprobación demuestra compromiso con el proceso de medición.
- Identificar los recursos que estarán disponibles para implementar lo planeado y aprobado

tareas de medida. Disponibilidad de recursos puede organizarse en casos donde los cambios son para ser probado antes del despliegue generalizado. Se debe tener en cuenta los recursos. necesario para el despliegue exitoso de nuevos procedimientos o medidas.
- Adquirir e implementar tecnologías de soporte. Esto incluye la evaluación del apoyo disponible tecnologías, selección de las más adecuadas tecnologías, adquisición de esas tecnologías, y despliegue de esas tecnologías.

6.3. Realizar el proceso de medición

[7 *, c1, c2]

- Integrar procedimientos de medición con rel procesos evasivos de software. La medida procedimientos, como la recopilación de datos, deben estar integrado en los procesos de software Ellos están midiendo. Esto puede implicar cambios ing procesos de software actuales para acomodar fecha de recopilación de datos o actividades de generación. También puede involucrar análisis de software actual procesos de software para minimizar el esfuerzo adicional y evaluación del efecto sobre los empleados para asegúrese de que los procedimientos de medición ser aceptado. Problemas de moral y otros humanos Los factores deben ser considerados. además, el los procedimientos de medición deben ser commu- dedicado a quienes proporcionan los datos. Formación y el soporte también puede necesitar ser proporcionado. El análisis de datos y los procedimientos de informe son típicamente integrado en la organización y / o procesos del proyecto de manera similar.
- Recolectar datos. Los datos deben ser recopilados, veri fied, y almacenado. La colección puede a veces ser automatizado mediante el uso de ingeniero de software ing herramientas de gestión (ver tema 7, Soft- herramientas de gestión de ingeniería de mercancías) a analizar datos y desarrollar informes. Los datos pueden Fundamentos de Ingeniería KA). Los resultados y las conclusiones generalmente se revisan, utilizando un proceso definido por la organización (que puede ser formal o informal). Proveedores de datos y los usuarios de medición deberían participar al revisar los datos para asegurarse de que estén significativo y preciso y que pueden resultar en acciones razonables.
- Comunicar resultados. Productos de información debe documentarse y comunicarse a usuarios y partes interesadas.

6.4. Evaluar medición

[7 *, c1, c2]

- Evaluar los productos de información y las medidas. proceso de aseguramiento contra la evaluación especificada criterios de acción y determinar fortalezas y debilidades de los productos de información o proceso, respectivamente. La evaluación puede ser realizado por un proceso interno o un exter- auditoría final; debe incluir comentarios de usuarios de medidas. Las lecciones aprendidas deberían ser registrado en una base de datos apropiada.
- Identificar posibles mejoras. Tal las mejoras pueden ser cambios en el formato de indicadores, cambios en las unidades medidas, o reclasificación de categorías de medida. Los costos y beneficios de la mejora potencial Las decisiones deben ser determinadas y apropiadas. Se deben informar las acciones de mejora.
- Comunicar las mejoras propuestas a la propietario del proceso de medición y parte interesada ers para revisión y aprobación. Además, falta de posibles mejoras deben ser comunitarias señalado si el análisis no logra identificar mejoras

7. Herramientas de gestión de ingeniería de software

[3 *, c5, c6, c7]

ser agregado, transformado o recodificado como parte del proceso de análisis, utilizando un título de rigor apropiado a la naturaleza de los datos y las necesidades de información. Los resultados de este análisis son típicamente indicadores como gráficos, números u otras indicaciones que será interpretado, dando como resultado conclusiones y recomendaciones para ser presentadas a partes interesadas (ver Análisis estadístico en el

Las herramientas de gestión de ingeniería de software son a menudo Se utiliza para proporcionar visibilidad y control de software procesos de gestión de ingeniería. Algunas herramientas están automatizados mientras que otros se implementan manualmente Mented. Ha habido una tendencia reciente hacia el uso de suites integradas de ingeniero de software ing herramientas que se utilizan en todo un proyecto para planificar, recopilar y registrar, monitorear y controlar, y

informe de proyecto e información del producto. Las herramientas pueden dividirse en las siguientes categorías:

Planificación de proyectos y herramientas de seguimiento. Las herramientas de planificación y seguimiento se pueden utilizar para estimar esfuerzo y costo del proyecto de mate y para preparar el proyecto horarios. Algunos proyectos usan estimaciones automatizadas. Herramientas que aceptan como entrada el tamaño estimado y otras características de un producto de software y producir estimaciones del esfuerzo total requerido, horario y costo. Las herramientas de planificación también incluyen herramientas de programación automatizadas que analizan las tareas dentro de una estructura de desglose del trabajo, su estimación duraciones apareadas, sus relaciones de precedencia, y los recursos asignados a cada tarea para produce un horario en forma de diagrama de Gantt.

Las herramientas de seguimiento se pueden utilizar para rastrear proyectos hitos, estado del proyecto programado regularmente reuniones, ciclos de iteración programados, producto demostraciones y / o elementos de acción.

Herramientas de gestión de riesgos. Gestión de riesgos las herramientas (ver sección 2.5, Gestión de riesgos *)* pueden ser utilizado para rastrear la identificación, estimación de riesgos, y monitoreando. Estas herramientas incluyen el uso de enfoques como la simulación o los árboles de decisión para analizar el efecto de los costos versus los pagos

y estimaciones subjetivas de las probabilidades de eventos de riesgo. Las herramientas de simulación de Monte Carlo pueden ser utilizado para producir distribuciones de probabilidad de riesgo combinando múltiples distribuciones de probabilidad de entrada en un algoritmo de manera.

Herramientas de comunicación. Herramientas de comunicación puede ayudar a proporcionar oportuna y consistente información a las partes interesadas relevantes involucradas en un proyecto. Estas herramientas pueden incluir cosas como el correo electrónico notificaciones y transmisiones a los miembros del equipo y partes interesadas. También incluyen comunicación ción de minutos del proyecto programado regularmente reuniones, reuniones diarias de pie, más gráficos mostrando progreso, retrasos y mantenimiento solicitar resoluciones.

Herramientas de medición. Herramientas de medición support actividades portuarias relacionadas con la medida del software programa de ment (ver tema 6, Ingeniero de Software- ing medida). Hay pocos completamente Herramientas automatizadas en esta categoría. Medición herramientas utilizadas para recopilar, analizar e informar proyectos los datos de medición pueden basarse en hojas de cálculo desarrollado por miembros del equipo u organización del proyecto empleados nacionales.

Page 145

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	09 [3 *] airley 20 F	ile 2011 erv [4 *] metro om S	03 er 20 rn tu d T [5 *] m an oeh segundo	01 [7 *] arry y col. 20 cG METRO
1. Iniciación y alcance Definición				
1.1. Determinación y Negociación de requisitos	c3			
1.2. Análisis de viabilidad		c4		
1.3. Proceso para la Revisión y Revisión de requisitos	c3			
2. Planificación de proyectos de software				
2.1. Planificación de procesos	c2, c3, c4, c5		c1	
2.2. Determinar los entregables	c4, c5, c6			
2.3. Esfuerzo, horario y costo Estimacion	c6			
2.4. Asignación de recursos	c5, c10, c11			
2.5. Gestión de riesgos	c9		c5	
2.6. Gestión de la calidad	c4	c24		
2.7. Gestión del plan	c4			
3. Implementación de proyectos de software				
3.1. Implementación de Planes		c2		
3.2. Adquisición de software y Gestión de contratos de proveedores	c3, c4			
3.3. Implementación de Proceso de medición	c7			
3.4. Monitorear proceso	c8			
3.5. Proceso de control	c7, c8			
3.6. Informes	c11			
4. Revisión y evaluación				
4.1. Determinación de la satisfacción de Requisitos				
4.2. Revisión y evaluación Actuación	c8, c10			

Page 146

	09 [3 *] airley 20 F	ile 2011 erv [4 *] metro om S	03 er 20 rn tu d T [5 *] m an	01 [7 *] arry y col. 20 cG

				segundo	METRO
5. Cierre					
5.1. Determinación de cierre					
5.2. Actividades de cierre					
6. Ingeniería de software Medición					
6.1. Establecer y sostener Compromiso de medición					c1, c2
6.2. Planifica la medición Proceso					c1, c2
6.3. Realizar la medición Proceso					c1, c2
6.4. Evaluar medición					c1, c2
7. Ingeniería de software Herramientas administrativas	c5, c6, c7				

Page 147

LECTURAS ADICIONALES

Una guía para el organismo de gestión de proyectos de Conocimiento (Guía PMBOK ®) [1].

El *PMBOK ®* *La guía* proporciona pautas para gestiona proyectos individuales y define proyecto conceptos relacionados con la gestión. También describe el ciclo de vida de la gestión de proyectos y sus relacionados procesos, así como el ciclo de vida del proyecto. Es una guía reconocida a nivel mundial para el proyecto man- profesión de agente.

Extensión de software a la guía de Proyecto Organismo de Gestión del Conocimiento (Guía de PMBOK®) [2].

SWX proporciona adaptaciones y extensiones a las prácticas genéricas de gestión de proyectos documentado en la *Guía PMBOK®* para la gestión ing proyectos de software. El aporte principal de esta extensión de la *Guía PMBOK®* es un

Referencias

[1] Project Management Institute, *una guía para el Proyecto Organismo de Gestión del Conocimiento (Guía de PMBOK (R))* , 5ª ed., Proyecto Instituto de Gestión, 2013.

[2] Instituto de Gestión de Proyectos e IEEE Sociedad de Computación, *Extensión de Software para la Guía PMBOK® Quinta Edición* , Proyecto Instituto de Gestión, 2013.

[3 *] RE Fairley, *Gestión y liderazgo Proyectos de software* , computadora Wiley-IEEE Society Press, 2009.

[4 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[5 *] B. Boehm y R. Turner, *Equilibrio de la agilidad y disciplina: una guía para perplejos* , Addison-Wesley, 2003.

Descripción de los procesos que son aplicables para
Gestión de proyectos de software de ciclo de vida adaptativo.

Adopción estándar IEEE de ISO / IEC 15939 [6].

Esta norma internacional identifica un proceso
que apoya la definición de un conjunto adecuado de medidas
para abordar necesidades específicas de información. Identifica-
fies las actividades y tareas que son necesarias para
identificar, definir, seleccionar, aplicar y
mejorar la medición dentro de un proyecto general
o estructura de medición organizacional.

J. McDonald, *Gestión del desarrollo de
Sistemas intensivos de software* , Wiley, 2010 [8].

Este libro de texto proporciona una introducción al proyecto.
gestión para comenzar software y hardware
desarrolladores de software más material avanzado único
para gestores de proyectos con experiencia. Estudios de caso
se incluyen para planificar y gestionar la verificación
ción y validación para grandes proyectos de software,
software complejo y sistemas de hardware, también
como resultados de inspección y métricas de prueba para monitorear
Tor estado del proyecto.

[6] *IEEE Std. 15939-2008 Adopción estándar de
ISO / IEC 15939: 2007 Sistemas y software
Ingeniería: proceso de medición* ,
IEEE, 2008.

[7 *] J. McGarry et al., *Software práctico
Medición: información objetiva
para tomadores de decisiones* , Addison-Wesley
Profesional, 2001.

[8] J. McDonald, *Gestión del desarrollo de
Sistemas Intensivos de Software* , John Wiley y
Sons, Inc., 2010.

CAPÍTULO 8

PROCESO DE INGENIERÍA DE SOFTWARE

SIGLAS

BPMN	Modelado de Procesos de Negocio Notación
CASO	Software asistido por computadora Ingenieria
CM	Gestión de la configuración
CMMI	Modelo de Capacidad de Madurez Integración
GQM	Meta-Pregunta-Métrica
IDEF0	Definición de integración
LOE	Nivel de esfuerzo
ODC	Clasificación de defectos ortogonales
SDLC	Ciclo de vida del desarrollo de programas
SPLC	Ciclo de vida del producto de software
UML	Lenguaje de modelado unificado

INTRODUCCIÓN

Un proceso de ingeniería consiste en un conjunto de
actividades relacionadas que transforman una o más entradas
en productos mientras se consumen recursos para acomodar
pulir la transformación. Muchos de los procesos de
disciplinas de ingeniería tradicionales (p. ej., electricidad,
proceso "se denominará" proceso de software "
en este KA Además, tenga en cuenta que "software
proceso "denota actividades de trabajo, no la ejecución
proceso de software implementado.

Los procesos de software se especifican para un número
de razones: para facilitar la comprensión humana,
comunicación y coordinación; para ayudar al hombre
gestión de proyectos de software; medir y
mejorar la calidad de los productos de software en un
manera eficiente; para apoyar el proceso de mejora
ment y para proporcionar una base para el apoyo automatizado
puerto de ejecución del proceso.

SWEBOK KA estrechamente relacionado con este Soft-
Ware Engineering Process KA incluye software
Gerencia de Ingeniería, Ingeniero de Software
ing Modelos y métodos, y calidad de software;
el tema de la medición y el análisis de la causa raíz
encontrado en las Fundaciones de Ingeniería KA es también
estrechamente relacionada. Gestión de Ingeniería de Software
Mente se refiere a la adaptación, adaptación y
implementando procesos de software para un específico
proyecto de software (consulte Planificación de procesos en el
Gerencia de Ingeniería de Software KA). Mod-
Els y los métodos apoyan un enfoque sistemático para
Desarrollo y modificación de software.

La calidad del software KA se preocupa por
los procesos de planificación, aseguramiento y control

mecánica, civil, química) se preocupan por transformando la energía y las entidades físicas de una forma en otra, como en una presa hidroeléctrica que transforma la energía potencial en eléctrica energía o una refinería de petróleo que usa químicos procesos para transformar petróleo crudo en gasolina.

En esta área de conocimiento (KA), el ingeniero de software Los procesos están relacionados con las actividades laborales. realizado por ingenieros de software para desarrollar, mantener y operar software, como requerir mentos, diseño, construcción, pruebas, configuraciones gestión de operaciones y otra ingeniería de software procesos. Para facilitar la lectura, "ingeniería de software para proyecto y calidad de producto. Medido y resultados de medición en la Fundación de Ingeniería Las KA son esenciales para evaluar y controlar procesos de software ling.

DESGLOSE DE TEMAS PARA PROCESO DE INGENIERÍA DE SOFTWARE

Como se ilustra en la Figura 8.1, este KA está relacionado con definición de proceso de software, vida de software ciclos, evaluación de procesos de software y mejora- ment, medición de software e ingeniería de software herramientas de proceso de neering.

Figura 8.1. Desglose de temas para el proceso de ingeniería de software KA

1. Definición del proceso de software

[1 *, p177] [2 *, p295] [3 *, p28–29, p36, c5]

Este tema se refiere a una definición de soft- proceso de software, gestión de procesos de software y Infraestructura de proceso de software.

Como se indicó anteriormente, un proceso de software es un conjunto de actividades y tareas interrelacionadas que transforman productos de trabajo de entrada en productos de trabajo de salida. Como mínimo, la descripción de un programa de software cess incluye entradas requeridas, trabajo transformador actividades y resultados generados. Como se ilustra en Figura 8.2, un proceso de software también puede incluir sus criterios de entrada y salida y descomposición de las actividades laborales en tareas, que son las unidades de trabajo más pequeñas sujetas a gestión responsabilidad. Una entrada de proceso puede ser un disparador ing evento o la salida de otro proceso. Entrada los criterios deben cumplirse antes de que un proceso pueda comenzar. Todas las condiciones especificadas deben ser satisfecho antes de que un proceso pueda ser exitoso concluido, incluidos los criterios de aceptación para el producto de trabajo de salida o productos de trabajo.

Un proceso de software puede incluir subprocesos. Por ejemplo, la validación de requisitos de software es Un proceso utilizado para determinar si los elementos proporcionarán una base adecuada para el software desarrollo o un subproceso del software proceso de requisitos Entradas para requisitos val- idación suele ser una especificación de requisitos de software. ification y los recursos necesarios para realizar la validación dación (personal, herramientas de validación, tiempo suficiente). Las tareas de la actividad de validación de requisitos. podría incluir revisiones de requisitos, creación de prototipos, y validación del modelo. Estas tareas implican trabajo asignaciones para individuos y equipos. La salida de validación de requisitos es típicamente un validado especificación de requisitos de software que proporciona entradas al diseño de software y prueba de software procesos de ing. Validación de requisitos y otros subprocesos del proceso de requisitos de software a menudo se entrelazan e iteran de varias maneras;

Page 150

Figura 8.2. Elementos de un proceso de software

el proceso de requisitos de software y su subproceso las entradas se pueden ingresar y salir varias veces durante el desarrollo o modificación de software.

La definición completa de un proceso de software puede también incluyen los roles y competencias, el soporte de TI puerto, técnicas y herramientas de ingeniería de software, y entorno de trabajo necesario para realizar el proceso, así como los enfoques y medidas (Indicadores clave de rendimiento) utilizados para determinar la eficiencia y efectividad de realizar el proceso.

Además, un proceso de software puede incluir entrelazado técnico, colaborativo y administrativo actividades trative.

Anotaciones para definir procesos de software incluir listas textuales de actividades constituyentes y tareas descritas en lenguaje natural; flujo de datos diagramas; cartas estatales; BPMN; IDEF0; Redes de Petri; y diagramas de actividad UML. El transformador Las tareas dentro de un proceso pueden definirse como dures un procedimiento puede especificarse como un pedido conjunto de pasos o, como alternativa, como una lista de verificación trabajo a realizar en la realización de una tarea.

Debe enfatizarse que no existe el mejor software blando. proceso de software o conjunto de procesos de software. Suaire- los procesos de software deben seleccionarse, adaptarse y aplicado según corresponda para cada proyecto y cada contexto organizacional. Ningún proceso ideal, o conjunto de procesos, existe.

1.1. Gestión de procesos de software

[3 *, s26.1] [4 *, p453–454]

Dos objetivos de la gestión de procesos de software. son para darse cuenta de la eficiencia y efectividad que resultado de un enfoque sistemático para lograr ing procesos de software y producción de productos de trabajo ucts, ya sea en el individuo, proyecto u organización nivel nacional, y para introducir nuevos o mejorados procesos.

Los procesos cambian con la expectativa de que un proceso nuevo o modificado mejorará la eficiencia ciencia y / o efectividad del proceso y la calidad de los productos de trabajo resultantes. Cambiando a un nuevo proceso, mejorando un proceso existente, cambio organizacional y cambio de infraestructura (inserción de tecnología o cambios en las herramientas) son estrechamente relacionados, ya que todos suelen iniciarse con el objetivo de mejorar el costo, programa de desarrollo ule, o calidad de los productos de software. Proceso el cambio tiene un impacto no solo para el software producto; a menudo conducen a cambios organizacionales. Cambiar un proceso o introducir un nuevo proceso puede tener efectos de onda en toda una organización ción Por ejemplo, cambios en la infraestructura de TI Las herramientas y la tecnología a menudo requieren un proceso ification de

Los procesos existentes pueden modificarse cuando otros procesos nuevos se implementan para el primero iempo (por ejemplo, introducir una inspección actividad dentro de un proyecto de desarrollo de software probablemente impactará el proceso de prueba de software: ver revisiones y auditorías en la calidad del software KA y en el Software Testing KA). Estas situaciones Las acciones también pueden denominarse "evolución del proceso". Si las modificaciones son extensas, entonces los cambios en la cultura organizacional y modelo de negocio probablemente será necesario para acomodar el pro- cess cambios.

Page 151

1.2. Infraestructura de proceso de software

[2 *, p183, p186] [4 *, p437–438]

Establecimiento, implementación y gestión de software. proceso de adaptación y consideraciones prácticas.

Un ciclo de vida de desarrollo de software (SDLC) incluye los procesos de software utilizados para especificar y transformar los requisitos de software en una entrega

procesos de software y modelos de ciclo de vida del software a menudo ocurre a nivel de software individual proyectos Sin embargo, la aplicación sistemática de procesos de software y mod de ciclo de vida de software Els a través de una organización puede proporcionar beneficios a todo el trabajo de software dentro de la organización, aunque requiere compromiso en la organización nivel nacional Una infraestructura de proceso de software puede proporcionar definiciones de procesos, políticas para apli- Preparando y aplicando los procesos, y descrip- de los procedimientos que se utilizarán para implementar los procesos. Además, un proceso de software la infraestructura puede proporcionar financiación, herramientas y apoyo- ing, y miembros del personal que han sido asignados responsabilidades para establecer y mantener La infraestructura del proceso de software.

La infraestructura del proceso de software varía, depende- ing en el tamaño y la complejidad de la organización y los proyectos emprendidos dentro de la organización ción Organizaciones y proyectos pequeños y simples. tener necesidades de infraestructura pequeñas y simples. Grande organizaciones y proyectos complejos, según sea necesario sity, tenga un software más grande y complejo Infraestructuras de procesos. En este último caso, varios Se pueden establecer unidades organizativas (como un grupo de proceso de ingeniería de software o una dirección- ing comité) para supervisar la implementación y mejora de los procesos de software.

Una percepción errónea común es que establecer un infraestructura de procesos de software e implementación los procesos de software repetibles agregarán tiempo y costo para el desarrollo y mantenimiento de software. Hay un costo asociado con la introducción o mejorar un proceso de software; sin embargo, experi- ence ha demostrado que la implementación sistemática la mejora de los procesos de software tiende a resultar en menor costo a través de una mayor eficiencia, evite Un poco de retrabajo y más confiable y asequible. software. El rendimiento del proceso influye así Calidad del producto de software.

2. Ciclos de vida del software

[1 *, c2] [2 *, p190]

Este tema aborda categorías de programas de software ceses, modelos de ciclo de vida del software, software ciclo (SPLC) incluye el desarrollo de software producto de software hasta- ciclo de vida más procesos de software adicionales que proporcionar implementación, mantenimiento, soporte, evolución, jubilación y todos los demás inicios procesos de retiro para un producto de software, incluyendo la gestión de configuración de software y procesos de aseguramiento de la calidad del software que son aplicado a lo largo de un ciclo de vida del producto de software. El ciclo de vida de un producto de software puede incluir múltiples ple ciclos de vida de desarrollo de software para evolucionar y mejorando el software.

Los procesos de software individuales no tienen tempo- ral ordenando entre ellos. La relación temporal los envíos entre procesos de software son proporcionados por un modelo de ciclo de vida del software: ya sea un SDLC o SPLC. Los modelos de ciclo de vida suelen enfatizar los procesos clave del software dentro del modelo y su interdependencia temporal y lógica Cies y relaciones. Definiciones detalladas de los procesos de software en un modelo de ciclo de vida pueden ser proporcionado directamente o por referencia a otro documentos.

Además de transmitir lo temporal y relaciones lógicas entre procesos de software, el modelo de ciclo de vida de desarrollo de software (o modelos utilizados dentro de una organización) incluye el mecanismos de control para aplicar entrada y salida criterios (p. ej., revisiones de proyectos, aprobación del cliente als, pruebas de software, umbrales de calidad, dem- Ubicaciones, consenso del equipo). La salida de uno el proceso de software a menudo proporciona la entrada para otros ers (p. ej., los requisitos de software proporcionan información para un proceso de diseño arquitectónico de software y el construcción de software y pro de pruebas de software cesses). Ejecución concurrente de varios programas. las actividades del proceso pueden producir una salida compartida (por ejemplo, las especificaciones de interfaz para interfaces entre múltiples componentes de software desarrollados por diferentes equipos). Algunos procesos de software puede considerarse menos efectivo a menos que otro los procesos de software se realizan en el mismo tiempo (por ejemplo, planificación de pruebas de software durante El análisis de requisitos de software puede mejorar el Requisitos de Software).

Página 152

2.1. Categorías de procesos de software

[1 *, Prefacio] [2 *, p294–295] [3 *, c22 – c24]

Muchos procesos de software distintos han sido definido para su uso en las diversas partes del soft- vida de desarrollo de software y mantenimiento de software ciclos Estos procesos pueden clasificarse como sigue:

1. *Los procesos primarios* incluyen pro- software ceses para el desarrollo, operación y mantenimiento de software.
2. *Los procesos de apoyo* se aplican intermitentemente de manera tensa o continua a lo largo de un software ciclo de vida del producto para respaldar pro- ceses incluyen procesos de software tales como gestión de configuración, garantía de calidad ance, y verificación y validación.
3. Los *procesos organizacionales* proporcionan apoyo

2.2. Modelos de ciclo de vida del software

[1 *, c2] [2 *, s3.2] [3 *, s2.1] [5]

La naturaleza intangible y maleable del software. permite una amplia variedad de desarrollo de software modelos de ciclo de vida, que van desde modelos lineales en cuáles son las fases del desarrollo de software logrado secuencialmente con retroalimentación y iteración según sea necesario seguido de integración, prueba- ing, y entrega de un solo producto; a iterativo modelos en los que el software se desarrolla en incrementos medios de aumentar la funcionalidad en iterativo ciclos; a modelos ágiles que típicamente involucran demostraciones frecuentes de software de trabajo para un cliente o representante de usuario que dirige desarrollo del software en iteración corta ciclos que producen pequeños incrementos de trabajo, Software entregable. Incremental, iterativo y los modelos ágiles pueden entregar los primeros subconjuntos de trabajo

puerto para ingeniería de software; Incluyen capacitación, análisis de medición de procesos, gestión de infraestructura, cartera y gestión de reutilización, proceso organizacional mejora y gestión de software modelos de ciclo de vida.

4. *Procesos de proyectos cruzados,* como reutilización, software línea de productos de cerámica e ingeniería de dominio, implican más de un solo software proyecto en una organización.

Procesos de software además de los enumerados arriba incluye lo siguiente.

Los procesos de gestión de proyectos incluyen procesos para planificar y estimar, recursos gestión, medición y control, liderazgo, gestión de riesgos, gestión de partes interesadas y coordinación cenando en lo primario, de apoyo, organizacional, y procesos de desarrollo de software entre proyectos proyectos de mantenimiento y mantenimiento.

Los procesos de software también se desarrollan para necesidades particulares, como actividades de proceso que abordar las características de calidad del software (ver la calidad del software KA). Por ejemplo, secu- Las preocupaciones de la ciudad durante el desarrollo de software necesita uno o más procesos de software para proteger la seguridad del entorno de desarrollo ment y reducir el riesgo de actos maliciosos. Suave- los procesos de software también se pueden desarrollar para bases adecuadas para establecer confianza en La integridad del software.

software en el entorno del usuario, si lo desea.

Los modelos SDLC lineales a veces se refieren como ciclo de vida de desarrollo de software predictivo modelos, mientras que los SDLC iterativos y ágiles son denominado desarrollo de software adaptativo modelos de ciclo de vida. Cabe señalar que vari- Ous actividades de mantenimiento durante un SPLC puede se llevará a cabo utilizando diferentes modelos SDLC, como apropiado para las actividades de mantenimiento.

Una característica distintiva de los diversos soft- modelos de ciclo de vida de desarrollo de mercancías es el camino en qué requisitos de software se gestionan. Lin- los modelos de desarrollo del oído suelen desarrollar un conjunto completo de requisitos de software, en la medida posible, durante la iniciación y planificación del proyecto. Los requisitos de software son rigurosamente ón-isado. Cambios en los requisitos de software. se basan en solicitudes de cambio que se procesan por un tablero de control de cambios (ver Solicitud, Evaluación y aprobación de cambios de software en la placa de control de cambios en el software Gestión de la figuración KA). Un incremental modelo produce incrementos sucesivos de trabajo ing, software entregable basado en particionamiento de los requisitos de software a implementar en cada uno de los incrementos. El software requiere Los controles pueden ser rigurosamente controlados, como en un caso lineal. modelo, o puede haber cierta flexibilidad en la revisión proporcionar de software como producto de software evoluciona Los modelos ágiles pueden definir el alcance del producto y características de alto nivel inicialmente; sin embargo, ágil

Page 153

Los modelos están diseñados para facilitar la evolución de la requisitos de software durante el proyecto.

Debe enfatizarse que el continuo de Los SDLC de lineal a ágil no son delgados, rectos línea. Elementos de diferentes enfoques pueden ser incorporado en un modelo específico; para examen- ple, una vida de desarrollo de software incremental modelo de ciclo puede incorporar secuencial soft- requisitos de hardware y fases de diseño pero permiten considerable flexibilidad en la revisión del software requisitos y arquitectura durante el software construcción.

2.3. Adaptación de procesos de software

[1 *, s2.7] [2 *, p51]

SDLC predefinidos, SPLC y software individual los procesos de software a menudo necesitan ser adaptados (o "A medida") para atender mejor las necesidades locales. Organizativo contexto nacional, innovaciones tecnológicas, proyecto tamaño, criticidad del producto, requisitos reglamentarios, las prácticas de la industria y la cultura corporativa pueden Determinar las adaptaciones necesarias. Adaptación de procesos de software individuales y vida de software modelos de ciclo (desarrollo y producto) pueden consiste en agregar más detalles al programa de software ceses, actividades, tareas y procedimientos para abordar preocupaciones críticas Puede consistir en usar una alternativa Nate conjunto de actividades que logra el propósito y resultados del proceso de software. Adaptación también puede incluir omitir procesos de software o actividades de un desarrollo o vida del producto modelo de ciclo que son claramente inaplicables al construcción y pruebas) pueden adaptarse para facilitar tate operación, soporte, mantenimiento, migración, y retiro del software.

Factores adicionales a considerar cuando Definir y adaptar un modelo de ciclo de vida del software incluir la conformidad requerida con los estándares, direc- tives y políticas; demandas del cliente; criticidad del producto de software; y organización organizacional ridad y competencias. Otros factores incluyen el naturaleza del trabajo (p. ej., modificación de la existencia) ing software versus nuevo desarrollo) y el dominio de aplicación (p. ej., aeroespacial versus hotel administración).

3. Evaluación de procesos de software y Mejora

[2 *, p188, p194] [3 *, c26] [4 *, p397, c15]

Este tema aborda la evaluación del proceso de software modelos de ment, método de evaluación de procesos de software ods, modelos de mejora de procesos de software, y clasificaciones de proceso continuo y por etapas. Software las evaluaciones de proceso se utilizan para evaluar el formulario y contenido de un proceso de software, que puede ser especificado por un conjunto estandarizado de criterios. En en algunos casos, los términos "evaluación del proceso" y "evaluación de capacidad" se utilizan en lugar de evaluación de procesos. Las evaluaciones de capacidad son típicamente realizado por un adquiriente (o potencial adquirente) o por un agente externo en nombre de un adquirente (o adquirente potencial). Los resultados se usan como un indicador de si el software procesos utilizados por un proveedor (o posible apoyo

alcance del trabajo a realizar.

2.4. Consideraciones prácticas

[2 *, p188-190]

En la práctica, los procesos y actividades de software son a menudo intercalados, superpuestos y aplicados concurrentemente. Rently. Modelos de ciclo de vida de software que especifican crete procesos de software, con rigurosamente especificados criterios de entrada y salida y límites prescritos e interfaces, deben reconocerse como idealizaciones opciones que deben adaptarse para reflejar las realidades de desarrollo y mantenimiento de software dentro del contexto organizacional y ambiente de negocios.

Otra consideración práctica: software procesos (como la gestión de la configuración, alicates) son aceptables para el adquirente. Actuación las evaluaciones generalmente se realizan dentro de una organización nización para identificar procesos de software que necesitan mejora o para determinar si un proceso (o procesos) satisface los criterios en un nivel dado de capacidad de proceso o madurez. Las evaluaciones del proceso se realizan a nivel ls de organizaciones enteras, unidades organizativas dentro de organizaciones y proyectos individuales. La evaluación puede involucrar problemas tales como la evaluación si el proceso de software entra y sale se están cumpliendo los términos para revisar los factores de riesgo y gestión de riesgos, o para identificar las lecciones aprendidas. La evaluación del proceso se lleva a cabo utilizando tanto un modelo de evaluación y un método de evaluación. los modelo puede proporcionar una norma para una evaluación comparativa

Page 154

comparación entre proyectos dentro de una organización ción y entre organizaciones.

Una auditoría de proceso difiere de una evaluación de proceso ment. Se realizan evaluaciones para determinar niveles de capacidad o madurez e identificar Procesos de software a mejorar. Las auditorias son típicamente realizado para determinar el cumplimiento de Políticas y normas. Las auditorías proporcionan gestión visibilidad del ment en las operaciones reales siendo realizado en la organización para que precisa y las decisiones significativas pueden tomarse en consideración problemas que están impactando un proyecto de desarrollo ect, una actividad de mantenimiento o un software relacionado tema.

Factores de éxito para la evaluación de procesos de software ment y mejora dentro del ingeniero de software Las organizaciones incluyen patrocinio administrativo nave, planificación, entrenamiento, experimentado y capaz líderes, compromiso del equipo, gestión de expectativas ment, el uso de agentes de cambio, más proyectos piloto y experimentación con herramientas. Fac- cion adicional Los tors incluyen la independencia del asesor y el puntualidad de la evaluación.

3.1. Modelos de evaluación de procesos de software

[2 *, s4.5, s4.6] [3 *, s26.5] [4 *, p44–48]

Modelos de evaluación de procesos de software típicamente incluir criterios de evaluación para procesos de software que se consideran como buenas prácticas Estas prácticas pueden abordar el desarrollo de software solo procesos de ment, o también pueden incluir temas como mantenimiento de software, software gestión de proyectos, ingeniería de sistemas o administración de recursos humanos.

3.2. Métodos de evaluación de procesos de software

[1 *, p322–331] [3 *, s26.3] [4 *, p44-48, s16.4] [6]

Un método de evaluación de proceso de software puede ser cualitativa o cuantitativa. Evaluación cualitativa los recursos se basan en el juicio de expertos; cuanti las evaluaciones tativas asignan puntajes numéricos a procesos de software basados en análisis de objetivos evidencia que indica el logro de los objetivos y resultados de un proceso de software definido. por examinar los pasos procesales seguidos y resultados obtenidos más datos sobre defectos encontrado y tiempo requerido para encontrar y corregir los defectos en comparación con las pruebas de software.

Un método típico de evaluación de procesos de software ment incluye planificación, búsqueda de hechos (por recopilación Evidencia a través de cuestionarios, entrevistas, y observación de prácticas laborales), colección y validación de datos de proceso, y análisis y informes Las evaluaciones de procesos pueden depender de juicio subjetivo y cualitativo del evaluador, o en la presencia objetiva o ausencia de definido defectos, registros y otras pruebas.

Las actividades realizadas durante un programa de software evaluación de cess y la distribución del esfuerzo para las actividades de evaluación son diferentes dependiendo de El propósito de la evaluación del proceso del software. Se pueden realizar evaluaciones del proceso del software para desarrollar clasificaciones de capacidad utilizadas para hacer recomendaciones menciones para mejoras de procesos o pueden ser emprendido para obtener una calificación de madurez del proceso en Para calificar para un contrato o adjudicación.

La calidad de los resultados de la evaluación depende de el método de evaluación del proceso del software, el integridad y calidad de los datos obtenidos, el capacidad y objetividad del equipo de evaluación, y La evidencia examinada durante la evaluación. El objetivo de una evaluación del proceso de software es obtener información que establezca el estado actual de un proceso o procesos y proporcionar una base para la mejora de procesos; realizar un software evaluación del proceso siguiendo una lista de verificación para conformidad sin obtener información agrega poco valor.

3.3. Modelos de mejora de procesos de software

[2 *, p187–188] [3 *, s26.5] [4 *, s2.7]

Los modelos de mejora de procesos de software enfatizan dimensionar ciclos iterativos de mejora continua. Un ciclo de mejora de procesos de software típicamente implica los subprocesos de medición, ana- Lyzing y cambiante. El Plan-Do-Check-Act el modelo es un enfoque iterativo bien conocido para mejora de procesos de software. Mejora las actividades incluyen identificar y priorizar mejoras deseadas (planificación); introduciendo

ejemplo, una evaluación cuantitativa de la soft- una mejora, incluida la gestión del cambio
proceso de inspección de mercancías puede ser realizado por y entrenamiento (hacer); evaluando la mejora

en comparación con el proceso anterior o ejemplar visibilidad en productos de trabajo intermedio y
resultados y costos (verificación); y haciendo más puede ejercer cierto control sobre las transiciones entre
modificaciones (actuando). El Plan-Do-Check-Act procesos. En el nivel 3, un solo proceso de software o
Se puede aplicar el modelo de mejora de procesos, para los procesos en un grupo de nivel de madurez 3 más el
ejemplo, para mejorar los procesos de software que proceso o procesos en madurez nivel 2 están bien
Mejorar la prevención de defectos. definido (quizás en políticas organizacionales y
procedimientos) y se repiten a través de diferentes
3.4. Proceso de software continuo y por etapas Proyectos diferentes. Nivel 3 de capacidad de proceso o
Calificaciones la madurez proporciona la base para mejorar el proceso
[1 *, p28–34] [3 *, s26.5] [4 *, p39–45] ment a través de una organización porque el proceso
se realiza (o se procesan) de manera similar
Capacidad de proceso de software y proceso de software ner. Esto permite la recopilación de datos de rendimiento.
la madurez generalmente se clasifica utilizando cinco o seis de manera uniforme en múltiples proyectos. A
para caracterizar la capacidad o madurez de la nivel de madurez 4, las medidas cuantitativas pueden ser
procesos de software utilizados dentro de una organización. aplicado y utilizado para la evaluación de procesos; statis-
Un sistema de calificación *continua* implica asignar Se puede utilizar el análisis tico. En el nivel de madurez 5,
otorgar una calificación a cada proceso de software de interés. los mecanismos para el proceso continuo mejoran
se establece un sistema de calificación por *etapas* Ments se aplican.
La misma calificación de madurez para todo el software Las representaciones continuas y por etapas pueden ser
procesos dentro de un nivel de proceso especificado. Un representante utilizado para determinar el orden en que el software
representación del proceso continuo y escalonado Los procesos deben ser mejorados . En el continuo
Els se proporciona en la Tabla 8.1. Modelos continuos representación, los diferentes niveles de capacidad para
normalmente usa una calificación de nivel 0; modelos por etapas typi- diferentes procesos de software proporcionan una guía
cally no. para determinar el orden en que el software pro-
se mejorarán los ceses. En la representación escenificada
ión, satisfaciendo los objetivos de un conjunto de programas de software
Tabla 8.1. Niveles de calificación del proceso de software las cesiones dentro de un nivel de madurez se logran para
ese nivel de madurez, que proporciona una base
Nivel	Continuo Representación de capacidad Niveles	Escenificado Representación de madurez Niveles	para mejorar todos los procesos de software en el siguiente nivel superior.
0 0	Incompleto		
1	Realizado	Inicial	**4. Medición de software**
2	Gestionado	Gestionado	[3 *, s26.2] [4 *, s18.1.1]
3	Definido	Definido	
4 4		Cuantitativamente Gestionado	Este tema aborda el proceso y los productos de software. medición de uct, calidad de los resultados de medición, modelos de información de software y pro- software de software técnicas de medición de cess (ver Medición en las Fundaciones de Ingeniería KA).
5 5		Optimizando	

En la Tabla 8.1, el nivel 0 indica que un software Antes de implementar un nuevo proceso o un plan
el proceso se realiza de forma incompleta o puede no ser se modifica el proceso de alquiler, resultados de medición para
realizado. En el nivel 1, se está procesando un proceso de software en la situación actual debe obtenerse para
realizado (calificación de capacidad), o el software tiva una línea de base para la comparación entre la corriente
los procesos en un grupo de nivel de madurez 1 están siendo Situación de alquiler y la nueva situación. Para examen-
realizado pero sobre una base ad hoc, informal. A ple, antes de introducir la inspección de software
nivel 2, un proceso de software (calificación de capacidad) o proceso, esfuerzo requerido para reparar defectos descubiertos
los procesos en el nivel de madurez 2 se están realizando mediante la prueba debe medirse. Después de un ini-
formado de una manera que proporciona gestión período de inicio de prueba después del proceso de inspección
se introduce, el esfuerzo combinado de inspección

Además, las pruebas se pueden comparar con las anteriores documentación de diseño y otros trabajos relacionados

cantidad de esfuerzo requerido para probar solo. Simi
Se aplican consideraciones importantes si se cambia un proceso

4.1. Proceso de software y medición de productos
[1 *, s6.3, p273] [3 *, s26.2, p638]

El proceso del software y la medición del producto son
preocupado por determinar la eficiencia y
efectividad de un proceso de software, actividad o
tarea. La *eficiencia* de un proceso de software, actividad,
o tarea es la proporción de recursos realmente consumidos
a los recursos esperados o deseados para ser consumidos
en la realización de un proceso de software, actividad o
tarea (ver Eficiencia en la Ingeniería del Software
Economía KA). El esfuerzo (o costo equivalente) es el
medida primaria de recursos para la mayoría del software
procesos, actividades y tareas; se mide en
unidades tales como horas-persona, días-persona, personal-
semanas o meses de esfuerzo del personal o equivalentes
unidades monetarias, como euros o dólares.

La efectividad es la relación de salida real a
salida esperada producida por un proceso de software,
actividad o tarea; por ejemplo, número real de
defectos detectados y corregidos durante el software
prueba de la cantidad esperada de defectos a ser
detectado y corregido, tal vez basado en his-
datos tóricos para proyectos similares (ver Efectividad
en el Software Engineering Economics KA).
Tenga en cuenta que la medición del efecto del proceso de software
la actividad requiere la medición de lo relevante
atributos del producto; por ejemplo, medición de
defectos de software descubiertos y corregidos durante
pruebas de software.

Hay que tener cuidado al medir el producto
atributos con el fin de determinar el proceso
eficacia. Por ejemplo, la cantidad de defectos
detectado y corregido por la prueba puede no lograr
el número esperado de defectos y así proporcionar
una medida engañosamente baja de efectividad, ya sea
porque el software que se está probando es mejor
calidad de lo habitual o tal vez porque introduc-
ción de una inspección aguas arriba recientemente introducida
proceso ha reducido el número restante de
Defectos en el software.

Medidas del producto que pueden ser importantes en
determinar la efectividad del software pro
las cesiones incluyen la complejidad del producto, defectos totales
densidad de defectos y la calidad de los requisitos,
productos
También tenga en cuenta que la eficiencia y la efectividad son
conceptos independientes Un programa de software efectivo
El proceso puede ser ineficaz para lograr la suavidad deseada
resultado del proceso de software; por ejemplo, la cantidad de
esfuerzo realizado para encontrar y reparar defectos de software
podría ser muy alto y resultar en baja eficiencia, ya que
en comparación con las expectativas

Un proceso eficiente puede ser ineficaz en el alojamiento
alisar la transformación deseada del trabajo de entrada
productos en productos de trabajo de salida; por ejemplo,
no encontrar y corregir un número suficiente de
defectos de software durante el proceso de prueba.

Causas de baja eficiencia y / o baja efectividad
ness en la forma en que un proceso de software, actividad o
la tarea ejecutada puede incluir uno o más de los
siguientes problemas: producto de trabajo de entrada deficiente
ucts, personal sin experiencia, falta de adecuada
herramientas e infraestructura, aprendiendo un nuevo proceso,
un producto complejo o un producto desconocido
dominio. La eficiencia y efectividad de soft-
la ejecución del proceso de hardware también se ve afectada (ya sea
positiva o negativamente) por factores como el giro
en el personal de software, cambios de horario, un
nuevo representante del cliente o una nueva organización
política nacional

En ingeniería de software, productividad en per-
formar un proceso, actividad o tarea es la proporción de
producción producida dividida por los recursos consumidos;
por ejemplo, la cantidad de defectos de software
cubierto y corregido dividido por persona-horas de
esfuerzo (ver Productividad en el Ingeniero de Software-
ing Economics KA). Medición precisa de
la productividad debe incluir el esfuerzo total utilizado para satisfacer
Indique los criterios de salida de un proceso de software, activ-
ity, o tarea; por ejemplo, el esfuerzo requerido para
defectos correctos descubiertos durante la prueba de software
ing debe estar incluido en el desarrollo de software
productividad.

El cálculo de la productividad debe tener en cuenta
El contexto en el que se realiza el trabajo.
Por ejemplo, el esfuerzo para corregir descubrió
los defectos se incluirán en el cálculo de productividad
formulación de un equipo de software si los miembros del equipo
corregir los defectos que encuentran, como en las pruebas unitarias
por desarrolladores de software o en una función cruzada
equipo ágil O el cálculo de la productividad
puede incluir el esfuerzo del software

desarrolladores o el esfuerzo de una prueba independiente
ing equipo, dependiendo de quién arregla los defectos
encontrado por los probadores independientes. Tenga en cuenta que esto
ejemplo se refiere al esfuerzo de los equipos de desarrollo
operadores o equipos de evaluadores y no a individuos.
Productividad del software calculada a nivel de
los individuos pueden ser engañosos debido a la
Muchos factores que pueden afectar el pro-
ductividad de los ingenieros de software.

Definiciones estandarizadas y reglas de conteo
para la medición de procesos y trabajos de software
los productos son necesarios para proporcionar estandarizados
resultados de medición en proyectos dentro de un
organización, para poblar un repositorio de histori
datos de calibración que pueden analizarse para identificar software

4.3. Modelos de información de software
[1 *, p310–311] [3 *, p712–713] [4 *, s19.2]

Los modelos de información de software permiten modelar,
análisis y predicción de procesos de software y
atributos del producto de software para proporcionar respuestas a
preguntas relevantes y lograr proceso y producto
Metas de mejora. Los datos necesarios se pueden recopilar
y retenido en un repositorio; los datos pueden ser ana-
Lyzed y modelos pueden ser construidos. Validación
y perfeccionamiento de los modelos de información de software
ocurrir durante proyectos de software y después de proyectos
se completan para garantizar que el nivel de precisión
es suficiente y que sus limitaciones son conocidas
entendido Los modelos de información de software pueden

procesos que necesitan ser mejorados y construir Modelos predictivos basados en datos acumulados. En el ejemplo anterior, definiciones de defectos de software y horas de personal de esfuerzo de prueba más conteo serían necesarias reglas para defectos y esfuerzo para Obtener resultados de medición satisfactorios.

La medida en que el proceso del software es institucionalizado es importante; falta de institución La internacionalización de un proceso de software puede explicar por qué Los procesos de software "buenos" no siempre favorecen duce resultados anticipados. Los procesos de software pueden ser institucionalizado por adopción dentro del local unidad organizativa o en unidades más grandes de un empresa.

4.2. Calidad de los resultados de medición

[4 *, s3.4–3.7]

La calidad del proceso y la medición del producto. los resultados están determinados principalmente por la confiabilidad y validez de los resultados medidos. Medida- Mentores que no satisfacen estos criterios de calidad. puede resultar en interpretaciones incorrectas y defectuosas iniciativas de mejora de procesos de software. Otro propiedades deseables de las mediciones de software incluyen facilidad de recolección, análisis y presentación ción más una fuerte correlación entre causa y efecto.

El tema de medición de ingeniería de software en el Software Engineering Management KA describe un proceso para implementar un software programa de medida.

también se desarrollará para contextos distintos al software proyectos; por ejemplo, una información de software El modelo puede ser desarrollado para procesos que aplican en una organización, como la configuración de software gestión de raciones o garantía de calidad del software procesos a nivel organizacional.

Modelo de información de software basado en análisis la construcción implica el desarrollo, calibración, y evaluación de un modelo. Un software informa- El modelo de mación se desarrolla estableciendo un transformación hipotética de variables de entrada en salidas deseadas; por ejemplo, tamaño del producto y la complejidad podría transformarse en esti- esfuerzo necesario para desarrollar un producto de software uct utilizando una ecuación de regresión desarrollada a partir de datos observados de proyectos pasados. Un modelo es calibrado ajustando parámetros en el modelo para igualar los resultados observados de proyectos pasados; para ejemplo, el exponente en una regresión no lineal ecuación de sión puede cambiar aplicando la regresión a un conjunto diferente de proyectos pasados aparte de los proyectos utilizados para desarrollar el modelo.

Un modelo se evalúa comparando el cálculo resultados a resultados reales para un conjunto diferente de datos similares Hay tres posibles evaluaciones resultados:

1. los resultados calculados para un conjunto de datos diferente varían ampliamente de los resultados reales para esos datos conjunto, en cuyo caso el modelo derivado no es aplicable para el nuevo conjunto de datos y debe no se aplicará para analizar o hacer predicciones para futuros proyectos;

2. los resultados calculados para un nuevo conjunto de datos son cercano a los resultados reales para ese conjunto de datos, en cuyo caso se hacen ajustes menores a los parámetros del modelo para mejorar acuerdo;
3. resultados calculados para el nuevo conjunto de datos y conjuntos de datos posteriores están muy cerca y no Se necesitan ajustes al modelo.

La evaluación continua del modelo puede indicar Catear la necesidad de ajustes a lo largo del tiempo a medida que El texto en el que se aplica el modelo cambia.

El método Objetivos / Preguntas / Métricas (GQM) originalmente estaba destinado a establecer medidas ment actividades, pero también se puede utilizar para guiar Análisis y mejora de procesos de software.

Se puede usar para guiar software basado en análisis construcción de modelos de información; resultados obtenidos del modelo de información de software se puede utilizar para guiar la mejora del proceso.

El siguiente ejemplo ilustra la aplicación del método GQM:

- Objetivo: reducir la solicitud de cambio promedio tiempo de procesamiento en un 10% dentro de seis meses.
- Pregunta 1-1: ¿Cuál es el cambio de línea de base? solicitar tiempo de procesamiento?
- Métrica 1-1-1: promedio de solicitud de cambio tiempos de procesamiento en la fecha de inicio

Las técnicas de medición de procesos también proporcionan la información necesaria para medir los efectos de iniciativas de mejora de procesos. Proceso de medición Se pueden utilizar técnicas de aseguramiento para recolectar ambos Datos cuantitativos y cualitativos.

4.4.1. Medición cuantitativa de procesos Tecnicas

[4 *, s5.1, s5.7, s9.8]

El propósito de la medición cuantitativa del proceso. técnicas es recolectar, transformar y analizar proceso cuantitativo y datos de productos de trabajo que puede usarse para indicar dónde mejora el proceso son necesarios y evaluar los resultados de iniciativas de mejora de procesos. Cuantitativo Las técnicas de medición de procesos se utilizan para combinar ect y analizar datos en forma numérica a la cual las técnicas matemáticas y estadísticas pueden ser aplicado.

Los datos cuantitativos del proceso se pueden recopilar como Un subproducto de los procesos de software. Por ejemplo, la cantidad de defectos descubiertos durante el software las pruebas y las horas de personal gastadas pueden ser col- sido por medición directa, y el productiv- El descubrimiento del defecto puede obtenerse calculando Se descubren defectos por hora de personal.

Se pueden utilizar herramientas básicas para el control de calidad para analizar datos cuantitativos de medición de procesos

• Métrica 1-1-2: desviación estándar del cambio solicitar tiempos de procesamiento en la fecha de inicio
• Pregunta 1-2: ¿Cuál es el cambio actual? solicitar tiempo de procesamiento?
• Métrica 1-2-1: promedio de solicitud de cambio tiempos de procesamiento actualmente
• Métrica 1-2-2: desviación estándar del cambio solicitar tiempos de procesamiento actualmente

4.4. Técnicas de medición de procesos de software [1 *, c8]

Las técnicas de medición de procesos de software son utilizado para recopilar datos de proceso y productos de trabajo datos, transformar los datos en información útil, y analizar la información para identificar el proceso actividades que son candidatos para la mejora. En algunos casos, los nuevos procesos de software pueden ser necesario.

(p. ej., hojas de verificación, diagramas de Pareto, histogramas, diagramas de dispersión, gráficos de ejecución, gráficos de control y diagramas de causa y efecto) (ver Causa raíz Análisis en los Fundamentos de Ingeniería KA). En Además, se pueden utilizar varias técnicas estadísticas que van desde el cálculo de medianas y medias a métodos de análisis multivariados (ver Estadística Análisis en los Fundamentos de Ingeniería KA).

Datos recolectados usando medidas cuantitativas del proceso las técnicas de aseguramiento también pueden usarse como entradas a modelos de simulación (ver Modelado, Prototipo) ing, y Simulación en la Fundación de Ingeniería relaciones KA); estos modelos pueden usarse para evaluar la impacto de varios enfoques para el proceso de software mejora.

La clasificación de defectos ortogonales (ODC) puede ser utilizado para analizar medidas cuantitativas de procesos datos de ment. ODC se puede utilizar para agrupar detectado defectos en categorías y vincular los defectos en

cada categoría al proceso de software o software procesos de software donde se origina un grupo de defectos Nated (ver Caracterización de defectos en Soft-Ware de calidad KA). Defectos de la interfaz del software, por ejemplo, puede haberse originado durante un incidente equiparar el proceso de diseño de software; mejorando el proceso de diseño de software reducirá el número de defectos de la interfaz del software. ODC puede proporcionar datos cuantitativos para aplicar el análisis de causa raíz.

El control estadístico de procesos se puede utilizar para rastrear estabilidad del proceso, o la falta de estabilidad del proceso, utilizando cuadros de control.

4.4.2. Medición cualitativa de procesos Tecnicas [1 *, s6.4]

Técnicas cualitativas de medición de procesos. incluyendo entrevistas, cuestionarios y expertos juicio: puede usarse para aumentar cuantitativamente Técnicas de medición de procesos. Consen-sus técnicas, incluida la técnica Delphi, puede usarse para obtener consenso entre grupos de partes interesadas

5. Herramientas de proceso de ingeniería de software [1 *, s8.7]

Las herramientas de proceso de software son compatibles con la automatización de las funciones utilizadas para definir, implementar y administrar procesos de software individuales y vida de software modelos de ciclo. Incluyen editores para anotaciones tales como diagramas de flujo de datos, gráficos de estado, BPMN Diagramas IDEF0, redes de Petri y actividad UML diagramas En algunos casos, herramientas de proceso de software permitir diferentes tipos de análisis y simulaciones (por ejemplo, simulación de eventos discretos). En Además, herramientas empresariales de uso general, como una hoja de cálculo, puede ser útil.

Ingeniería de software asistida por computadora (CASO) las herramientas pueden reforzar el uso de integradas procesos, apoyar la ejecución del proceso defi niciones, y proporcionar orientación a los humanos en per-formando procesos bien definidos. Herramientas simples tales como procesadores de texto y hojas de cálculo pueden ser usado para preparar descripciones textuales de pro-cesses, actividades y tareas; Estas herramientas también son compatibles trazabilidad de puertos entre las entradas y salidas de múltiples procesos de software (como las partes interesadas análisis de necesidades, especificación de requisitos de software ción, arquitectura de software y software detallado diseño), así como los resultados del software pro procesos como documentación, componentes de software nents, casos de prueba e informes de problemas.

La mayoría de las áreas de conocimiento en esta *guía* Describir herramientas especializadas que se pueden utilizar para gestionar los procesos dentro de ese KA. En particular lar, consulte la Gestión de configuración de software KA para una discusión sobre la configuración del software herramientas de administración que se pueden usar para administrar procesos de construcción, integración y liberación para productos de software. Otras herramientas, como las para la gestión de requisitos y pruebas, son descrito en los KAs apropiados.

Las herramientas de proceso de software pueden apoyar proyectos que incluyan a geográficamente disperso (virtual) equipos Cada vez más, las herramientas de proceso de software son disponible a través de las instalaciones de computación en la nube como así como a través de infraestructuras dedicadas.

Un panel de control o panel de control del proyecto puede deshabilitar reproducir el proceso seleccionado y los atributos del producto para proyectos de software e indican medidas que están dentro de los límites de control y aquellos que necesitan acción correctiva

160 de 1189.

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	09 [1 *] airley 20 F	**09 [2 *] oore 20 METRO**	**ile 2011 erv[3 *] metro om S**	**03 [4 *] un 20 K**
1. Definición del proceso de software	p177	p295	p28–29, p36, c5	
1.1. Gestión de procesos de software			s26.1	p453–454
1.2. Infraestructura de proceso de software		p183, p186		p437–438
2. Ciclos de vida del software	c2	p190		
2.1. Categorías de procesos de software	prefacio	p294–295	c22, c23, c24	
2.2. Modelos de ciclo de vida del software	c2	s3.2	s2.1	
2.3. Adaptación de procesos de software	s2.7	p51		
2.4. Consideraciones prácticas		p188–190		
3. Evaluación de procesos de software y Mejora		p188, p194	c26	p397, c15
3.1. Modelos de evaluación de procesos de software		s4.5, s4.6	s26.5	p44–48
3.2. Evaluación de procesos de software Métodos	p322–331		s26.3	p44–48, s16.4
3.3. Mejora de procesos de software Modelos		p187–188	s26.5	s2.7
3.4. Clasificaciones continuas y por etapas	p28–34		s26.5	p39–45
4. Medición de software			s26.2	s18.1.1
4.1. Proceso de software y producto Medición	s6.3, p273		s26.2, p638	
4.2. Calidad de los resultados de medición				s3.4, s3.5, s3.6, s3.7
4.3. Modelos de información de software	p310–311		pag. 712–713	s19.2
4.4. Medición de procesos de software Tecnicas	s6.4, c8			s5.1, s5.7, s9.8
5. Herramientas de proceso de ingeniería de software	s8.7			

Page 161

LECTURAS ADICIONALES

Extensión de software a la guía del proyecto Cuerpo de Gestión del Conocimiento® (SWX) [5]

SWX proporciona adaptaciones y extensiones a la prácticas genéricas de gestión de proyectos documencionado en la *Guía PMBOK®* para gestionar proyectos de software. La principal contribución de esta extensión de la *Guía PMBOK®* es descriptiva ción de procesos que son aplicables para gestionar proyectos de software de ciclo de vida adaptativo.

D. Gibson, D. Goldenson y K. Kost, "Resultados de rendimiento de CMMI-Based Mejora de procesos "[6].

Este informe técnico resume la disponibilidad pública evidencia empírica capaz sobre el rendimiento resultados que pueden ocurrir como consecuencia de CMMI-Mejora de procesos basada. El informe contiene una serie de descripciones breves de casos que fueron creadas con la colaboración de representantes de 10 organizaciones que han logrado notables resultados de rendimiento cuantitativos a través de su Esfuerzos de mejora basados en CMMI.

CMMI ® for Development, Versión 1.3 [7].

CMMI ® for Development, la versión 1.3 proporciona un conjunto integrado de pautas de proceso para el desarrollo ing y mejora de productos y servicios. Estas Las pautas incluyen las mejores prácticas para el desarrollo y mejorando productos y servicios para cumplir con el necesidades de clientes y usuarios finales.

ISO / IEC 15504-1: 2004 Tecnología de la información nología — Evaluación del proceso — Parte 1: Conceptos y vocabulario [8].

Este estándar, comúnmente conocido como SPICE (Mejora de procesos de software y capacidad Determinación), incluye múltiples partes. Parte 1 proporciona conceptos y vocabulario para software procesos de desarrollo y negocios relacionados funciones administrativas. Otras partes de 15504 definir los requisitos y procedimientos para formando evaluaciones de procesos.

Referencias

[1 *] RE Fairley, *Gestión y liderazgo Proyectos de software* , computadora Wiley-IEEE Society Press, 2009.

[2 *] JW Moore, *La hoja de ruta hacia el software Ingeniería: una guía basada en estándares* , Wiley-IEEE Computer Society Press, 2006.

[3 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[4 *] SH Kan, *métricas y modelos en software Ingeniería de calidad* , 2ª ed., Addison-Wesley, 2002.

[5] Instituto de Gestión de Proyectos e IEEE Sociedad de Computación, *Extensión de Software a la Quinta Edición de la Guía PMBOK®* , ed .: Instituto de Gestión de Proyectos, 2013.

[6] D. Gibson, D. Goldenson y K. Kost, "Resultados de rendimiento de CMMI-Based Mejora de procesos ", software Instituto de Ingeniería, 2006; http: // resources.sei.cmu.edu/library/asset-view. cfm? assetID = 8065 .

[7] Equipo de producto CMMI, "CMMI para Desarrollo, Versión 1.3, "Software Instituto de Ingeniería, 2010; http: // resources.sei.cmu.edu/library/asset-view. cfm? assetID = 9661.

[8] *ISO / IEC 15504-1: 2004, información Tecnología — Evaluación de procesos — Parte 1: Conceptos y vocabulario* , ISO / IEC, 2004.

CAPÍTULO 9

MODELOS DE INGENIERÍA DE SOFTWARE Y métodos

SIGLAS

3GL	Lengua de 3ra generación
BNF	Forma Backus-Naur
FDD	Desarrollo basado en funciones
IDE	Desarrollo integrado Ambiente
PBI	Artículo del Backlog del producto
RAD	Desarrollo rápido de aplicaciones
UML	Lenguaje de modelado unificado
XP	Programación extrema

INTRODUCCIÓN

Modelos y métodos de ingeniería de software. imponer estructura a la ingeniería de software con el objetivo de hacer que esa actividad sea sistemática, repetible y, en última instancia, más orientado al éxito. El uso de modelos proporciona un enfoque del problema. resolución, una notación y procedimientos para el modelo construcción y análisis. Los métodos proporcionan un enfoque a la especificación sistemática, diseño, construcción, prueba y verificación del producto final software y productos de trabajo asociados.

Modelos y métodos de ingeniería de software. varían mucho en alcance, desde abordar un solo fase del ciclo de vida del software para cubrir la empresa Completo ciclo de vida del software. El énfasis en esto área de conocimiento (KA) está en ingeniero de software ing modelos y métodos que abarcan múltiples fases del ciclo de vida del software, ya que los métodos específicos para las fases del ciclo de vida único están cubiertos por otra KAs.

DESGLOSE DE TEMAS PARA MODELOS DE INGENIERÍA DE SOFTWARE Y métodos

Este capítulo sobre modelos de ingeniería de software y Los métodos se dividen en cuatro áreas temáticas principales:

- *Modelado* : discute la práctica general de modelado y presenta temas en model- principios de ing; propiedades y expresión de modelos; sintaxis de modelado, semántica y pragmática; y precondiciones, postcondi- iones e invariantes.
- *Tipos de modelos* : discute brevemente los modelos y agregación de submodelos y proporciona Algunas características generales de los tipos de modelos comúnmente encontrado en la ingeniería de software práctica.
- *Análisis de modelos* : presenta algunos de los técnicas de análisis comunes utilizadas en el modelo ing para verificar la integridad, consistencia, corrección rectitud, trazabilidad e interacción.
- *Métodos de ingeniería de software* : presenta un breve resumen del software de uso común métodos de ingeniería Las guías de discusión el lector a través de un resumen de heurística métodos, métodos formales, creación de prototipos y métodos ágiles

El desglose de temas para el Software Se muestran los modelos y métodos de ingeniería KA en la figura 9.1.

1. Modelado

El modelado de software se está generalizando técnica para ayudar a los ingenieros de software a comprender,

Figura 9.1. Desglose de temas para los modelos y métodos de ingeniería de software KA

ingeniero, y comunicar aspectos de la soft-
Ware a los interesados apropiados. Partes interesadas
son aquellas personas o partes que tienen un
o interés implícito en el software (por ejemplo,
usuario, comprador, proveedor, arquitecto, autor certificador
ity, evaluador, desarrollador, ingeniero de software y
quizás otros).

Si bien hay muchos lenguajes de modelado,
anotaciones, técnicas y herramientas en la literatura
y en la práctica, hay unificación general unificadora
conceptos que se aplican de alguna forma a todos ellos. los
Las siguientes secciones proporcionan antecedentes sobre estos
conceptos generales.

1.1. Principios de modelado

[1 *, c2s2, c5s1, c5s2] [2 *, c2s2] [3 *, c5s0]

El modelado proporciona al ingeniero de software
Un enfoque organizado y sistemático para la representación
enviando aspectos significativos del software bajo
estudio, facilitando la toma de decisiones sobre el soft-
artículos o elementos de la misma, y comunicar esos
decisiones importantes para otros en la parte interesada
comunidades Hay tres principios generales.
guiar tales actividades de modelado:

- *Modele lo esencial* : los buenos modelos no generalmente representan todos los aspectos o características de El software bajo todas las condiciones posibles. El modelado generalmente implica desarrollar solo aquellos aspectos o características del software que necesita respuestas específicas, abstrayendo cualquier Información no esencial. Este enfoque mantiene los modelos manejables y útiles.
- *Proporcionar perspectiva* : el modelado proporciona vistas del software en estudio utilizando un conjunto definido de reglas para la expresión de modelo dentro de cada vista. Esta perspectiva enfoque dirigido proporciona dimensionalidad a el modelo (por ejemplo, una vista estructural, vista de comportamiento, vista temporal, organización-visión nacional y otras vistas según corresponda). Organizar la información en puntos de vista los esfuerzos de modelado de software en específico

preocupaciones relevantes a esa vista usando el notación apropiada, vocabulario, métodos, y herramientas.
- *Habilitar comunicaciones efectivas* : modelado emplea el vocabulario del dominio de la aplicación del software, un lenguaje de modelado y expresión semántica (en otras palabras, media-ing dentro del contexto). Cuando se usa rigurosamente y sistemáticamente, este modelado resulta en Un enfoque de presentación de informes que facilita comunicación de información de software a partes interesadas del proyecto.

Un modelo es una *abstracción* o simplificación de
Un componente de software. Una consecuencia del uso
abstracción es que no hay una sola abstracción
describe completamente un componente de software. Más bien,
el modelo del software se representa como un
agregación de abstracciones que, cuando se toman
juntos: describa solo los aspectos seleccionados, per-
espectivas o vistas, solo aquellas que son necesarias
para tomar decisiones informadas y responder a la
razones para crear el modelo en primer lugar.
Esta simplificación lleva a un conjunto de supuestos
sobre el contexto dentro del cual se encuentra el modelo
colocado que también debe ser capturado en el modelo.
Luego, al reutilizar el modelo, estas suposiciones
se puede validar primero para establecer la relevancia de
El modelo reutilizado dentro de su nuevo uso y contexto.

1.2. Propiedades y expresión de modelos

[1 *, c5s2, c5s3] [3 *, c4s1.1p7, c4s6p3, c5s0p3]

Las propiedades de los modelos son las características distintivas
Tures de un modelo particular utilizado para caracterizar
su integridad, consistencia y corrección
dentro de la notación de modelado y herramientas elegidas
usado. Las propiedades de los modelos incluyen lo siguiente:

Los modelos se construyen para representar el mundo real.
objetos y sus comportamientos para responder específicos
preguntas sobre cómo se espera el software
para operar. Interrogando a los modelos, ya sea
mediante exploración, simulación o revisión, puede
exponer áreas de incertidumbre dentro del modelo y
El software al que se refiere el modelo. Estas
incertidumbres o preguntas sin respuesta con respecto a
los requisitos, diseño y / o implementación
entonces puede ser manejado apropiadamente.

El elemento de expresión principal de un modelo es
una entidad. Una entidad puede representar arte concreto
hechos (por ejemplo, procesadores, sensores o robots)
o artefactos abstractos (por ejemplo, software mod-
ules o protocolos de comunicación). Modelo enti-
los lazos están conectados a otras entidades usando relaciones
ions (en otras palabras, líneas u operadores textuales
en entidades objetivo). Expresión de entidades modelo.
puede lograrse usando texto o gráficos
lenguajes de modelado; ambos lenguaje de modelado
los tipos conectan entidades modelo a través de un lenguaje específico
construcciones de calibre. El significado de una entidad puede
estar representado por su forma, atributos textuales o
ambos. En general, la información textual se adhiere a
estructura sintáctica específica del lenguaje. El pre
cise significados relacionados con el modelado de contexto,
estructura o comportamiento usando estas entidades y
las relaciones dependen del lenguaje de modelado
utilizado, el rigor de diseño aplicado al modelado
esfuerzo, la vista específica que se está construyendo, y
la entidad a la que el elemento de notación específico
Se puede adjuntar. Múltiples vistas del modelo.
puede ser necesario para capturar la semántica necesaria
del software

Cuando se utilizan modelos compatibles con automa-
ción, los modelos pueden ser verificados por su integridad
y consistencia. La utilidad de estos controles
depende en gran medida del nivel de semántica y sincrónica
- rigor táctico aplicado al esfuerzo de modelado adicional

• *Integridad* : el grado en que todos los requisitos han sido implementados y verificado dentro del modelo.
• *Consistencia* : el grado en que el modelo no contiene requisitos contradictorios, funciones, restricciones, funciones o componentes descripciones
• *Corrección* : el grado en que el modelo satisface sus requisitos y especificaciones de diseño cationes y está libre de defectos.

ción para el soporte explícito de herramientas. La corrección es típica comprobado a través de simulación y / o revisión.

1.3. Sintaxis, Semántica y Pragmática

[2 * c2s2.2.2p6] [3 *, c5s0]

Los modelos pueden ser sorprendentemente engañosos. El hecho que un modelo es una abstracción con información faltante la mación puede conducir a uno a un falso sentido de comunicación Comprender completamente el software desde un solo modelo. Un modelo completo (ser "completo"

relativo al esfuerzo de modelado) puede ser una unión de múltiples submodelos y cualquier función especial modelos. Examen y toma de decisiones tive a un solo modelo dentro de esta colección de Los submodelos pueden ser problemáticos.

Comprender los significados precisos de mod- El desarrollo de construcciones también puede ser difícil. Modelado los idiomas se definen por sintáctico y semántico reglas. Para los lenguajes textuales, se define la sintaxis usando una gramática de notación que define un lenguaje válido construcciones de medida (por ejemplo, Backus-Naur Formulario (BNF)). Para lenguajes gráficos, la sintaxis es definido usando modelos gráficos llamados metamod- els. Al igual que con BNF, los metamodelos definen la validez construcciones sintácticas de un modelado gráfico idioma; el metamodelo define cómo estos Las estructuras se pueden componer para producir modelos válidos.

La semántica para lenguajes de modelado especifica el significado adjunto a las entidades y relaciones capturado dentro del modelo. Por ejemplo, un simple El diagrama de dos cajas conectadas por una línea está abierto. a una variedad de interpretaciones. Sabiendo que el diagrama en el que se colocan las cajas y conectado es un diagrama de objeto o un diagrama de actividad puede ayudar en la interpretación de este modelo.

Como cuestión práctica, generalmente hay un buen comprensión de la semántica de un determinado modelo de software debido al lenguaje de modelado seleccionado, cómo se usa ese lenguaje de modelado expresar entidades y relaciones dentro de ese modelo, la base de experiencia de los modeladores y contexto dentro del cual ha sido el modelado emprendido y así representado. El significado es com comunicado a través del modelo incluso en presencia de información incompleta a través de la abstracción; la pragmática explica cómo se encarna el significado en el modelo y su contexto y comunicado efectivamente a otros ingenieros de software.

Sin embargo, todavía hay casos en los que es necesaria la modelación y semántica. Por ejemplo, cualquier pieza del modelo importada de otro modelo o biblioteca debe ser examinado para supuestos semánticos que entran en conflicto en el nuevo entorno de modelado; Esto puede no ser obvio. El modelo debe ser verificado para documentar Suposiciones Si bien la sintaxis de modelado puede ser idéntico, el modelo puede significar algo bastante diferente en el nuevo entorno, que es un dif- contexto actual Además, considere eso como software madura y se hacen cambios, discordia semántica puede ser introducido, lo que lleva a errores. Con muchos ingenieros de software que trabajan en una parte del modelo tiempo junto con actualizaciones de herramientas y quizás nuevas requisitos, hay oportunidades para porciones del modelo para representar algo diferente de la intención del autor original y el modelo inicial contexto.

1.4. Condiciones previas, condiciones posteriores y Invariantes

[2 *, c4s4] [4 *, c10s4p2, c10s5p2p4]

Al modelar funciones o métodos, el software El ingeniero de hardware generalmente comienza con un conjunto de supuestos sobre el estado del software anterior a, durante y después de la función o método exe- Cálculos Estas suposiciones son esenciales para la corrección. funcionamiento directo de la función o método y son agrupados, para discusión, como un conjunto de condiciones previas, postcondiciones e invariantes.

• *Condiciones previas* : un conjunto de condiciones que deben estar satisfecho antes de la ejecución de la función o método Si estas condiciones previas no se cumplen antes de la ejecución de la función o método, la función o método puede producir errores Ous resultados.
• *Postcondiciones* : un conjunto de condiciones que es garantizado para ser verdad después de la función o El método se ha ejecutado con éxito. Típicamente, las condiciones posteriores representan cómo el estado del software ha cambiado, cómo eters pasados a la función o método tienen cambiado, cómo han cambiado los valores de datos, o cómo se ha visto afectado el valor de retorno.
• *Invariantes* : un conjunto de condiciones dentro de entorno operativo que persiste (en otras palabras, no cambies) antes y después ejecución de la función o método. Estas los invariantes son relevantes y necesarios para el software y el correcto funcionamiento de la función o método

2. Tipos de modelos

Un modelo típico consiste en una agregación de submodelos Cada submodelo es una descripción parcial ción y se crea para un propósito específico; puede estar compuesto por uno o más diagramas. los colección de submodelos puede emplear múltiples

Page 166

lenguajes de modelado o un solo lenguaje de modelado calibrador El lenguaje de modelado unificado (UML) reconoce una rica colección de modelos de diagramos Uso de estos diagramas, junto con el construcciones de lenguaje de modelado, produce tres Amplios tipos de modelos comúnmente utilizados: información modelos, modelos de comportamiento y modelos de estructura (ver sección 1.1).

2.1. Modelado de información

[1 *, c7s2.2] [3 *, c8s1]

Los modelos de información proporcionan un enfoque centrado en datos e información. Un modelo de información es un representación abstracta que identifica y define un conjunto de conceptos, propiedades, relaciones y conceptos limitaciones en las entidades de datos. La semántica o conceptual El modelo de información actual se utiliza a menudo para proporcionar algo de formalismo y contexto para que el software sea modelado como se ve desde la perspectiva del problema, sin preocuparse de cómo es este modelo en realidad mapeado a la implementación del software.
El modelo de información semántico o conceptual. es una abstracción y, como tal, incluye solo conceptos, propiedades, relaciones y restricciones necesario para conceptualizar la visión del mundo real de la información. Transformaciones posteriores de el modelo de información semántico o conceptual conducir a la elaboración de lógica y luego física Modelos de datos de calibración implementados en el software.

2.2. Modelado de comportamiento

[1 *, c7s2.1, c7s2.3, c7s2.4] [2 *, c9s2] [3 *, c5s4]

Los modelos de comportamiento identifican y definen la función funciones del software que se está modelando. Behav Los modelos iorales generalmente toman tres formas básicas: máquinas de estado, modelos de flujo de control y datos Modelos de flujo. Las máquinas de estado proporcionan un modelo del software como una colección de estados definidos, eventos y transiciones. Las transiciones de software de un estado a otro a través de un vigilado o evento desencadenante sin vigilancia que ocurre en el entorno modelado Modelos de control de flujo representan cómo una secuencia de eventos causa procesos para ser activado o desactivado. Comportamiento del flujo de datos ior se tipifica como una secuencia de pasos donde los datos se mueve a través de procesos hacia almacenes de datos o sumideros de datos.

2.3. Modelado de estructura

[1 *, c7s2.5, c7s3.1, c7s3.2] [3 *, c5s3] [4 *, c4]

Los modelos de estructura ilustran lo físico o lógico composición de software a partir de sus diferentes componentes piezas ponentes. El modelado de estructuras establece el límite definido entre el software siendo implementado o modelado y el medio ambiente en el que se va a operar. Algunas estructuras comunes Las construcciones turales utilizadas en el modelado de estructuras son composición, descomposición, generalización y especialización de entidades; identificación de relaciones evasivas y cardinalidad entre entidades; y la definición de proceso o funcional interfaces. Diagramas de estructura proporcionados por el UML para el modelado de estructuras incluyen clase, componente, diagramas de objeto, despliegue y empaquetado.

3. Análisis de modelos

El desarrollo de modelos permite el software. diseñar una oportunidad para estudiar, razonar sobre y entender la estructura, función, operauso nacional y consideraciones de montaje asociadas Ciado con el software. Análisis de construidos Se necesitan modelos para garantizar que estos modelos completo, consistente y lo suficientemente correcto para servir su propósito previsto para los interesados.

Las secciones que siguen describen brevemente técnicas de análisis generalmente utilizadas con softmodelos de software para garantizar que el ingeniero de software y otras partes interesadas relevantes ganan apropiadamente valor del desarrollo y uso de modelos.

3.1. Analizando la integridad

[3 *, c4s1.1p7, c4s6] [5 *, p8–11]

Para tener un software que satisfaga plenamente las necesidades de las partes interesadas, la integridad es crítica, desde El proceso de obtención de requisitos para implementar el código Mentación. La integridad es el grado en que Se han implementado todos los requisitos especificados. mented y verificado. Los modelos pueden ser revisados para integridad mediante una herramienta de modelado que utiliza tecnología niques como el análisis estructural y el espacio de estados análisis de accesibilidad (que garantiza que todas las rutas en los modelos de estado son alcanzados por algún conjunto de correctos entradas); los modelos también pueden ser verificados para completar ness manualmente mediante inspecciones u otra revisión técnicas (ver el Software Quality KA). Errores

Page 167

y advertencias generadas por estas herramientas de análisis y encontrado por inspección o revisión indica probable acciones correctivas necesarias para garantizar la integridad de las modelos.

3.2. Análisis de consistencia

[3 *, c4s1.1p7, c4s6] [5 *, p8–11]

y pseudocódigo, escrito a mano y generado por herramientas código, casos e informes de prueba manuales y automatizados, y archivos y datos. Estos productos de trabajo pueden ser relacionado a través de varias relaciones de dependencia (por ejemplo, usos, implementos y pruebas). Tan suave se está desarrollando, administrando, manteniendo o extendido, es necesario mapear y controlar estos

La consistencia es el grado en que los modelos no contener requisitos contradictorios, afirmaciones, constraints, funciones o descripciones de componentes. Por lo general, se realiza la comprobación de consistencia con la herramienta de modelado utilizando un análisis automatizado función; los modelos también pueden ser verificados por consistencia tenencia manualmente usando inspecciones u otra revisión técnicas (ver el Software Quality KA). Como con integridad, errores y advertencias generadas por estas herramientas de análisis y encontrado por inspección o revisión indica la necesidad de medidas correctivas.

3.3. Analizando la corrección

[5 *, p8-11]

La corrección es el grado en que un modelo satisface cumple sus requisitos de software y software especificaciones de diseño, está libre de defectos y, por último, mately satisface las necesidades de los interesados. Analizando para la corrección incluye verificar la corrección sintáctica rectitud del modelo (es decir, uso correcto del modelado de gramática y construcciones del lenguaje) y verificar la corrección semántica del modelo (que es decir, el uso de las construcciones de lenguaje de modelado para representar correctamente el significado de lo que es siendo modelado). Para analizar un modelo sintáctico y la corrección semántica, uno lo analiza, ya sea automáticamente (por ejemplo, usando el modelado herramienta para verificar la corrección sintáctica del modelo) o manualmente (usando inspecciones u otra revisión técnicas): búsqueda de posibles defectos y luego eliminar o reparar los defectos confirmados antes de lanzar el software para su uso.

3.4. Trazabilidad

[3 *, c4s7.1, c4s7.2]

El desarrollo de software generalmente implica el uso, Creación y modificación de muchos productos de trabajo. tales como documentos de planificación, especificación de proceso iones, requisitos de software, diagramas, diseños relaciones de trazabilidad para demostrar requisitos de hardware consistentes con el software modelo (consulte Rastreo de requisitos en el software Requisitos KA) y los muchos productos de trabajo. El uso de la trazabilidad generalmente mejora la gestión mación de productos de trabajo de software y proconsistencia cess; también proporciona garantías para estacar titulares que todos los requisitos han sido satisfechos. La trazabilidad permite el análisis de cambios una vez que el software Se desarrolla y libera software, ya que las relaciones a los productos de trabajo de software se pueden atravesar fácilmente para evaluar el impacto del cambio. Herramientas de modelado típicamente proporcionar algunos medios automatizados o manuales para especificar ify y gestionar los enlaces de trazabilidad entre require-Mentes, diseño, código y / o entidades de prueba según corresponda representado en los modelos y otros trabajos de software productos (Para más información sobre trazabilidad, ver el Software Configuration Management KA).

3.5. Análisis de interacción

[2 *, c10, c11] [3 *, c29s1.1, c29s5] [4 *, c5]

El análisis de interacción se centra en la comunicación o controlar las relaciones de flujo entre entidades dirigido para realizar una tarea o función específica dentro del modelo de software. Este examen de análisis Ines el comportamiento dinámico de las interacciones entre diferentes partes del modelo de software, incluyendo otras capas de software (como la operación sistema, middleware y aplicaciones). Eso también puede ser importante para algunas aplicaciones de software para examinar las interacciones entre las comunicaciones aplicación de software de computadora y la interfaz de usuario software. Algunos entornos de modelado de software proporcionar instalaciones de simulación para estudiar aspectos de El comportamiento dinámico del software modelado. Pasohacer ping a través de la simulación proporciona un análisis opción para que el ingeniero de software revise el diseño de interacción y verificar que los diferentes partes del software trabajan juntas para proporcionar funciones previstas

Page 168

4. Métodos de ingeniería de software

Los métodos de ingeniería de software proporcionan una organización enfoque nizado y sistemático para desarrollar Ware para una computadora de destino. Hay numerosos métodos para elegir, y es importante para que el ingeniero de software elija un apropiado método o métodos para el desarrollo de software tarea en cuestión; esta elección puede tener un efecto dramático sobre el éxito del proyecto de software. Uso de estos métodos de ingeniería de software junto con personas del conjunto de habilidades y herramientas adecuadas permiten el software ingenieros para visualizar los detalles del software y finalmente transformar la representación en una Conjunto de trabajo de código y datos.

Los métodos de ingeniería de software seleccionados son dismaldijo a continuación. Las áreas temáticas se organizan en discusiones sobre métodos heurísticos, métodos formales ods, métodos de creación de prototipos y métodos ágiles.

4.1. Métodos heurísticos

diseños de bases de datos o repositorios de datos se encuentra localmente en software empresarial, donde los datos se gestiona activamente como un sistema de negocios recurso o activo.

- *Análisis orientado a objetos y métodos de diseño. ods* : se representa el modelo orientado a objetos como una colección de objetos que encapsulan datos y relaciones e interactuar con otros objetos a través de métodos. Los objetos pueden ser Artículos del mundo real o artículos virtuales. Lo suave-El modelo de cerámica se construye usando diagramas. para constituir vistas seleccionadas del software. Refinamiento progresivo del software mod-Els lleva a un diseño detallado. El detallado el diseño evoluciona luego a través del éxito iteración cesiva o transformada (usando algunos mecanismo) en la vista de implementación del modelo, donde el código y el paquete enfoque de ing para un producto de software eventual lanzamiento y despliegue se expresa.

[1 *, c13, c15, c16] [3 *, c2s2.2, c5s4.1, c7s1,]

Los métodos heurísticos son aquellos basados en la experiencia.
métodos de ingeniería de software que han sido y
se practican bastante ampliamente en la industria del software
tratar. Esta área temática contiene tres amplios debates.
categorías de sion: análisis estructurado y diseño
métodos, métodos de modelado de datos y objetos
Análisis orientado y métodos de diseño.

- *Análisis estructurado y métodos de diseño* :
  El modelo de software se desarrolla principalmente
  desde un punto de vista funcional o conductual,
  a partir de una vista de alto nivel del soft-
  Ware (incluidos datos y elementos de control)
  y luego se descompone o refina progresivamente
  ing los componentes del modelo a través del aumento
  Diseños detalladamente detallados. El diseño detallado
  finalmente converge a detalles muy específicos
  o especificaciones del software que deben ser
  codificado (a mano, generado automáticamente, o
  ambos), construido, probado y verificado.
- *Métodos de modelado de datos* : el modelo de datos es
  construido desde el punto de vista de los datos o
  información utilizada Tablas de datos y relación
  Las naves definen los modelos de datos. Este mod de datos
  El método de eling se utiliza principalmente para definir
  y analizar los requisitos de datos que respaldan

4.2. Métodos formales [1 *, c18] [3 *, c27] [5 *, p8–24]

Los métodos formales son métodos de ingeniería de software.
ds utilizados para especificar, desarrollar y verificar el software
Ware mediante la aplicación de una rigurosa matemática
Notación y lenguaje basados icamente. A través del uso
de un lenguaje de especificación, el modelo de software
se puede verificar la coherencia (en otras palabras,
falta de ambigüedad), integridad y corrección
de forma sistemática y automatizada o semiautomatizada
Moda. Este tema está relacionado con el análisis formal
sis sección en los requisitos de software KA.

Esta sección aborda los lenguajes de especificación,
refinamiento y derivación del programa, verificación formal
cación e inferencia lógica.

- *Idiomas de* especificación: especificación
  los idiomas proporcionan la base matemática
  por un método formal; especificación lan-
  los medidores son computadoras formales de nivel superior
  idiomas (en otras palabras, no un clásico
  Programa de lenguaje de tercera generación (3GL)
  lenguaje ming) utilizado durante el software
  especificación, análisis de requisitos y /
  o etapas de diseño para describir entradas específicas /
  comportamiento de salida. Los lenguajes de especificación son
  idiomas no directamente ejecutables; son

Page 169

típicamente compuesto por una notación y sintaxis,
semántica para el uso de la notación, y un conjunto de
relaciones permitidas para objetos.

- *Programa de refinamiento y derivación* : Pro-
  refinamiento de gramo es el proceso de crear un
  especificación de nivel inferior (o más detallado)
  usando una serie de transformaciones. Es a través de
  transformaciones sucesivas que el software
  ingeniero deriva una representación ejecutable
  de un programa. Las especificaciones pueden ser refinadas
  agregando detalles hasta que el modelo pueda ser formal-
  en un lenguaje de programación 3GL o en
  una parte ejecutable de la especificación elegida
  idioma de la nación El refinamiento de esta especificación
  hecho posible mediante la definición de especificaciones con
  propiedades semánticas precisas; las especificaciones
  debe establecer no solo las relaciones entre
  entidades pero también los significados exactos de tiempo de ejecución
  esas relaciones y operaciones.
- *Verificación formal* : la *verificación del* modelo es
  un método de verificación formal; normalmente
  implica realizar una exploración del espacio de estado
  análisis de alcance o alcance para demostrar
  que el diseño de software representado tiene o
  conserva ciertas propiedades del modelo de inter
  est. Un ejemplo de comprobación de modelo es un
  análisis que verifica el comportamiento correcto del programa
  ior bajo toda posible intercalación de evento o
  Llegadas de mensajes. El uso de la verificación formal
  cación requiere un modelo rigurosamente especificado
  del software y su entorno operativo
  ment este modelo a menudo toma la forma de un
  máquina de estado finito u otra formalmente definida
  autómata.
- *Inferencia lógica* : la inferencia lógica es un

4.3. Métodos de prototipos
[1 *, c12s2] [3 *, c2s3.1] [6 *, c7s3p5]

La creación de prototipos de software es una actividad que generalmente
crea una versión incompleta o mínimamente funcional
siones de una aplicación de software, generalmente para probar
presentando nuevas características específicas, solicitando comentarios
sobre requisitos de software o interfaces de usuario,
explorando los requisitos de software, software
diseño u opciones de implementación, y / o ganar
alguna otra información útil sobre el software. los
ingeniero de software selecciona un método de creación de prototipos para
entender los aspectos menos entendidos o comp
componentes del software primero; este enfoque está en
contraste con otros métodos de ingeniería de software
que generalmente comienzan con el desarrollo más
porciones entendidas primero. Típicamente, el protocolo
prototipo de objeto no tiene se convierte en el software final
producto sin un extenso retrabajo de desarrollo
o refactorización.

Esta sección discute los estilos de creación de prototipos, tar-
obtiene, y técnicas de evaluación en breve.

- *Estilo de creación de prototipos* : aborda los distintos
  enfoques para desarrollar prototipos. Proto-
  los tipos se pueden desarrollar como código desechable
  o productos de papel, como evolución de un trabajo
  ing diseño, o como una especificación ejecutable.
  Los diferentes procesos del ciclo de vida de la creación de prototipos son
  típicamente usado para cada estilo. El estilo cho-
  sen se basa en el tipo de resultados del proyecto
  necesidades, la calidad de los resultados necesarios, y
  La urgencia de los resultados.
- *Objetivo de creación de prototipos* : el objetivo del pro-
  la actividad total es el producto específico que se está

método de diseño de software que involucra especificando precondiciones y postcondiciones alrededor de cada bloque significativo del diseño, y, utilizando la lógica matemática, desarrollando la prueba de que esas condiciones previas y posteriores Las condiciones deben mantenerse bajo todas las entradas. Esta proporciona una manera para que el ingeniero de software predecir el comportamiento del software sin tener para ejecutar el software Algunos integrados Los entornos de desarrollo (IDE) incluyen formas de representar estas pruebas junto con el Diseño o código.

servido por el esfuerzo de creación de prototipos. Ejemplos de los objetivos de creación de prototipos incluyen un requisito especificación, un elemento de diseño arquitectónico o componente, un algoritmo o un humano interfaz de usuario de la máquina.

- *Técnicas de evaluación de prototipos* : un prototype puede usarse o evaluarse en un número Ber de maneras por el ingeniero de software o otras partes interesadas del proyecto, impulsadas principalmente por las razones subyacentes que llevaron a prototype desarrollo en primer lugar. Prolos tipos pueden ser evaluados o probados contra el software implementado real o contra

un conjunto objetivo de requisitos (por ejemplo, un requisitos prototipo); el prototipo puede también sirve como modelo para un software futuro esfuerzo de desarrollo (por ejemplo, como en un usuario especificación de interfaz).

4.4. Métodos ágiles

[3 *, c3] [6 *, c7s3p7] [7 *, c6, aplicación. UNA]

Los métodos ágiles nacieron en la década de 1990 del Necesidad de reducir la aparente gran sobrecarga asociada con métodos pesados basados en planes utilizados en proyectos de desarrollo de software a gran escala. Los métodos ágiles se consideran métodos livianos. ods en que se caracterizan por cortos, iteraciclos de desarrollo activos, equipos autoorganizados, diseños más simples, refactorización de código, prueba de manejo desarrollo, participación frecuente del cliente, y un énfasis en crear un trabajo demostrable producto con cada ciclo de desarrollo.

Muchos métodos ágiles están disponibles en la literatura erature; algunos de los enfoques más populares, que se discuten aquí brevemente, incluyen Rapid Desarrollo de aplicaciones (RAD), eXtreme Pro-Gramming (XP), Scrum y Feature-Driven Desarrollo (FDD).

- *RAD:* métodos rápidos de desarrollo de software se utilizan principalmente en datos intensivos, negocios- Desarrollo de aplicaciones de sistemas. El RAD El método está habilitado con datos especiales herramientas de desarrollo base utilizadas por software ingenieros para desarrollar, probar e implementar rápidamente Aplicaciones comerciales nuevas o modificadas.
- *XP* : este enfoque utiliza historias o escenarios para requisitos, desarrolla pruebas primero, tiene participación directa del cliente en el equipo (típicamente definiendo pruebas de aceptación), usos programación de pares y proporciona continuidad Código de refactorización e integración. Cuentos se descomponen en tareas, priorizadas, estiapareado, desarrollado y probado. Cada incremento El software se prueba con sistemas automatizados. y pruebas manuales; un incremento puede ser lanzado con frecuencia, como cada par de semanas más o menos.
- *Scrum* : este enfoque ágil es más proyecto amigable con la administración que los demás. los scrum master gestiona las actividades dentro de el incremento del proyecto; cada incremento es llamado sprint y no dura más de 30 dias. Una lista de elementos de la cartera de productos (PBI) es desarrollado a partir del cual se identifican las tareas, definido, priorizado y estimado. Un trabajo- Se prueba la versión ing del software y lanzado en cada incremento. Scrum diario Las reuniones aseguran que el trabajo se gestione según lo planeado.
- *FDD:* este es un modelo, corto, iterativo. enfoque de desarrollo de software tive utilizando Un proceso de cinco fases: (1) desarrollar un producto modelo para abarcar la amplitud del dominio, (2) crear la lista de necesidades o características, (3) construir el plan de desarrollo de características, (4) desarrollar diseños para funciones específicas de iteración, y (5) codifique, pruebe y luego integre las características. FDD es similar a un software incremental enfoque de desarrollo; también es similar a XP, excepto que se asigna la propiedad del código a individuos más que al equipo. FDD enfatiza un enfoque arquitectónico general al software, que promueve la construcción de presentarse correctamente la primera vez en lugar de enfatizando la refactorización continua.

Hay muchas más variaciones de métodos ágiles. Ods en la literatura y en la práctica. Tenga en cuenta que siempre habrá un lugar para el peso pesado, métodos de ingeniería de software basados en planes también lugares donde brillan los métodos ágiles. Existen nuevos métodos derivados de combinaciones de agile y métodos basados en planes donde los profesionales son definiendo nuevos métodos que equilibren las características necesario tanto en peso pesado como ligero métodos basados principalmente en organismos prevalecientes Necesidades empresariales nacionales. Estas necesidades comerciales, como lo representa típicamente parte del proyecto las partes interesadas, deberían conducir la elección en utilizando un método de ingeniería de software sobre otro o al construir un nuevo método a partir de mejores características de una combinación de software de ingeniería métodos de neering.

Página 171

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	03 gen Re Bu [1 *]	02 alcer 20 d B [2 *] ellor an METRO	ile 2011 erv metro gm S [3 *]	es 1999 age-Jon PAG [4 *]	g 1990 en W [5 *]	08 oreja 20 sh [6 *] Arroyo	03 er 20 rn tu d T [7 *] m an oeh segundo
1. Modelado							
1.1. Modelado Principios	c2s2, c5s1, c5s2	c2s2	c5s0				
1.2. Propiedades y expresión de Modelos	c5s2, c5s3		c4s1.1p7, c4s6p3, c5s0p3				
1.3. Sintaxis, Semántica, y Pragmática		c2s2.2.2 p6	c5s0				
1.4. Precondiciones, Condiciones posteriores, y Invariantes		c4s4		c10s4p2, c10s5 p2p4			
2. Tipos de modelos							
2.1. Información Modelado	c7s2.2		c8s1				
2.2. Comportamiento Modelado	c7s2.1, c7s2.3, c7s2.4	c9s2	c5s4				
2.3. Estructura Modelado	c7s2.5, c7s3.1, c7s3.2		c5s3	c4			
3. Análisis de modelos							
3.1. Analizando para Lo completo			c4s1.1p7, c4s6		pp8-11		
3.2. Analizando para Consistencia			c4s1.1p7, c4s6		pp8-11		
3.3. Analizando para Exactitud					pp8-11		
3.4. Trazabilidad			c4s7.1, c4s7.2				
3.5. Interacción Análisis		c10, c11	c29s1.1, c29s5	c5			

Page 172

02 03

	03 gen 20 Bu [1 *]	**alcer 20 d B[2 *] ellor an METRO**	**ile 2011 erv[3 *] metro gm S**	**es 1999 [4 *] age-Jon PAG**	**g 1990 en [5 *] W**	**08 oreja 20 sh [6 *] Arroyo**	**er 20 rn tu d T[7 *] m an oeh segundo**
4. Software Métodos de ingeniería							
4.1. Heurístico Métodos	c13, c15, c16		c2s2.2, c7s1, c5s4.1				
4.2. Métodos formales	c18		c27		pp8–24		
4.3. Prototipos Métodos	c12s2		c2s3.1			c7s3p5	
4.4. Métodos ágiles			c3			c7s3p7	c6, app. UNA

Page 173

Referencias

[1 *] D. Budgen, *Diseño de software* , 2ª ed., Addison-Wesley, 2003.

[2 *] SJ Mellor y MJ Balcer, *ejecutable UML: una base para el modelo Arquitectura* , 1ª ed., Addison-Wesley, 2002.

[3 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[4 *] M. Page-Jones, *Fundamentos del objeto-Diseño orientado en UML* , 1ª ed., Addison-Wesley, 1999.

[5 *] JM Wing, "Introducción de un especificador a Métodos formales, " *Computer* , vol. 23, no. 9, 1990, págs. 8, 10–23.

[6 *] JG Brookshear, *Ciencias de la computación: un Descripción general* , décima edición, Addison-Wesley, 2008.

[7 *] B. Boehm y R. Turner, *Equilibrio de la agilidad y disciplina: una guía para perplejos* , Addison-Wesley, 2003.

CAPÍTULO 10

CALIDAD DE SOFTWARE

SIGLAS

CMMI	Modelo de Capacidad de Madurez Integración
CoSQ	Costo de la calidad del software
Cunas	Comercial fuera de la plataforma Software
FMEA	Modo de Fallos y Análisis de Efectos
TLC	Análisis del árbol de fallos
PDCA	Planificar-Hacer-Verificar-Actuar
PDSA	Planificar-hacer-estudiar-actuar
QFD	Despliegue de la función de calidad
SPI	Mejora de procesos de software
SQA	Aseguramiento de la calidad del software
SQC	Control de calidad del software
SQM	Gestión de calidad de software
TQM	Gestión de calidad total
V&V	Verificación y validación

calidad ", donde el" cliente es el árbitro final "
[3 *, p8].

Más recientemente, la calidad del software se define como la "Capacidad del producto de software para satisfacer lo declarado y necesidades implícitas en condiciones específicas "[4] y como "el grado en que un producto de software cumple con los requisitos establecidos; sin embargo, calidad depende del grado en que esos establecen Los requisitos detallados representan con precisión necesidades, deseos y expectativas del titular ”[5]. Ambos las definiciones abarcan la premisa de conformidad a los requisitos Ninguno de los dos se refiere a tipos de requisitos. ments (p. ej., funcional, fiabilidad, rendimiento, fiabilidad, o cualquier otra característica). Significativo sin embargo, estas definiciones enfatizan que La calidad depende de los requisitos.

Estas definiciones también ilustran otra razón para la prevalencia de la calidad del software a través de esta *guía* : una ambigüedad frecuente del *software calidad* versus *requisitos de calidad de software* ("Las *-ilicias* " es una abreviatura común). Software los requisitos de calidad son en realidad atributos de (o restricciones sobre) requisitos funcionales (qué

INTRODUCCIÓN

¿Qué es la calidad del software y por qué es tan importante? Tant que se incluye en muchas áreas de conocimiento (KAs) de la *Guía SWEBOK* ?

Una razón es que el término *calidad de software* es sobrecargado. La calidad del software puede referirse a: características capaces de productos de software, a la hasta qué punto un producto de software en particular ver esas características y procesos, herramientas, y técnicas utilizadas para lograr esos caracteres istica. Con los años, autores y organizaciones. han definido el término calidad de manera diferente. A Phil Crosby, fue "conformidad con los requisitos" [1] Watts Humphrey se refiere a él como "lograr excelentes niveles de "aptitud para el uso" [2]. Mientras que IBM acuñó la frase "impulsada por el mercado esto a veces se caracteriza como "aptitud para

175 de 1189.

Figura 10.1. Desglose de temas para la calidad del software KA

uso ". El valor de las partes interesadas se expresa en requisitos. Para los productos de software, las partes interesadas pueden valor precio (lo que pagan por el producto), plomo tiempo (qué tan rápido obtienen el producto) y software calidad.

Este KA aborda las definiciones y proporciona un Resumen de prácticas, herramientas y técnicas para definiendo la calidad del software y para evaluar el estado de la calidad del software durante el desarrollo, mantenimiento y despliegue. Referencias citadas Proporcionar detalles adicionales.

DESGLOSE DE TEMAS PARA CALIDAD DE SOFTWARE

El desglose de temas para la calidad del software KA se presenta en la Figura 10.1.

1. Fundamentos de calidad del software

el sistema lo hace). Los requisitos de software pueden también especifique el uso de recursos, una comunicación protocolo, o muchas otras características. Este KA intenta claridad utilizando *la calidad del software* en el sentido más amplio de las definiciones anteriores y mediante el uso de *requisitos de calidad de software* como limitaciones en los requisitos funcionales. Software la calidad se logra mediante la conformidad con todos los requisitos independientemente de qué característica es especi fied o cómo se agrupan o nombran los requisitos.

La calidad del software también se considera en muchos los KE SWEBOK porque es un parámetro básico Después de un esfuerzo de ingeniería de software. Para todos productos buscados, el objetivo principal es entregar valor máximo de las partes interesadas, mientras se equilibra el limitaciones del costo y cronograma de desarrollo; y disciplinado

Un ingeniero de software debe entender la calidad ity conceptos, características, valores y sus aplicación al software en desarrollo o mantenimiento. El concepto importante es que el Los requisitos de software definen la calidad requerida atributos del software. Requisitos de Software influir en los métodos de medición y aceptación criterios de evaluación para evaluar el grado en que el software y la documentación relacionada logran Los niveles de calidad deseados.

1.1. Ingeniería de Software Cultura y Ética

[3 *, c1s4] [6 *, c2s3.5]

Se espera que los ingenieros de software compartan un compromiso con la calidad del software como parte de su cultura. Una cultura saludable de ingeniería de software incluye muchas características, incluida la comprensión

Llegar a un acuerdo sobre lo que constituye calidad para todos los interesados y comunicando claramente ese acuerdo para ingenieros de software requiere que

que compensa entre costo, cronograma y calidad son un inquilino básico de la ingeniería de cualquier producto uct. Una fuerte ética de ingeniería de software supone

Page 176

que los ingenieros informan con precisión la información, diciones y resultados relacionados con la calidad.

La ética también juega un papel importante en el software calidad, cultura y actitudes del software ingenieros La IEEE Computer Society y el ACM ha desarrollado un código de ética y pro práctica profesional (ver Códigos de Ética y Pro-Conducta profesional en la Ingeniería del Software Práctica profesional KA).

1.2. Valor y costos de calidad [7 *, c17, c22]

Definir y luego lograr la calidad del software es no es simple. Las características de calidad pueden o pueden no ser requerido, o pueden ser requeridos a un mayor o menor grado, y las compensaciones pueden ser hecho entre ellos. Para ayudar a determinar el nivel de calidad de software, es decir, lograr la participación de los interesados valor, esta sección presenta el costo de la calidad del software ity (CoSQ): un conjunto de mediciones derivadas de La evaluación económica de la calidad del software procesos de desarrollo y mantenimiento. los Las mediciones de CoSQ son ejemplos de procesos. medidas que pueden usarse para inferir características características de un producto.

La premisa subyacente de la CoSQ es que el nivel de calidad en un producto de software puede ser inferido del costo de las actividades relacionadas con el trato ing con las consecuencias de la mala calidad. Pobre calidad significa que el producto de software no totalmente "satisfacer las necesidades declaradas e implícitas cumplidos requisitos ". Hay cuatro costos de calificación Categorías de la ity: prevención, evaluación, falla interna ure y falla externa.

Los costos de prevención incluyen inversiones en software esfuerzos de mejora de procesos, infraestructura de calidad Ture, herramientas de calidad, capacitación, auditorías y gestión comentarios de ment. Estos costos generalmente no son específicos a un proyecto; abarcan la organización. Evaluación Los costos surgen de las actividades del proyecto que encuentran defectos. Estas actividades de evaluación se pueden clasificar en costos de revisiones (diseño, pares) y costos de prueba ing (prueba de unidad de software, integración de software, prueba de nivel del sistema, prueba de aceptación); evaluación los costos se extenderían al software subcontratado proveedores. Los costos de fallas internas son aquellos que se incurre para reparar defectos encontrados durante la evaluación actividades y descubiertas antes de la entrega de la producto de software para el cliente. Fallo externo Los costos incluyen actividades para responder al software Problemas descubiertos después de la entrega al cliente.

Los ingenieros de software deberían poder usar CoSQ métodos para determinar los niveles de calidad del software y también debería ser capaz de presentar alternativas de calidad nativos y sus costos para que las compensaciones entre costo, cronograma y entrega del valor de los interesados Puede ser hecho.

1.3. Modelos y características de calidad [3 *, c24s1] [7 *, c2s4] [8 *, c17]

Terminología para las características de calidad del software. difiere de una taxonomía (o modelo de software calidad) a otro, cada modelo tal vez teniendo un número diferente de niveles jerárquicos y un diferente número total de características. Varios los investigadores han producido modelos de software de calidad características o atributos que pueden ser útiles para discutir, planificar y calificar la calidad de productos de software. ISO / IEC 25010: 2011 [4] define la calidad del producto y la calidad en uso como dos Modelos de calidad relacionados. Apéndice B en el *SWE-La Guía BOK* proporciona una lista de estándares aplicables por cada KA. Los estándares para este KA cubren varios formas de caracterizar la calidad del software.

1.3.1. Calidad del proceso de software

Gestión de calidad de software e ingeniería de software El establecimiento del proceso de interrelación tiene una relación directa con La calidad del producto de software.

Modelos y criterios que evalúan la capacidad los lazos de las organizaciones de software son principalmente proyectos ect consideraciones de organización y gestión y, como tal, están cubiertos en el Software Engine-Ingeniería de software y gestión Práctica profesional KAs.

No es posible distinguir completamente entre la calidad del producto de la calidad del proceso porque el proceso Los resultados incluyen productos. Determinando si un proceso tiene la capacidad de promover consistentemente Duce productos de calidad deseada no es simple. El proceso de ingeniería de software, discutido en el proceso de ingeniería de software KA, influ-enfatiza las características de calidad de los productos de software uct, que a su vez afectan la calidad percibida por partes interesadas

Page 177

1.3.2. Calidad del producto de software

El ingeniero de software, en primer lugar, debe determinar El verdadero propósito del software. A este respecto, los requisitos de las partes interesadas son primordiales y incluir requisitos de calidad además de funciones requisitos nacionales Por lo tanto, los ingenieros de software tener la responsabilidad de obtener requisitos de calidad eso puede no ser explícito desde el principio y subestimar destacan su importancia y el nivel de diferencia culto en alcanzarlos. Todo el desarrollo de software procesos (p. ej., obtención de requisitos, diseño, construyendo, construyendo, revisando, mejorando la calidad ity) están diseñados con estos requisitos de calidad en mente y puede llevar un desarrollo adicional costos si atributos tales como seguridad, y La fiabilidad es importante. El desarrollo adicional los costos de las opciones ayudan a garantizar que la calidad del producto tiene ser intercambiado con los beneficios anticipados.

El término producto de trabajo significa cualquier artefacto que es el resultado de un proceso utilizado para crear el producto de software final. Ejemplos de productos de trabajo uct incluye una especificación de sistema / subsistema, un especificación de requisitos de software para un software componente de software de un sistema, un diseño de software descripción, código fuente, documentación de prueba de software tation o informes. Mientras que algunos tratamientos de calidad se describen en términos de software final y rendimiento del sistema, práctica de ingeniería de sonido requiere que los productos de trabajo intermedios sean relevantes para evaluar la calidad en todo el software proceso de ingeniería

1.4. Mejora de calidad de software

[3 *, c1s4] [9 *, c24] [10 *, c11s2.4]

Se puede mejorar la calidad de los productos de software. a través de procesos preventivos o una iteración tive proceso de mejora continua, que requiere control de gestión, coordinación y retroalimentación de muchos procesos concurrentes: (1) los procesos del ciclo de vida del software, (2) el proceso de detección, eliminación y prevención de fallas / defectos ción, y (3) el proceso de mejora de la calidad.

La teoría y los conceptos detrás de la calidad Mejora de la productividad, como construir en calidad a través de la prevención y detección temprana de defectos, mejora continua y parte interesada enfoque: son pertinentes para la ingeniería de software. Estos conceptos se basan en el trabajo de expertos. en calidad que han declarado que la calidad de un El producto está directamente relacionado con la calidad de proceso utilizado para crearlo. Enfoques como el Ciclo de mejora de Deming de Plan-Do-Check-Ley (PDCA), entrega evolutiva, kaizen y despliegue de funciones de calidad (QFD) ofrece tecnología niques para especificar objetivos de calidad y determinar si se cumplen La ingeniería de software IDEAL del Instituto es otro método [7 *]. Qual- su gestión ahora es reconocida por *SWE-La guía BOK* como disciplina importante.

El patrocinio de la gerencia apoya el proceso y evaluaciones de productos y los resultados resultantes. Luego se desarrolla un programa de mejora Identificar acciones detalladas y mejoras proyectos que se abordarán en un marco de tiempo factible. El apoyo administrativo implica que cada mejora El proyecto tiene suficientes recursos para lograr el objetivo definido para ello. El patrocinio de la gerencia es solicitado con frecuencia mediante la implementación proactiva actividades de comunicación.

1.5. Seguridad de software

[9 *, c11s3]

Los sistemas críticos para la seguridad son aquellos en los que un sistema del fracaso tem podría dañar la vida humana, otras vidas cosas, estructuras físicas o el medio ambiente. El software en estos sistemas es crítico para la seguridad. Cada vez hay más aplicaciones de software crítico para la seguridad en un número creciente de industrias. Ejemplos de sistemas con seguridad el software crítico incluye sistemas de transporte público, plantas de fabricación de productos químicos y médicos dispositivos. La falla del software en estos sistemas podría tener efectos catastróficos. Hay indus- pruebe estándares, como DO-178C [11], y emerg- ing procesos, herramientas y técnicas para el desarrollo Software de seguridad crítico. La intención de estos estándares, herramientas y técnicas es reducir el riesgo de inyectar fallas en el software y por lo tanto Mejorar la fiabilidad del software.

El software crítico para la seguridad se puede clasificar como directa o indirecta Directo es ese software incrustado Ded en un sistema crítico de seguridad, como el vuelo computadora de control de una aeronave. Indirecta incluye aplicaciones de software utilizadas para desarrollar seguridad software crítico El software indirecto está incluido en entornos de ingeniería de software y software entornos de prueba

Page 178

Tres técnicas complementarias para la reducción El riesgo de fracaso es evitar, detectar y eliminación, y limitación de daños. Estas las técnicas impactan en los requisitos funcionales del software- mentos, requisitos de rendimiento del software y procesos de desarrollo. Niveles crecientes de riesgo implican niveles crecientes de garantía de calidad del software. Técnicas de control y control tales como inspecciones. Los niveles de riesgo más altos pueden requerir más minuciosos inspecciones de requisitos, diseño y código o el uso de técnicas analíticas más formales. Otra técnica de gestión y control. El riesgo de software ling está construyendo casos de garantía. El caso de aseguramiento es un artefacto razonado y auditable cumplir con los estándares establecidos para el proyecto (incluidos requisitos, restricciones, diseños, contratos y planes). SQC evalúa intermedios ación productos, así como los productos finales.

La cuarta categoría de SQM que trata de mejorar ment tiene varios nombres dentro de la industria del software intente, incluyendo SPI, mejora de la calidad del software, y acciones correctivas y preventivas de software. los Las actividades en esta categoría buscan mejorar el proceso efectividad, eficiencia y otras características tics con el objetivo final de mejorar el software calidad. Aunque SPI podría incluirse en cualquiera de las primeras tres categorías, un número creciente de organizaciones organizan SPI en una categoría separada

creado para respaldar la afirmación de que su reclamo o reclamos están satisfechos. Contiene lo siguiente y sus relaciones: una o más afirmaciones sobre propiedades; argumentos que vinculan lógicamente el evidence y cualquier suposición a los reclamos; y un cuerpo de evidencia y suposiciones que apoyan estos argumentos [12].

2. Procesos de gestión de calidad de software

La gestión de la calidad del software es la colección de todos los procesos que aseguran que los productos de software, servicios e implementaciones de procesos de ciclo de vida cumplir con los objetivos de calidad del software organizacional y lograr la satisfacción de las partes interesadas [13, 14]. SQM define procesos, propietarios de procesos, requerimentos para los procesos, medidas de la procesos y sus resultados, y canales de retroalimentación nels a lo largo de todo el ciclo de vida del software.

SQM comprende cuatro subcategorías: software planificación de calidad, garantía de calidad de software (SQA), control de calidad del software (SQC) y software-mejora de procesos de software (SPI). Software cualificado La planificación de la actividad incluye determinar qué calidad se utilizarán estándares que definan una calidad específica metas y estimar el esfuerzo y el cronograma de actividades de calidad de software. En algunos casos, suave la planificación de la calidad de las mercancías también incluye definir Procesos de calidad de software a utilizar. Activ SQA Las entidades definen y evalúan la adecuación del software procesos para proporcionar evidencia que establezca confianza en que los procesos de software son apropiados priate para y producir productos de software de suit-calidad capaz para los fines previstos [5]. SQC las actividades examinan artefactos específicos del proyecto (documents y ejecutables) para determinar si

egory que puede abarcar muchos proyectos (ver el Proceso de Ingeniería de Software KA).

Los procesos de calidad del software consisten en tareas y técnicas para indicar cómo los planes de software (por ejemplo, gestión de software, desarrollo, calidad-gestión de ity o gestión de configuración planes) se están implementando y qué tan bien los productos intermedios y finales están cumpliendo sus requisitos especificados Resultados de estas tareas se ensamblan en informes para la gestión antes Se toman medidas correctivas. La administración de un proceso de SQM tiene la tarea de garantizar que el Los resultados de estos informes son precisos. La gestión de riesgos también puede jugar un papel importante papel en la entrega de software de calidad. Incorporando análisis disciplinado de riesgos y tecnología de gestión niques en los procesos del ciclo de vida del software pueden ayudar a mejorar la calidad del producto (ver el Software Ingeniería de Gestión KA para material sobre gestión de riesgos).

2.1. Aseguramiento de la calidad del software [7 *, c4 – c6, c11, c12, c26–27]

Para sofocar un malentendido generalizado, suave-El aseguramiento de la calidad de las mercancías no es una prueba. software El aseguramiento de la calidad (SQA) es un conjunto de actividades que definir y evaluar la idoneidad del software pro cesa proporcionar evidencia que establezca confidenciales dence que los procesos de software son apropiados y producir productos de software de calidad adecuada ity para los fines previstos. Un atributo clave de SQA es la objetividad de la función SQA con respeto al proyecto. La función SQA puede (también ser organizacionalmente independiente del proyecto ect; es decir, libre de aspectos técnicos, gerenciales y

presiones financieras del proyecto [5]. SQA tiene dos aspectos: aseguramiento del producto y aseguramiento del proceso ance, que se explican en la sección 2.3.

El plan de calidad del software (en alguna industria sectores se denomina garantía de calidad del software plan) define las actividades y tareas empleadas para garantizar que el software desarrollado para un determinado El producto satisface los requisitos establecidos del proyecto. necesidades y necesidades del usuario dentro del costo del proyecto y programar restricciones y es acorde con riesgos del proyecto. El SQAP primero asegura esa calidad los objetivos están claramente definidos y entendidos.

Las actividades y tareas de calidad del plan SQA son especificado con sus costos, recursos necesarios, objetivos y cronograma en relación con objetivos en la gestión de ingeniería de software ment, desarrollo de software y mantenimiento de software planes de tenencia. El plan SQA debe ser consistente tienda con la gestión de configuración de software plan (consulte la Gestión de configuración de software-ment KA). El plan SQA identifica documentos, estándares, prácticas y convenciones que rigen el proyecto y cómo se verifican estos elementos y monitoreado para asegurar la adecuación y el cumplimiento. El plan SQA también identifica medidas; estadístico tecnicas; procedimientos para informar problemas y acción correctiva; recursos como herramientas, tecnología niques y metodologías; seguridad física

ciclo vital. Esta evaluación demuestra si los requisitos son correctos, com Completo, preciso, consistente y comprobable. Los procesos V&V determinan si los productos de desarrollo de una determinada actividad se ajusta a los requisitos de ese actividad y si el producto satisface su uso previsto y las necesidades del usuario.

La verificación es un intento de garantizar que El producto está construido correctamente, en el sentido de que el Los productos de salida de una actividad cumplen con las especificaciones cationes impuestos a ellos en actividades anteriores. La validación es un intento de garantizar que el derecho el producto está construido, es decir, el producto cumple con su propósito específico previsto. Tanto la verificación proceso y el proceso de validación comienzan temprano en la fase de desarrollo o mantenimiento. Ellos proporcionar un examen de las características clave del producto en relación tanto con la predefinición inmediata del producto evaluador y las especificaciones a cumplir.

El propósito de planificar V&V es asegurar que cada recurso, rol y responsabilidad es claramente asignado Los documentos del plan V&V resultantes describir los diversos recursos y sus roles y actividades, así como las técnicas y herramientas para ser utilizado.Una comprensión de los diferentes propósitos de cada actividad de V&V ayuda en la planificación cuidadosa de

medios de comunicación; formación; e informes y documentación de SQA
Mentación. Además, el plan SQA aborda
las actividades de aseguramiento de la calidad del software documentadas
otro tipo de actividad descrita en el software
planes, como la adquisición de software de proveedores
para el proyecto, software comercial listo para usar
(COTS) instalación y servicio después de la entrega de
El software. También puede contener criterios de aceptación.
ria, así como actividades de informes y gestión
vínculos que son críticos para la calidad del software.

2.2. Verificación validación

[9 *, c2s2.3, c8, c15s1.1, c21s3.3]

Como se indica en [15],

> El propósito de V&V es ayudar al desarrollo
> La organización de opciones crea calidad en el
> sistema durante el ciclo de vida. V&V pro-
> los procesos proporcionan una evaluación objetiva
> de productos y procesos en todo el

las técnicas SQA recursos necesarios para cumplir con sus
propósitos El plan también aborda la gestión
documentación, comunicación, políticas y procedimientos de
las actividades de V&V y su interacción, así como
informes de defectos y requisitos de documentación.

2.3. Revisiones y auditorías

[9 *, c24s3] [16 *]

Las revisiones y los procesos de auditoría están ampliamente definidos
como estático, lo que significa que no hay programas de software o
se ejecutan modelos: examen del software
artefactos de ingeniería con respecto a estándares que
han sido establecidos por la organización o el proyecto
ect para esos artefactos. Diferentes tipos de revisiones
y las auditorías se distinguen por su propósito, nivel
els de independencia, herramientas y técnicas, roles,
y por el tema de la actividad. Aseguramiento de producto
Las auditorías de garantía de procesos y procesos suelen ser
conducido por el aseguramiento de la calidad del software (SQA)
personal independiente del desarrollo

Page 180

equipos Las revisiones de la gerencia son conducidas por
gestión organizacional o de proyectos. El ingenio
El personal directivo realiza revisiones técnicas.

- Las revisiones de la gerencia evalúan el proyecto real
  resultados con respecto a los planes.
- Revisiones técnicas (incluidas inspecciones,
  tutorial y verificación de escritorio) examinar
  productos de trabajo de ingeniería.
- Auditorías de aseguramiento de procesos. Proceso de SQA
  Las actividades de aseguramiento aseguran que el
  procesos utilizados para desarrollar, instalar, operar,
  y mantener el software conforme a los contratos,
  cumplir con las leyes, reglas y
  regulaciones y son adecuadas, eficientes y
  eficaz para su finalidad prevista [5].
- Auditorías de aseguramiento de producto. Producto SQA
  las actividades de aseguramiento se aseguran de proporcionar
  evidencia de que los productos de software y relacionados
  la documentación se identifica y cumple
  con contratos; y asegurar que no
  se identifican y abordan los mances [5].

2.3.1. Revisiones de la gerencia

Como se indica en [16 *],

> El propósito de una revisión administrativa es
> monitorear el progreso, determinar el estado de
> planes y horarios, y evaluar el efecto
> Actividad de los procesos de gestión, herramientas
> y tecnicas. La gerencia revisa las comunicaciones
> comparar los resultados reales del proyecto con los planes para
> determinar el estado de proyectos o mantenimiento
> esfuerzos financieros. Los principales parámetros del hombre
> las revisiones de gestión son el costo del proyecto, el cronograma,
> alcance y calidad. Revisiones de la gerencia
> evaluar decisiones sobre acciones correctivas,
> cambios en la asignación de recursos, o
> cambios en el alcance del proyecto.

Los aportes a las revisiones de la gerencia pueden incluir

2.3.2. Revisiones técnicas

Como se indica en [16 *],

> El propósito de una revisión técnica es
> evaluar un producto de software por un equipo de
> personal calificado para determinar su demanda
> capacidad para su uso previsto e identificar
> discrepancias de especificaciones y
> normas Proporciona gestión con
> evidencia para confirmar el estado técnico de
> el proyecto.

Aunque cualquier producto de trabajo puede ser revisado,
Las revisiones técnicas se realizan en el principal
ingeniería de software productos de trabajo de software
requisitos y diseño de software.
Propósito, roles, actividades y lo más importante
nivel de formalidad distingue diferentes tipos
de revisiones técnicas. Las inspecciones son las más
mal, tutoriales menos y opiniones par o escritorio
los cheques son los menos formales.

Ejemplos de roles específicos incluyen una decisión
fabricante (es decir, líder de software), un líder de revisión, un
grabadora y verificadores (miembros del personal técnico
quienes examinan los productos de trabajo). Los comentarios son
También se distingue por si las reuniones (cara a
cara o electrónica) están incluidos en el proceso. En
algunos métodos de revisión verificadores solitariamente
productos de trabajo ine y enviar sus resultados a
Un coordinador. En otros métodos, los verificadores funcionan
cooperativamente en reuniones. Una revisión técnica
puede requerir que las entradas obligatorias estén en su lugar en
para proceder:

- Declaración de objetivos
- Producto de software específico
- Plan de gestión del proyecto específico.
- Lista de problemas asociados con este producto
- Procedimiento de revisión técnica.

El equipo sigue el programa de revisión documentado.

informes de auditoría, informes de progreso, informes de V&V y planes de muchos tipos, incluida la gestión de riesgos, gestión de proyectos, configuración de software gestión, seguridad de software y evaluación de riesgos ment, entre otros. (Consulte el Software Engine- Gestión de Neering y la Configuración de Software Gestión de raciones KAs para material relacionado.)

&Luego La revisión técnica se completa una vez todas las actividades enumeradas en el examen tienen ha sido completado

Las revisiones técnicas del código fuente pueden incluir un gran variedad de inquietudes, como el análisis de algo ritmos, utilización de recursos informáticos críticos, adherencia a los estándares de codificación, estructura y

Página 181

organización del código de comprobabilidad y seguridad consideraciones críticas

Tenga en cuenta que las revisiones técnicas del código fuente y los modelos de diseño como UML también se denominan estáticos análisis (ver tema 3, Consideraciones prácticas).

2.3.3. Inspecciones

"El propósito de una inspección es detectar y identificar anomalías de productos de software "[16 *]. Algunos diferenciadores importantes de las inspecciones como en comparación con otros tipos de revisiones técnicas son estas:

1. Reglas. Las inspecciones se basan en el examen. un producto de trabajo con respecto a un conjunto definido de criterios especificados por la organización. Conjuntos de reglas se pueden definir para diferentes tipos de productos de trabajo (por ejemplo, reglas para requisitos, descripciones de arquitectura, código fuente).
2. Muestreo. Más bien ese intento de examinar cada palabra y figura en un documento, el El proceso de inspección permite a los inspectores evaluar comió subconjuntos definidos (muestras) del documento Menciones bajo revisión.
3. Compañero. Individuos que ocupan puestos directivos iones sobre los miembros del equipo de inspección No participe en la inspección. Esto es una distinción clave entre revisión por pares y revisión de gestión.
4. Led. Un moderador imparcial que está capacitado. en técnicas de inspección conduce inspección reuniones
5. Reunión. El proceso de inspección incluye reuniones (cara a cara o electrónicas) conducido por un moderador de acuerdo con un formal procedimiento en el cual el equipo de inspección integra Bers informan las anomalías que han encontrado y otras cuestiones

Las inspecciones de software siempre involucran al autor de un producto intermedio o final; otros comentarios tal vez no. Las inspecciones también incluyen una inspección líder, una grabadora, un lector y unos pocos (de dos a cinco) Damas (inspectores). Los miembros de una inspección el equipo de la nación puede poseer experiencia diferente, como experiencia en el dominio, experiencia en métodos de diseño de software tise, o experiencia en lenguaje de programación. Inspec- Por lo general, las acciones se realizan en una relativamente pequeña sección del producto a la vez (muestras). Cada miembro del equipo examina el producto de software. artículos otras entradas de revisión antes de la revisión técnica, quizás aplicando una tecnología analítica nique (ver sección 3.3.3) a una pequeña sección de el producto o para todo el producto con un enfoque en un solo aspecto, por ejemplo, interfaces. Durante el inspección, el moderador dirige la sesión y verifica que todos se hayan preparado para el inspección y realiza la sesión. La inspección registrador de documentos documenta anomalías encontradas. Un conjunto de reglas, con criterios y preguntas relacionadas con los temas de interés, es una herramienta común utilizada en inspecciones La lista resultante a menudo clasifica el anomalías (ver sección 3.2, Caracterización de defectos- ción) y se revisa para verificar su integridad y precisión picante por el equipo. La decisión de salida de inspección corresponde a una de las siguientes opciones:

1. Acepte sin reelaboración o, a lo sumo, con modificaciones menores
2. Aceptar con verificación de retrabajo
3. Reinspectar.

2.3.4. Tutoriales

Como se indica en [16 *],

> El propósito de un recorrido sistemático es evaluar un producto de software. Un paseo- a través se puede llevar a cabo con el propósito de educar a una audiencia con respecto a un producto de cerámica.

Los tutoriales se distinguen de la inspección iones La principal diferencia es que el autor presEntra el producto de trabajo a los otros participantes en una reunión (cara a cara o electrónica). A diferencia de un inspección, los participantes de la reunión pueden no tener necesariamente visto el material antes de la reunión En g. Las reuniones pueden realizarse menos por- mally El autor toma el papel de explicar y mostrando el material a los participantes y solicita realimentación. Al igual que las inspecciones, los recorridos pueden ser realizado en cualquier tipo de producto de trabajo incluyendo plan de proyecto, requisitos, diseño, código fuente, informes de prueba.

Page 182

2.3.5. Aseguramiento de proceso y aseguramiento de producto auditorias

Como se indica en [16 *],

> El propósito de una auditoría de software es pro vide una evaluación independiente de la con- formacion de productos de software y pro- ceses a las regulaciones, estándares aplicables, pautas, planes y procedimientos.

Las auditorías de aseguramiento de procesos determinan la adecuación de planes, horarios y requisitos para lograr objetivos del proyecto [5]. La auditoría es formalmente actividad organizada con participantes que tienen roles específicos, tales como auditor principal, otro auditor, un grabadora o un iniciador, e incluyendo un representante representación de la organización auditada. Identificaciones de auditorias instancias de incumplimiento y producir un informe exigir al equipo que tome medidas correctivas.

Si bien puede haber muchos nombres formales para revisiones y auditorías, como las identificadas en el estándar [16 *], lo importante es que puede ocurrir en casi cualquier producto en cualquier etapa de El proceso de desarrollo o mantenimiento.

3. Consideraciones prácticas

3.1. Requisitos de calidad del software

[9 *, c11s1] [18 *, c12] [17 *, c15s3.2.2, c15s3.3.1, c16s9.10]

3.1.1. Factores de influencia

Varios factores influyen en la planificación, gestión, y selección de actividades y técnicas de SQM, incluso

- el dominio del sistema en el que el software la vajilla reside; las funciones del sistema podrían ser seguridad crítica, misión crítica, negocios- crítico, crítico de seguridad
- el entorno físico en el que el sistema de artículos reside
- sistema y software funcional (lo que el sistema) y calidad (qué tan bien el sistema tem realiza sus funciones) requisitos
- el comercial (externo) o estándar (inter- nal) componentes que se utilizarán en el sistema
- los estándares específicos de ingeniería de software aplicable
- los métodos y herramientas de software que se utilizarán para desarrollo y mantenimiento y por calidad Evaluación y mejora de la itidad
- el presupuesto, el personal, la organización del proyecto, los planes, y programación de todos los procesos
- los usuarios previstos y el uso del sistema
- El nivel de integridad del sistema.

La información sobre estos factores influye en cómo las actividades de SQM están organizados y documentados. ¿Cómo se seleccionan las actividades específicas de SQM? qué recursos se necesitan y cuál de esos los recursos imponen límites a los esfuerzos.

3.1.2. Confianza

En casos donde la falla del sistema puede tener extremadamente consecuencias graves, confiabilidad general (difícil- Ware, software y humanos u operacionales) es el Requisito de calidad principal además de básico funcionalidad Este es el caso de lo siguiente razones: las fallas del sistema afectan una gran cantidad de gente; los usuarios a menudo rechazan sistemas que no son responsable, inseguro o inseguro; costos de falla del sistema puede ser enorme; y sistemas poco confiables puede causar pérdida de información. Sistema y soft- la confiabilidad de las mercancías incluye tales características como disponibilidad, fiabilidad, seguridad y protección. Al desarrollar software confiable, herramientas y Se pueden aplicar técnicas para reducir el riesgo de inyectando fallas en los entregables intermedios o el producto de software final. Verificación, validación ción y procesos de prueba, técnicas, métodos, y las herramientas identifican fallas que afectan la confiabilidad tan pronto como sea posible en el ciclo de vida. Adición- Además, los mecanismos pueden necesitar estar en su lugar en el software para proteger contra ataques externos y para tolerar fallas.

3.1.3. Niveles de integridad del software

La definición de los niveles de integridad es un método de riesgo. administración.

> Los niveles de integridad del software son una gama de valores que representan la complejidad del software, criticidad, riesgo, nivel de seguridad, nivel de seguridad,

Página 183

rendimiento deseado, confiabilidad u otro características únicas del proyecto que definen La importancia del software para el usuario y adquirente. Las características utilizadas para determinar el nivel de integridad del software varía dependiendo de la aplicación prevista y uso del sistema. El software es parte de el sistema, y su nivel de integridad debe ser determinado como parte de ese sistema.

Los tipos específicos de problemas deben agruparse para Identificar tendencias a lo largo del tiempo. El punto es establecer una taxonomía defectuosa que es significativa para la organización nización y para ingenieros de software.

Las actividades de control de calidad del software descubren información mación en todas las etapas del desarrollo de software y mantenimiento. En algunos casos, la palabra *defecto* es sobrecargado para referirse a diferentes tipos de anomalías. Sin embargo, diferentes culturas de ingeniería y estándares los padres pueden usar significados algo diferentes para

Los niveles de integridad de software asignados pueden cambiar a medida que el software evoluciona. Diseño, codificación, características procesales y tecnológicas implementadas en el sistema o software puede subir o bajar el niveles de integridad de software asignados. El software niveles de integridad establecidos para el resultado de un proyecto de acuerdos entre el adquirente, el proveedor, desarrollador y autoridades de aseguramiento independientes. Un esquema de nivel de integridad de software es una herramienta utilizada para determinar los niveles de integridad del software. [5]

Como se señaló en [17 *], "los niveles de integridad pueden ser aplicado durante el desarrollo para asignar más esfuerzos de verificación y validación para alta integ- componentes de la ciudad ".

3.2. Caracterización de defectos

[3 *, c3s3, c8s8, c10s2]

Evaluación de la calidad del software (es decir, calidad del software control) técnicas encuentran defectos, fallas y fallas ures Caracterizar estas técnicas conduce a una comprensión del producto, facilita la corrección acciones al proceso o al producto, e informa gerencia y otras partes interesadas del personal tus del proceso o producto. Muchas taxonomías existen y, aunque se han hecho intentos para ganar consenso, la literatura indica que hay bastantes en uso. La caracterización de defectos también es utilizado en auditorías y revisiones, con el líder de la revisión a menudo presentando una lista de problemas proporcionados por los miembros para su consideración en una reunión de revisión.

A medida que evolucionan los nuevos métodos de diseño y lenguajes, junto con los avances en tecnología de software en general aparecen nuevas clases de defectos y un gran Se requiere mucho esfuerzo para interpretar previamente clases definidas Al rastrear defectos, el el ingeniero de mercancías está interesado no solo en el número de defectos pero también los tipos. Información sola, sin alguna clasificación, puede no ser suficiente para identificar las causas subyacentes de los defectos. estos términos. La variedad de términos suscita este segundo ción para proporcionar un conjunto de definiciones ampliamente utilizado [19]:

- *Error computacional* : "la diferencia entre una calculada, observada o medida valor o condición y el verdadero, especificado o valor o condición teóricamente correcta ".
- *Error* : "Una acción humana que produce un resultado incorrecto ". Un deslizamiento o error que un per- hijo hace. También se llama error humano.
- *Defecto* : una "imperfección o deficiencia en un producto de trabajo donde lo hace ese producto de trabajo no cumple con sus requisitos o especificaciones y necesita ser reparado o reemplazado ". Un defecto es causado por una persona que comete un error.
- *Falla* : un defecto en el código fuente. Un "incorrecto paso, proceso o definición de datos en la computadora programa ". La codificación de un error humano en código fuente. Falla es el nombre formal de un error .
- *Falla* : un "evento en el que un sistema o sistema componente tem no realiza un requerido funcionar dentro de los límites especificados. " producido cuando se encuentra una falla por el procesador bajo condiciones especificadas.

Usando estas definiciones, tres soft- ampliamente utilizados las mediciones de calidad de la cerámica son densidad de defectos (número de defectos por unidad de tamaño de documentos), densidad de fallas (número de fallas por 1K líneas de código) e intensidad de falla (fallas por hora de uso y por lenguaje de prueba). Se construyen modelos de confiabilidad de datos de falla recopilados durante la prueba de software ing o desde software en servicio y por lo tanto puede ser usado para estimar la probabilidad de fallas futuras y para ayudar en las decisiones sobre cuándo detener la prueba.

Una acción probable resultante de la búsqueda de SQM ings es eliminar los defectos del producto bajo examen (p. ej., buscar y corregir errores, crear nueva construcción). Otras actividades intentan eliminar

Page 184

Calidad de software 10-11

las causas de los defectos, por ejemplo, la causa raíz análisis (RCA). Las actividades de RCA incluyen analizar y resumiendo los hallazgos para encontrar las causas raíz y usando técnicas de medición para mejorar el producto y el proceso, así como rastrear defectos y su eliminación. La mejora de procesos se discute principalmente en el Software Engineer- ing Process KA, siendo el proceso SQM un Fuente de información.

Datos sobre deficiencias y defectos encontrados por las técnicas de control de calidad del software pueden perderse a menos que estén registrados. Para algunas técnicas (por ejemplo, revisiones técnicas, auditorías, inspecciones), grabadores están presentes para establecer dicha información ción, junto con cuestiones y decisiones. Cuando auto- se utilizan herramientas acopladas (ver tema 4, Software Qual- ity Tools), la salida de la herramienta puede proporcionar el defecto información. Se proporcionan informes sobre defectos. a la gestión de la organización.

3.3. Técnicas de gestión de calidad de software

[7 *, c7s3] [8 *, c17] [9 *, c12s5, c15s1, p417] [dieciséis *]

También Métodos formales en el ingeniero de software ing Modelos y métodos KA.)

3.3.2. Técnicas Dinámicas

Las técnicas dinámicas implican ejecutar el soft- código de mercancías Diferentes tipos de técnicas dinámicas. se realizan durante todo el desarrollo y mantenimiento de software. En general, estos son técnicas de prueba, pero técnicas como simu- lación Se puede considerar el análisis del modelo y la relación. dinámico (ver los Modelos de Ingeniería de Software y Métodos KA). La lectura de código se considera un técnica estática, pero con experiencia en software de ingeniería los neers pueden ejecutar el código mientras leen uso. La lectura de códigos puede utilizar técnicas dinámicas. Esta discrepancia en la categorización indica que personas con diferentes roles y experiencia en el la organización puede considerar y aplicar estas tecnologías niques de manera diferente.

Diferentes grupos pueden realizar pruebas durante desarrollo de software, incluidos grupos independientes ndiente del equipo de desarrollo. El software Testing KA está dedicado por completo a este tema.

Las técnicas de control de calidad del software se pueden utilizar
egorizado de muchas maneras, pero de manera directa
El enfoque utiliza solo dos categorías: estática y
dinámica. Las técnicas dinámicas implican ejecutar
El software; las técnicas estáticas implican analizar
documentos y código fuente pero no ejecutando el
software.

3.3.1 Técnicas Estáticas

Las técnicas estáticas examinan software docu-
ción (incluidos requisitos, especificación de interfaz
fuentes, diseños y modelos) y fuente de software
código sin ejecutar el código. Hay muchos
herramientas y técnicas para examinar estáticamente soft-
productos de trabajo de cerámica (ver sección 2.3.2). Además
ción, herramientas que analizan el flujo de control del código fuente
y la búsqueda de código muerto se considera
herramientas de análisis estático porque no implican
ejecutando el código del software.

Otros tipos de tecnología analítica más formales
Las niques se conocen como métodos formales . Son
notablemente utilizado para verificar los requisitos de software y
diseños Se han utilizado principalmente en el veri
ficación de partes cruciales de sistemas críticos, como
como requisitos específicos de seguridad y protección. (Ver

3.3.3 Pruebas

Dos tipos de pruebas pueden caer bajo V&V porque
de su responsabilidad por la calidad del mate-
riales utilizados en el proyecto:

- Evaluación y pruebas de herramientas para ser utilizadas en el proyecto
- Pruebas de conformidad (o revisión de conformidad pruebas de mance) de componentes y productos COTS ucts que se utilizarán en el producto.

A veces un independiente (de terceros o
IV&V) la organización puede tener la tarea de realizar
probar o monitorear el proceso de prueba V&V puede
se ha llevado a cabo para evaluar la prueba en sí: ade-
calidad de planes, procesos y procedimientos, y
adecuación y precisión de los resultados.

El tercero no es el desarrollador, ni es
asociado con el desarrollo del producto.
En cambio, el tercero es un facil independiente
ity, generalmente acreditado por algún cuerpo de autoridad.
Su propósito es probar la conformidad de un producto
a un conjunto específico de requisitos (ver el Software
Prueba de KA).

Page 185

3.4. Medición de calidad de software
[3 *, c4] [8 *, c17] [9 *, p90]

Las mediciones de calidad del software se utilizan para
Apoyar la toma de decisiones. Con el aumento
sofisticación de software, cuestiones de calidad
ir más allá de si el software funciona o no
qué tan bien logra objetivos de calidad medibles.

Decisiones respaldadas por medidas de calidad de software
aseguramiento incluye determinar niveles de software
calidad (especialmente porque los modelos de software
La calidad del producto incluye medidas para determinar
el grado en que el producto de software alcanza
objetivos de calidad); preguntas gerenciales sobre esfuerzo,
costo y horario; determinar cuándo detener la prueba
ing y lanzar un producto (ver Terminación en
sección 5.1, Consideraciones prácticas, en Soft-
prueba de mercancías KA); y determinar la eficacia
de esfuerzos de mejora de procesos.

El costo de los procesos de SQM es un problema
planteada de manera que decide cómo un proyecto o un programa
grupo de desarrollo y mantenimiento de mercancías debe
organizar. A menudo, los modelos genéricos de costo son
utilizado, que se basan en cuándo se encuentra un defecto
y cuánto esfuerzo lleva arreglar el defecto
tive para encontrar el defecto antes en el desarrollo
proceso de ment. Datos de medición de calidad de software
recopilado internamente puede dar una mejor idea de
costo dentro de este proyecto u organización.

Mientras que los datos de medición de calidad del software
puede ser útil en sí mismo (p. ej., el número de defec-
requisitos tive o la proporción de defectos
requisitos), tecnología matemática y gráfica
las niques se pueden aplicar para ayudar en la interpretación
de las medidas (ver los Fundamentos de Ingeniería
KA). Estas técnicas incluyen

- estadística descriptiva basada (por ejemplo, Pareto

áreas problemáticas del producto de software bajo
examen. Los cuadros y gráficos resultantes
son ayudas de visualización, que la decisión toma
Los usuarios pueden utilizar para enfocar recursos y realizar pro- gramas
cess mejoras donde parecen ser más
necesario. Los resultados del análisis de tendencias pueden indicar
que se está cumpliendo un cronograma, como en las pruebas, o
que ciertas clases de fallas pueden volverse más
probable que ocurra a menos que alguna acción correctiva sea
tomado en desarrollo. Las tecnicas predictivas
ayudar a estimar el esfuerzo y el horario de las pruebas
y en predecir fallas. Más discusión sobre
la medida en general aparece en el Software
Ingeniería de Procesos y Software
Gestión de KAs. Información más específica sobre
la medida de prueba se presenta en el Software
Prueba de KA.

La medición de la calidad del software incluye mea-
ocurrencias de defectos crecientes y aplicando estadísticas
métodos para comprender los tipos de defectos que
ocurren con mayor frecuencia. Esta información puede ser
utilizado por la mejora del proceso de software para disuadir
métodos de minería para prevenir, reducir o eliminar
su recurrencia También ayudan a comprender
tendencias, qué tan bien la tecnología de detección y contención
las niques están funcionando, y qué tan bien el desarrollo
Los procesos de mantenimiento y mantenimiento están progresando.

De estos métodos de medición, defecto
los perfiles se pueden desarrollar para una aplicación específica
dominio de la nación Luego, para el próximo proyecto de software
dentro de esa organización, los perfiles se pueden usar
para guiar los procesos de SQM, es decir, para gastar
el esfuerzo donde es más probable que ocurran problemas.
Del mismo modo, los puntos de referencia o los recuentos de defectos típicos de
ese dominio, puede servir como una ayuda para determinar
cuando el producto está listo para la entrega. Disco-
Sión sobre el uso de datos de SQM para mejorar el desarrollo
Los procesos de opciones y mantenimiento aparecen en el

análisis, gráficos de ejecución, diagramas de dispersión, control
distribución)

- pruebas estadísticas (p. Ej., La prueba binomial, quimioterapia)
  prueba al cuadrado)
- análisis de tendencias (p. Ej., Gráficos de control; ver
  La caja de *herramientas de calidad* en la lista de más
  lecturas)
- predicción (p. Ej., Modelos de confiabilidad).

Técnicas descriptivas basadas en estadísticas y
las pruebas a menudo proporcionan una instantánea de los más

Gestión de Ingeniería de Software y Software
Proceso de Ingeniería KAs.

4. Herramientas de calidad de software

Las herramientas de calidad de software incluyen estática y dinámica.
herramientas de análisis. Fuente de entrada de herramientas de análisis estático
codificar, realizar análisis sintáctico y semántico
sin ejecutar el código y presentar resultados a
usuarios. Hay una gran variedad en la profundidad, tor-
aspereza y alcance de las herramientas de análisis estático que

Page 186

se puede aplicar a artefactos, incluidos modelos, en
Además del código fuente. (Consulte el Con-
estructura, prueba de software y software principal
tenencia KAs para descripciones de análisis dinámico
herramientas.)

Las categorías de herramientas de análisis estático incluyen el
siguiendo:

- Herramientas que facilitan y automatizan parcialmente
  revisiones e inspecciones de documentos y
  código. Estas herramientas pueden enrutar el trabajo para diferentes
  participantes para automatizar parcialmente
  y controlar un proceso de revisión. Ellos permiten
  usuarios para ingresar defectos encontrados durante la inspección
  opciones y revisiones para su posterior eliminación.
- Algunas herramientas ayudan a las organizaciones a realizar
  análisis de riesgos de seguridad de las mercancías. Estas herramientas
  proporcionar, por ejemplo, soporte automatizado para fallas
  análisis de modo y efectos (FMEA) y falla
  Análisis de árboles (TLC).
- Herramientas que admiten el seguimiento de problemas de software.
  los lems prevén la entrada de anomalías
  ered durante las pruebas de software y posteriores
  análisis, disposición y resolución. Algunos
  las herramientas incluyen soporte para flujo de trabajo y para
  seguimiento del estado de la resolución del problema.
- Herramientas que analizan los datos capturados desde soft-
  entornos de ingeniería de software y software
  entornos de prueba de software y producen visual
  muestra de datos cuantificados en forma de
  gráficos, cuadros y tablas. Estas herramientas algunas
  veces incluyen la funcionalidad para realizar
  análisis estadístico sobre conjuntos de datos (para el propósito
  de discernir tendencias y hacer precedencia
  moldes). Algunas de estas herramientas presentan defectos
  y tasas de inyección de eliminación; densidades de defectos;
  rendimientos distribución de inyección de defectos y
  eliminación para cada una de las fases del ciclo de vida.

Page 187

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	02 un 20 K [3 *]	0 0 0 0 ott y col. 20 segundo [6 *]	04 alin 20 ol [7 *]	08 y 20 ath Dep d T aik an norte [8 *]	ile 2011 erv metro om S [9 *]	03 d 20 Olan V [10 *]	08 -20 . 1028 td IEEE S [dieciséis *]	06 oore 20 METRO [17 *]	03 iegers 20 W [18 *]
1. Software Calidad Fundamentos									
1.1. Software Ingenieria Cultura y Ética	c1s4	c2s3.5							
1.2. Valor y Costo de calidad			c17, c22						
1.3. Modelos y calidad Caracteristicas	c24s1		c2s4	c17					
1.4. Software Calidad Mejora	c1s4				c24	c11 s2.4			
1.5. Software La seguridad					c11s3				
2. Software Calidad administración Procesos									
2.1. Software Calidad Garantía			c4 – c6, c11, c26–27						
2.2. Verificación y validación					c2 s2.3, c8, c15 s1.1, c21 s3.3				
2.3. Comentarios y auditorías					c24s3		* *		

Page 188

02	0 0 0 0	04	08 y 20 ath	ile 2011	03	08 -20	06	03

	[3 *] un 20 K	[6 *] ott y col. 20 segundo	[7 *] alin 20 sol	[8 *] Dep d T aik an norte	[9 *] erv metro gm S	[10 *] d 20 Olan V	[12 *] td [13 *] ecisé is oore 20 METRO	[17 *] [18 *] iegers 20 W
					IEEE S			
3. Software Calidad práctica Consideraciones								
3.1. Software Calidad Requisitos					c11s1		c15 s3.2.2, c15 s3.3.1, c16 s9.10	c12
3.2. Defecto Caracterización	c3s3, c8s8, c10s2							
3.3. SQM Tecnicas			c7s3	c17	c12s5, c15s1, p417		* *	
3.4. Software Calidad Medición	c4			c17	p90			
4. Software Herramientas de calidad								

LECTURAS ADICIONALES

N. Leveson, *Safeware: seguridad del sistema y Ordenadores* [20].

Este libro describe la importancia del software. prácticas de seguridad y cómo pueden ser estas prácticas incorporado en proyectos de desarrollo de software.

T. Gilb, *Principios de Ingeniería de Software Gestión* [21].

Este es uno de los primeros libros sobre iterativo y Técnicas de desarrollo incremental. El evo El método define objetivos cuantificados, tiempo frecuente iteraciones en caja, medidas de progreso hacia objetivos y adaptación de planes basados en resultados actuales.

T. Gilb y D. Graham, *Inspección de software* [22]

KE Wiegers, *evaluaciones por pares en software: A Guía práctica* [23].

Este libro proporciona explicaciones claras y sucintas. de diferentes métodos de revisión por pares distinguidos por nivel de formalidad y efectividad. Pragmático orientación para implementar los métodos y cómo para seleccionar qué métodos son apropiados para Se proporcionan circunstancias.

NR Tague, *The Quality Toolbox* , 2ª ed., [24].

Proporciona una explicación práctica pragmática de un conjunto integral de métodos, herramientas y tecnología niques para resolver problemas de mejora de calidad lems Incluye los siete controles básicos de calidad. herramientas y muchos otros.

IEEE Std. Proyecto de norma P730-2013 para

Este libro presenta mediciones y estadísticas. muestreo cal para revisiones y defectos. Presenta técnicas que producen resultados cuantificados para Reducción de defectos, mejora de la productividad, seguimiento ing proyectos y creación de documentación.

Procesos de aseguramiento de la calidad del software [5].

Este borrador del estándar expande los procesos SQA identificado en IEEE / ISO / IEC 12207-2008. P730 establece estándares para iniciar, planificar, controlar y ejecutar la calidad del software procesos de aseguramiento de un desarrollo de software o proyecto de mantenimiento. Aprobación de este borrador estándar se espera en 2014.

Page 190

Referencias

[1] PB Crosby, *Quality Is Free* , McGraw-Hill, 1979

[2] W. Humphrey, *Gestión del software Proceso* , Addison-Wesley, 1989.

[3 *] SH Kan, *métricas y modelos en software Ingeniería de calidad* , 2ª ed., Addison-Wesley, 2002.

[4] *ISO / IEC 25010: 2011 Sistemas y software Ingeniería: sistemas y software Requisitos de calidad y evaluación (SQuaRE): calidad de sistemas y software Modelos* , ISO / IEC, 2011.

[5] *IEEE P730 ™ / D8 Draft Standard para Procesos de aseguramiento de la calidad del software* , IEEE, 2012.

[6 *] F. Bott et al., *Problemas profesionales en Ingeniería de Software* , 3ª ed., Taylor & Francis, 2000.

[7 *] D. Galin, *Garantía de calidad del software: De la teoría a la implementación* , Pearson Education Limited, 2004.

[8 *] S. Naik y P. Tripathy, *Pruebas de software y garantía de calidad: teoría y Práctica* , Wiley-Spektrum, 2008.

[9 *] P. Clements et al., *Software de documentación*

[13] *IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Norma para sistemas y Ingeniería de software: ciclo de vida del software Procesos* , IEEE, 2008.

[14] *ISO 9000: Gestión de calidad 2005 Sistemas: fundamentos y vocabulario* , ISO, 2005.

[15] *IEEE Std. Norma 1012-2012 para el sistema y verificación y validación de software* , IEEE, 2012.

[16 *] *IEEE Std. 1028-2008, Revisiones de software y Auditorías* , IEEE, 2008.

[17 *] JW Moore, *La hoja de ruta hacia el software Ingeniería: una guía basada en estándares* , Wiley-IEEE Computer Society Press, 2006.

[18 *] KE Wiegers, *Requisitos de software* , 2do ed., Microsoft Press, 2003.

[19] *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010.

[20] N. Leveson, *Safeware: seguridad del sistema y Computadoras* , Addison-Wesley Professional, 1995

[21] T. Gilb, *Principios de Ingeniería de Software Gerencia* , Addison-Wesley Professional,

Arquitecturas: Vistas y más allá , 2ª ed., Pearson Education, 2010.

[10 *] G. Voland, *Ingeniería por diseño* , 2do ed., Prentice Hall, 2003.

[11] *RTCA DO-178C, Consideraciones de software en sistemas y equipos aerotransportados Certificación* , Comisión Técnica de Radio para Aeronáutica, 2011.

[12] *IEEE Std. 15026.1-2011 Estándar de uso de prueba Adopción de ISO / IEC TR 15026-1: 2010 Ingeniería de Sistemas y Software Sistemas y Software Assurance — Parte 1: Conceptos y vocabulario* , IEEE, 2011.

1988.

[22] T. Gilb y D. Graham, *Software Inspección* , Addison-Wesley Professional, 1993.

[23] K. Wiegers, *evaluaciones por pares en software: A Guía práctica* , Addison-Wesley Profesional, 2001.

[24] NR Tague, *The Quality Toolbox* , 2ª ed., ASQ Quality Press, 2010.

CAPÍTULO 11

INGENIERÍA DE SOFTWARE PRACTICA PROFESIONAL

SIGLAS

ACM	Asociación para la informática Maquinaria
BCS	Sociedad Británica de Computación
CSDA	Desarrollo de software certificado Asociar
PCSD	Desarrollo de software certificado Profesional
IEC	Electrotécnica Internacional Comisión
IEEE CS	IEEE Computer Society
IFIP	Internacional. Federación para Procesamiento de información
IP	Propiedad intelectual
YO ASI	Organización internacional para Normalización
NDA	Acuerdo de no divulgación
OMPI	Propiedad intelectual mundial Organización
OMC	Organización de Comercio Mundial

INTRODUCCIÓN

La práctica profesional de ingeniería de software El área de conocimiento (KA) se refiere a la conocimiento, habilidades y actitudes que el software los ingenieros deben poseer para practicar ingeniería de software neering en un profesional, responsable y eti manera cal. Debido a la aplicación generalizada iones de productos de software en redes sociales y personales vida, la calidad de los productos de software puede tener profundo impacto en nuestro bienestar personal y armonía social. Los ingenieros de software deben software con características conocidas y confiabilidad ity. Este requisito requiere ingenieros de software quienes poseen un conjunto adecuado de conocimientos, habilidades, formación y experiencia en práctica profesional.

El término "práctica profesional" se refiere a un forma de llevar a cabo servicios para lograr cer- Mantener estándares o criterios tanto en el proceso de realizar un servicio y el resultado del producto final ing del servicio. Estas normas y criterios ria puede incluir tanto técnicos como no técnicos aspectos. El concepto de práctica profesional puede ser visto como más aplicable dentro de esos profesiones que tienen un cuerpo generalmente aceptado del conocimiento; códigos de ética y profesional conducta con sanciones por infracciones; aceptado procesos de acreditación, certificación y licenciamiento; y sociedades profesionales para proporcionar y administrar todo esto. Admisión a estos las sociedades profesionales a menudo se basan en un pre combinación descrita de educación y experiencia.

Un ingeniero de software mantiene un profesional. practicar realizando todo el trabajo de acuerdo con prácticas generalmente aceptadas, estándares y Directrices establecidas en particular por el pro- sociedad profesional Por ejemplo, la Asociación para Computing Machinery (ACM) y IEEE Com- puter Society (IEEE CS) ha establecido un Soft- Código de Ética y Profesional de Ingeniería de Ware Práctica. Tanto la British Computer Society (BCS) y la Federación Internacional de Información El procesamiento (IFIP) ha establecido profesiones similares normas prácticas nacionales. ISO / IEC e IEEE tienen más software internacionalmente aceptado normas de ingeniería (ver Apéndice B de este *Guía*). IEEE CS ha establecido dos internacionales programas de certificación (CSDA, CSDP) y un corre- *Guía de* patrocinio *del organismo de ingeniería de software*

manejar problemas de ingeniería únicos, produciendo *de conocimiento* (*Guía SWEBOK*). Todos estos son

Figura 11.1. Desglose de temas para la práctica profesional de ingeniería de software KA

elementos que sientan las bases del profesorado práctica nacional de ingeniería de software.

DESGLOSE DE TEMAS PARA INGENIERÍA DE SOFTWARE PRACTICA PROFESIONAL

La práctica profesional de ingeniería de software El desglose de temas de KA se muestra en la Figura 11.1 Las subáreas presentadas en este KA son pro- profesionalismo, dinámica de grupo y psicología, y habilidades de comunicación.

1. Profesionalismo

Un ingeniero de software muestra profesionalismo. notablemente a través de la adhesión a los códigos de ética y conducta profesional y a las normas y

prácticas establecidas por el ingeniero comunidad profesional

de certificación es la certificación profesional, donde una persona está certificada como capaz de completar un

La comunidad profesional a menudo es representada sembrado por una o más sociedades profesionales; esas sociedades publican códigos de ética y profesiones conducta nacional así como criterios de admisión a la comunidad Esos criterios forman la base para actividades de acreditación y licencia y puede ser usado como una medida para determinar la ingeniería competencia o negligencia.

1.1. Acreditación, Certificación y Licencias

[1 *, c1s4.1, c1s5.1 – c1s5.4]

1.1.1. Acreditación

La acreditación es un proceso para certificar la competencia. tenencia, autoridad o credibilidad de una organización. Las escuelas o programas acreditados tienen la seguridad de adherirse a estándares particulares y mantener ciertas calidades de tain. En muchos países, los medios básicos por el cual los ingenieros adquieren conocimiento es a través de finalización de un curso de estudio acreditado. A menudo, la acreditación de ingeniería es realizada por una organización gubernamental, como el ministerio de Educación. Tales países con gobierno Las acreditaciones incluyen China, Francia, Alemania, Israel, Italia y Rusia.

En otros países, sin embargo, la acreditación proceso de acción es independiente del gobierno y realizado por asociaciones de miembros privados. Por ejemplo, en los Estados Unidos, ingeniero La acreditación es realizada por una organización ción conocida como ABET. Una organización conocida como CSAB sirviendo como un organismo participante de ABET es la sociedad líder dentro de ABET para la acreditación ción de programas de grado en ingeniería de software.

Si bien el proceso de acreditación puede ser diferente ferent para cada país y jurisdicción, el general El significado es el mismo. Para el curso de una institución de estudiar para acreditarse significa que "la acreditación Organismo reconoce una institución educativa como mantener estándares que califiquen a los graduados para admisión a instituciones superiores o más especializadas instituciones o para la práctica profesional "[2].

1.1.2. Proceso de dar un título

La certificación se refiere a la confirmación de un Las características particulares del hijo. Un tipo común actividad en una determinada disciplina en un nivel establecido de competencia. Certificación profesional también También puede verificar la capacidad del titular para cumplir normas profesionales y para aplicar profesional juicio al resolver o abordar problemas. La certificación profesional también puede involucrar a verificación del conocimiento prescrito, el maestro ing de mejores prácticas y metodologías comprobadas, y la cantidad de experiencia profesional.

Un ingeniero generalmente obtiene la certificación por aprobar un examen en conjunto con otro criterios basados en la experiencia. Estos exámenes a menudo son administrados por organizaciones no gubernamentales organizaciones, como las sociedades profesionales.

En ingeniería de software, certificación testi cumple con la calificación de uno como ingeniero de software. Por ejemplo, el IEEE CS ha promulgado dos cer- Programas de tificación (CSDA y CSDP) diseñados para confirmar el conocimiento de un ingeniero de software de prácticas estándar de ingeniería de software y para avanzar en la carrera de uno. La falta de certificación hace no excluya al individuo de trabajar como ingeniero de software. Actualmente certificación en soft- La ingeniería de artículos es completamente voluntaria. De hecho, la mayoría de los ingenieros de software no están certificados bajo cualquier programa

1.1.3. Licencia

"Licencia" es la acción de dar a una persona autorización para realizar ciertos tipos de actividades lazos y asumir la responsabilidad del ingeniero resultante ing productos. El sustantivo "licencia" se refiere a ambos esa autorización y el registro del documento esa autorización Autoridades gubernamentales o Los organismos estatutarios suelen emitir licencias.

Obtener una licencia para practicar requiere no solo que un individuo cumple un cierto estándar, pero también que lo hacen con cierta habilidad para practicar tice u opere. A veces hay un nivel de entrada requisito que establece las habilidades mínimas y capacidades para practicar, pero como profesional se mueve a través de su carrera, lo requerido Las habilidades y capacidades cambian y evolucionan.

En general, los ingenieros tienen licencia como medio de protegiendo al público de personas no calificadas. En algunos países, nadie puede practicar como un pro- ingeniero profesional a menos que tenga licencia; o más, no

Página 194

la empresa puede ofrecer "servicios de ingeniería" a menos que al menos un ingeniero con licencia trabaja allí.

1.2. Códigos de ética y conducta profesional

[1 *, c1s6 – c1s9] [3 *, c8] [4 *, c1s2] [5 *, c33] [6 *]

Códigos de ética y conducta profesional valorar los valores y el comportamiento de un ingeniero la práctica profesional y las decisiones deben encarnar.

La comunidad profesional establece códigos. de ética y conducta profesional. Ellos existen en el contexto de, y se ajustan para estar de acuerdo con, normas sociales y leyes locales. Por lo tanto, los códigos de ética y conducta profesional presente Desde estándares y códigos de ética y pro La conducta profesional puede ser introducida, modificada, o reemplazado en cualquier momento, motor de software individual los neers asumen la responsabilidad de sus propios problemas estudio continuo para mantenerse al día en su profesional práctica.

1.3. Naturaleza y papel de las sociedades profesionales.

[1 *, c1s1 – c1s2] [4 *, c1s2] [5 *, c35s1]

Las sociedades profesionales se componen de una mezcla. de profesionales y académicos. Estas sociedades sirven para definir, avanzar y regular su correlación Profesiones que responden. Sociedades profesionales ayudar a establecer estándares profesionales también como códigos de ética y conducta profesional. por

ante los imperativos conflictivos.

Una vez establecidos, los códigos de ética y profesla profesión hace cumplir la conducta nacional, representado por sociedades profesionales o por un organismo de derecho público.

Las violaciones pueden ser actos de comisión, como como ocultar el trabajo inadecuado, revelar información confidencial, información falsa o tergiversar las habilidades de uno. También pueden ocurrir por omisión, incluida la falta de riesgos cercanos o para proporcionar información importante, no otorgar el crédito adecuado o no reconocer referencias y falta de representación del cliente ests. Violaciones de códigos de ética y profesiones. La conducta nacional puede dar lugar a sanciones y sancionesexpulsión posible del estado profesional.

Un código de ética y conducta profesional para la ingeniería de software fue aprobada por la ACM El Consejo y la Junta de Gobernadores de IEEE CS en 1999 [6 *]. Según la versión corta de esto código:

> Los ingenieros de software los comprometerán yo mismo para hacer el análisis, especificación, diseño, desarrollo, pruebas y mantenimiento de un software beneficioso y Profesión respetada. De acuerdo con su compromiso con la salud, seguridad y bienestar del público, ingenieros de software se adherirá a los ocho principios concerniente al público, cliente y empleador, producto, juicio, gestión, profesion, colegas y self, respectivamente.

Por esta razón, también participan en actividades relacionadas, que incluye

- establecer y promulgar un cuerpo de gen- conocimiento aceptado por vía oral;
- acreditación, certificación y licenciamiento;
- dispensar acciones disciplinarias;
- avanzar en la profesión a través de conferir ences, capacitación y publicaciones.

La participación en sociedades profesionales ayuda el ingeniero individual en mantenimiento y afilado Observando su conocimiento profesional y relevancia y en expandir y mantener su profesión sd nacional.

1.4. Naturaleza y papel de la ingeniería de software

Normas

[1 *, c5s3.2, c10s2.1] [5 *, c32s6] [7 *, c1s2]

Los estándares de ingeniería de software cubren una observación: Gran variedad de temas. Proporcionan pautas para la práctica de ingeniería y procesos de software para ser utilizado durante el desarrollo, mantenimiento y soporte de software. Al establecer un consenso cuerpo de conocimiento y experiencia, ingeniería de software las normas de negociación establecen una base sobre la cual Se pueden desarrollar otras pautas. Apéndice B de Esta *guía* proporciona orientación sobre IEEE e ISO / Estándares de ingeniería de software IEC que admiten Las áreas de conocimiento de esta *guía* .

Los beneficios de los estándares de ingeniería de software son muchos e incluyen mejorar la calidad del software,

Página 195

ayudando a evitar errores, protegiendo ambos software productores y usuarios, aumentando el profesional dis cipline, y ayudando a la transición tecnológica.

1.5. Impacto económico del software

[3 *, c10s8] [4 *, c1s1.1] [8 *, c1]

El software tiene efectos económicos en el individuo, negocios y niveles sociales. Software "éxito" puede estar determinado por la idoneidad de un producto para un problema reconocido, así como por su efecto actividad cuando se aplica a ese problema.

A nivel individual, la continuidad de un ingeniero El empleo puede depender de su capacidad y disposición para interpretar y ejecutar tareas para satisfacer las necesidades de los clientes o empleadores y esperanzas de heredar. Las finanzas del cliente o del empleador La situación social a su vez puede ser positiva o negativa. afectado positivamente por la compra de software.

A nivel empresarial, el software se aplica correctamente a un problema puede eliminar meses de trabajo y se traducen en ganancias elevadas o más efecto 5 organizaciones. Además, organizaciones que adquirir o proporcionar software exitoso puede ser un bendición para la sociedad en la que operan por proofreciendo empleo y mejores servicios. Sin embargo, los costos de desarrollo o adquisición de el software también puede equipararse a los de cualquier importante adquisición.

A nivel social, impactos directos del software.

consideración. Aquí, estamos más preocupados por el arreglo de ingeniero a cliente y su acuerdos o contratos concomitantes, ya sea que son de contratación directa o de consultores, y los problemas que suelen abordar.

Una preocupación común en ingeniería de software contratos es confidencialidad. Los empleadores derivan ventaja comercial de la propiedad intelectual, entonces se esfuerzan por proteger esa propiedad de cierre. Por lo tanto, los ingenieros de software son a menudo requerido para firmar la no divulgación (NDA) o la inteligencia acuerdos de propiedad lectual (IP) como precondición al trabajo. Estos acuerdos generalmente se aplican a la información que el ingeniero de software solo podía ganar a través de la asociación con el cliente. los y términos de estos acuerdos pueden extenderse más allá del plazo ción de la asociación.

Otra preocupación es la propiedad de IP. Derechos a activos de ingeniería de software —productos, innovaiones, inventos, descubrimientos e ideas: mayo residir con el empleador o cliente, ya sea bajo términos contractuales explícitos o leyes relevantes, si esas los activos se obtienen durante la vigencia del softrelación del ingeniero de mercancías con ese empleador o cliente Los contratos difieren en la propiedad de activos creados utilizando equipos no propiedad del empleador ment o información.

Finalmente, los contratos también pueden especificar entre otros elementos la ubicación en la que el trabajo es ser realizado normas a las que va ese trabajo

el éxito o el fracaso incluyen o excluyen accidentes, interrupciones y pérdida de servicio. Impactos indirectos incluir el éxito o el fracaso de la organización que adquirió o produjo el software, aumentó o disminución de la productividad social, armonioso o orden social disruptivo, e incluso el ahorro o pérdida de propiedad y vida.

1.6. Contratos de trabajo

[1 *, c7]

Se pueden proporcionar servicios de ingeniería de software bajo una variedad de relaciones cliente-ingeniero. El trabajo de ingeniería de software puede ser solicitado fue designado como proveedor de empresa a cliente, ingeniero consultoría al cliente, contratación directa o incluso trabajar como voluntario. En todas estas situaciones, el cliente Tomer y proveedor acuerdan que un producto o servicio se proporcionará vicio a cambio de algún tipo de

ser retenido la configuración del sistema que se utilizará para desarrollo, limitaciones de la ingeniería del software responsabilidad del empleador y del neer; una comunicacion matriz y / o plan de escalamiento; y administrativa detalles como tasas, frecuencia de compensación, horas de trabajo y condiciones de trabajo.

1.7. Asuntos legales

[1 *, c6, c11] [3 *, c5s3 – c5s4] [9 *, c1s10]

Problemas legales relacionados con la ingeniería de software la práctica profesional incluye especialmente asuntos relacionados con normas, marcas registradas, patentes, copias derechos, secretos comerciales, responsabilidad profesional, legal requisitos, cumplimiento comercial y cibercrimen. Por lo tanto, es beneficioso poseer conocimiento de estos problemas y su aplicabilidad.

Los asuntos legales están basados en la jurisdicción; suave- los ingenieros de mercancías deben consultar a abogados que

Página 196

especializarse en el tipo y jurisdicción de cualquier iden- problemas legales justificados.

1.7.1. Normas

Las normas de ingeniería de software establecen guías líneas para prácticas generalmente aceptadas y mini- requisitos de mamá para productos y servicios pro- dirigido por un ingeniero de software. Apéndice B de este *La guía* proporciona orientación sobre el ingeniero de software ing estándares que son aplicables a cada KA.

Las normas son valiosas fuentes de requisitos. y asistencia durante la conducta diaria de actividades de ingeniería de software. Adherido a las normas facilitan la disciplina al enumerar Características mínimas de los productos y la práctica. Esa disciplina ayuda a mitigar el subconsciente. supuestos o exceso de confianza en un diseño. por Por estas razones, las organizaciones que realizan software las actividades de ingeniería a menudo incluyen conformidad a los estándares como parte de su política organizacional llora Además, el cumplimiento de las normas es un importante componente de defensa de acciones legales o de acusaciones de mala práctica.

1.7.2. Marcas registradas

Una marca registrada se refiere a cualquier palabra, nombre, símbolo, o dispositivo que se utiliza en transacciones comerciales. Se utiliza "para indicar la fuente u origen del bienes "[2].

La protección de marca protege los nombres, logotipos, imágenes y empaques. Sin embargo, si un nombre, imagen, u otro activo registrado se convierte en un término genérico, entonces la protección de la marca queda anulada.

La Organización Mundial de la Propiedad Intelectual (OMPI) es la autoridad que enmarca las reglas y normativa sobre marcas registradas. La OMPI es Estados Agencia de las naciones dedicada al uso de intelec- propiedad real como medio para estimular la innovación ción y creatividad.

1.7.3. Patentes

Las patentes protegen el derecho de un inventor a la fabricación. son una antigua forma de protección de propiedad de ideas y datan del siglo XV.

La solicitud de patente implica registros cuidadosos del proceso que condujo a la invención. Patentar los abogados son útiles para escribir la divulgación de patentes reclama de la manera más probable para proteger el blando derechos de ingeniero de mercancías.

Tenga en cuenta que, si los inventos se realizan durante el curso de un contrato de ingeniería de software, propietario- abarco puede pertenecer al empleador o cliente o ser conjuntamente, en lugar de pertenecer al software ingeniero.

Hay reglas sobre lo que es y lo que no es patentable En muchos países, el código del software es no patentable, aunque los algoritmos de software pueden ser. Las solicitudes de patentes existentes y presentadas pueden ser buscado en la OMPI.

1.7.4. Derechos de autor

La mayoría de los gobiernos del mundo dan exclusivos derechos de una obra original a su creador, generalmente por tiempo limitado, promulgado como copyright. Dupdo- los derechos protegen la forma en que se presenta una idea, no La idea misma. Por ejemplo, pueden proteger el redacción particular de una cuenta de un histórico evento, mientras que el evento en sí no está protegido. Sin embargo, los derechos de autor son a largo plazo y renovables; ellos datan del siglo 17.

1.7.5. Secretos comerciales

En muchos países, un activo intelectual como una fórmula, algoritmo, proceso, diseño, método, patrón, instrumento o compilación de información se puede considerar un "secreto comercial", siempre que que estos activos no son generalmente conocidos y pueden Proporcionar a una empresa alguna ventaja económica. La designación de "secreto comercial" proporciona legal protección si el activo es robado. Esta proteccion no está sujeto a un límite de tiempo. Sin embargo, si otro la parte deriva o descubre el mismo activo legalmente, entonces el activo ya no está protegido y el otro La parte también tendrá todos los derechos para usarlo.

Ture y venda una idea. Una patente consiste en un conjunto de derechos exclusivos otorgados por un gobierno soberano mensaje a un individuo, grupo de individuos, o organización por un período de tiempo limitado. Patentes

1.7.6. Responsabilidad profesional

Es común que los ingenieros de software sean relacionado con asuntos de responsabilidad profesional. Como

Page 197

un individuo brinda servicios a un cliente o empleador, es vital cumplir con los estándares y prácticas generalmente aceptadas, protegiendo así contra acusaciones o procedimientos de o relacionados con negligencia, negligencia o incompetencia.

Para ingenieros, incluidos ingenieros de software, La responsabilidad profesional está relacionada con la responsabilidad de ity. Según las leyes y normas que rigen en su jurisdicción, los ingenieros pueden rendir cuentas por no seguir plena y concienzudamente práctica recomendada; esto se conoce como "negligencia gence ". También pueden estar sujetos a las leyes que gobiernan ing "responsabilidad estricta" e implícita o expresa garantía, donde, al vender el producto, el motor Neer se considera para garantizar que el producto es tanto Apto y seguro para su uso. En algunos países (para ejemplo, en los Estados Unidos), "privity" (la idea de que un solo podría demandar a la persona que vende el producto) es ya no es una defensa contra acciones de responsabilidad.

Las demandas legales por responsabilidad pueden ser sometidas a La ley de responsabilidad civil en los EE. UU. permite que cualquier para recuperar su pérdida incluso si no hubiera garantías hecho. Porque es difícil medir el traje capacidad o seguridad del software, falta de cumplimiento el cuidado se puede usar para probar negligencia por parte de ingenieros de software. Una defensa contra tal alegación es para mostrar que las normas y en general las prácticas aceptadas fueron seguidas en el desarrollo ment del producto.

1.7.7. Requerimientos legales

Los ingenieros de software deben operar dentro del multas de legal local, nacional e internacional marcos. Por lo tanto, los ingenieros de software deben Tenga en cuenta los requisitos legales para

- registro y licencia, incluidos los exámenes nación, educación, experiencia y entrenamiento requisitos;
- acuerdos contractuales;
- legalidades no contractuales, como las de los gobiernos responsabilidad civil;
- Información básica sobre la legalidad internacional. Se puede acceder al marco desde el mundo Organización de Comercio (OMC).

1.7.8. Cumplimiento comercial

Todos los profesionales de software deben tener en cuenta restricciones legales a la importación, exportación o reexportación de bienes, servicios y tecnología en la jurisprudencia Dicciones en las que trabajan. Las consideraciones incluidas en el control de exportación y clasificación, transferencia de bienes, adquisición de los necesarios gubernamentales licencias para uso extranjero de hardware y software, servicios y tecnología por nación sancionada, empresas o entidades individuales, e importación restricciones y deberes. Los expertos en comercio deberían ser consultado para una guía detallada de cumplimiento.

1.7.9. Cibercrimen

El delito cibernético se refiere a cualquier delito que implique una computadora, software de computadora, red de computadoras funciona o software integrado que controla un sistema tenido. La computadora o el software pueden haber sido utilizado para cometer un delito o puede tener sido el objetivo Esta categoría de delito incluye fraude, acceso no autorizado, spam, obsceno o contenido ofensivo, amenazas, acoso, robo de datos personales confidenciales o secretos comerciales, y uso de una computadora para dañar o infiltrarse en otra computadoras en red y sistema automatizado controles.

Los usuarios de computadoras y software cometen fraude por alterar los datos electrónicos para facilitar la actividad ilegal ity. Las formas de acceso no autorizado incluyen pirateo ing, espionaje y uso de sistemas informáticos de una manera que está oculta a sus dueños.

Muchos países tienen leyes separadas para cubrir delitos cibernéticos, pero a veces ha sido difícil para enjuiciar los delitos cibernéticos debido a la falta de pre estatutos enmarcados cisely. El ingeniero de software tiene obligación profesional de considerar la amenaza de cibercrimen y entender cómo funciona el software el sistema protegerá o pondrá en peligro el software y el usuario información de acceso accidental o malicioso, uso, modificación, destrucción o divulgación.

1.8. Documentación

[1 *, c10s5.8] [3 *, c1s5] [5 *, c32]

Proporcionar documentos claros, completos y precisos. la responsabilidad es responsabilidad de cada software ingeniero. La adecuación de la documentación es

Página 198

juzgado por diferentes criterios basados en las necesidades de los diversos públicos interesados

Buena documentación cumple con aceptado normas y pautas. En particular, el software los ingenieros deben documentar

- hechos relevantes,
- riesgos significativos y compensaciones, y
- advertencias de consecuencias indeseables o peligrosas quejas del uso o mal uso del software.

Los ingenieros de software deben evitar

- certificar o aprobar productos inaceptables,
- revelar información confidencial, o
- falsificar hechos o datos.

Además, los ingenieros de software y su personal los agentes deben proporcionar, en particular, los siguientes documentos Mentalización para su uso por otros elementos del software organización de desarrollo de artículos:

- especificaciones de requisitos de software, software-documentos de diseño de la cerámica, detalles sobre el software herramientas de ingeniería de hardware utilizadas, prueba de software especificaciones y resultados, y detalles sobre el métodos de ingeniería de software adoptados;
- problemas encontrados durante el desarrollo proceso de ment.

Para partes interesadas externas (clientes, usuarios, otros) la documentación del software debe notablemente proporcionar

- información necesaria para determinar si el software es probable que el software cumpla con los requisitos de los objetivos necesidades de los usuarios,
- descripción del uso seguro e inseguro del software,
- descripción de la protección de los sensibles información creada o almacenada usando el software y
- identificación clara de advertencias y críticas procedimientos

El uso del software puede incluir instalación, operación ación, administración y desempeño de otros funciones de varios grupos de usuarios y soporte personal. Si el cliente adquirirá la propiedad del código fuente del software o el derecho a modificar el código, el ingeniero de software debe proporcionar documentación de las especificaciones funcionales, el diseño del software, el conjunto de pruebas y lo necesario entorno operativo sary para el software.

El período mínimo de tiempo que los documentos deben mantenerse es la duración de los productos de software ' ciclo de vida o el tiempo requerido por las organizaciones relevantes requisitos nacionales o reglamentarios.

1.9. Análisis de compensación

[3 *, c1s2, c10] [9 *, c9s5.10]

Dentro de la práctica de la ingeniería de software, un ingeniero de software a menudo tiene que elegir entre Soluciones alternativas de problemas. El resultado de Estas opciones están determinadas por el software del motor. evaluación profesional de los riesgos, costos, beneficios de alternativas, en cooperación con partes interesadas La evaluación del ingeniero de software se llama "análisis de compensación". Análisis de compensación notablemente permite la identificación de la competencia ing y requisitos de software complementarios software para priorizar el conjunto final de requisitos del software a construir (ver Negociación de requisitos en el software Requisitos KA y Determinación y Negociación tiación de requisitos en la ingeniería del software Dirección de gestión KA).

En el caso de un desarrollo continuo de software, proyecto atrasado o por encima del presupuesto, compensación a menudo se realiza un análisis para decidir qué software los requisitos de hardware se pueden relajar o descartar dados los efectos de los mismos.

Un primer paso en un análisis de compensación es establecer los objetivos de diseño (ver Diseño de ingeniería en el Fundamentos de Ingeniería KA) y establecer el importancia relativa de esos objetivos. Esto permite identificación de la solución que más se acerca cumple con esos objetivos; esto significa que la forma en que Los objetivos establecidos son de importancia crítica.

Los objetivos de diseño pueden incluir la minimización de costo monetario y maximización de la confiabilidad, rendimiento, o algún otro criterio en general gama de dimensiones. Sin embargo, es difícil formular un análisis de compensación de costo contra riesgo, especialmente donde la producción primaria y la segunda Los costos basados en el riesgo deben ser negociados contra cada otro.

Page 199

Un ingeniero de software debe realizar una compensación análisis de una manera ética, especialmente por ser objetivo e imparcial al seleccionar criterios para comparación de soluciones alternativas de problemas y al asignar pesos o importancia a estos criterios Cualquier conflicto de intereses debe ser revelado. en la delantera.

2. Dinámica grupal y psicología

El trabajo de ingeniería se realiza muy a menudo en el contexto de trabajo en equipo. Un ingeniero de software debe ser capaz de interactuar cooperativamente y construir

Un punto a destacar es que el software los neers deben poder trabajar en forma multidisciplinaria entornos y en dominios de aplicación variados. Desde hoy el software está en todas partes, desde un teléfono para un automóvil, el software está afectando la vida de las personas lejos más allá del concepto más tradicional de software hecho para la gestión de información en un negocio ambiente.

2.2. Cognición individual

[3 *, c1s6.5] [5 *, c33]

Los ingenieros desean resolver problemas. La habilidad para

interactivamente con otros para determinar primero y luego Satisfacer las necesidades y expectativas. Conocimiento de La dinámica de grupo y la psicología son un activo cuando interactuando con clientes, compañeros de trabajo, proveedores y subordinados para resolver ingeniería de software problemas.

2.1. Dinámica del trabajo en equipos / grupos

[3 *, c1s6] [9 *, c1s3.5, c10]

Los ingenieros de software deben trabajar con otros. En Por un lado, trabajan internamente en ingeniería equipos; por otro lado, trabajan con clientes tomers, miembros del público, reguladores y otras partes interesadas Equipos en ejecución: aquellos que demuestren una calidad de trabajo consistente y progresar hacia los objetivos: son cohesivos y poseen Un ambiente cooperativo, honesto y centrado. Los objetivos individuales y de equipo están alineados para que los miembros se comprometen naturalmente y se sienten dueños de resultados compartidos.

Los miembros del equipo facilitan este ambiente al ser intelectualmente honesto, hacer uso del grupo pensar, admitir ignorancia y reconocer errores de ing. Comparten responsabilidad, recompensas, y carga de trabajo justamente. Cuidan de la comunidad. cate claramente, directamente entre sí y en documentos ments, así como en el código fuente, para que la información Esta es accesible para todos. Revisiones por pares sobre Los productos de trabajo se enmarcan de forma constructiva y de forma no personal (ver Críticas y Auditorías en el Calidad de software KA). Esto permite que todos los miembros Bers para perseguir un ciclo de mejora continua y crecimiento sin riesgo personal. En general, los miembros de equipos cohesivos demuestran respeto el uno para el otro y su líder.

a través de la capacitación y el estudio agregan habilidades y conocimientos ventaja para la cartera del ingeniero de software; leyendo, redes y experimentar con nuevas herramientas, técnicas y métodos son todos medios válidos de desarrollo profesional.

2.3. Lidiando con la Complejidad del Problema

[3 *, c3s2] [5 *, c33]

Muchos, si no la mayoría, problemas de ingeniería de software los lems son demasiado complejos y difíciles de abordar como un todo o para ser abordado por software individual ingenieros Cuando surgen tales circunstancias, el los medios habituales para adoptar es el trabajo en equipo y el descomposición (ver Técnicas de resolución de problemas en las Fundaciones de Computación KA).

Los equipos trabajan juntos para lidiar con complejos y grandes problemas al compartir cargas y sorteos Sobre el conocimiento y la creatividad del otro. Cuando los ingenieros de software trabajan en equipos, difieren Puntos de vista y habilidades de los ingenieros individuales complementarse y ayudar a construir una solución eso es de otra manera difícil de conseguir. Algunos ejemplos específicos de trabajo en equipo para ingeniería de software son programación de pares (ver Métodos ágiles en el Modelos y métodos de ingeniería de software KA) y revisión de código (consulte Revisiones y auditorías en el

Los ingenieros de software deben resolver problemas de manera efectiva y eficiente es lo que todo ingeniero se esfuerza por lograrlo. Sin embargo, los límites y los procesos de cognición individual afectan el problema Lem resolver. En ingeniería de software, especialmente debido a la naturaleza altamente abstracta del software mismo, la cognición individual juega un papel muy destacado en la resolución de problemas

En general, un individuo (en particular, un software capacidad del ingeniero para descomponer un problema y crear desarrollar activamente una solución puede ser inhibida por

- necesidad de más conocimiento,
- supuestos subconscientes,
- volumen de datos,
- miedo al fracaso o consecuencia del fracaso,
- cultura, ya sea dominio de aplicación o organizativo,
- falta de capacidad para expresar el problema,
- ambiente de trabajo percibido, y
- estado emocional del individuo.

El impacto de estos factores inhibidores puede ser reducido al cultivar una buena resolución de problemas hábitos que minimizan el impacto del engaño Suposiciones La capacidad de concentración es vital, como es humildad intelectual: ambos permiten un motor de software nunca para suspender consideraciones personales y interactuar libremente con otros, lo cual es especialmente importante Tant cuando se trabaja en equipo. Hay un conjunto de métodos básicos que usan los ingenieros para facilitar la resolución de problemas (ver Solución de problemas Técnicas de ing en los fundamentos informáticos KA). Desglosando problemas y resolviéndolos Una pieza a la vez reduce la sobrecarga cognitiva. Aprovechando la curiosidad profesional y persiguiendo el desarrollo profesional continuo

Página 200

Por lo tanto, es vital mantenerse abierto y pro comunicación conductiva con las partes interesadas para el duración de la vida útil del producto de software.

2.5. Lidiando con la incertidumbre y la ambigüedad

[4 *, c24s4, c26s2] [9 *, c9s4]

Al igual que con los ingenieros de otros campos, el software de ingeniería los neers a menudo deben tratar y resolver dudas elegancia y ambigüedades al proporcionar servicios y desarrollo de productos. El ingeniero de software debe atacar y reducir o eliminar cualquier falta de claridad que es un obstáculo para realizar el trabajo. A menudo, la incertidumbre es simplemente un reflejo de falta del conocimiento. En este caso, la investigación a través de recurrir a fuentes formales como libros de texto y revistas profesionales, entrevistas con partes interesadas ers, o consulta con compañeros de equipo y compañeros pueden superarlo.

Cuando no se puede superar la incertidumbre o la ambigüedad ven fácilmente, ingenieros de software u organizaciones puede optar por considerarlo como un riesgo de proyecto. En esto caso, las estimaciones de trabajo o los precios se ajustan a sopesar el costo anticipado de abordarlo (ver Gestión de Riesgos en la Ingeniería del Software Gestión KA).

Calidad de software KA).

2.4. Interactuando con las partes interesadas

[9 *, c2s3.1]

Éxito de un esfuerzo de ingeniería de software depende de interacciones positivas con estaca titulares. Deben proporcionar apoyo, información ción y comentarios en todas las etapas del software proceso del ciclo de vida. Por ejemplo, durante el inicio etapas, es crítico identificar a todos los interesados y descubra cómo los afectará el producto, para que suficiente definición de la parte interesada requiere Se pueden capturar de manera adecuada y completa.

Durante el desarrollo, los interesados pueden vide feedback sobre especificaciones y / o principios versiones del software, cambio de prioridad, como así como la aclaración de software nuevo o detallado requisitos Por último, durante el mantenimiento del software y hasta el final de la vida del producto, las partes interesadas vide feedback sobre requisitos en evolución o nuevos así como informes de problemas para que el software pueda ser extendido y mejorado.

2.6. Manejo de ambientes multiculturales

[9 *, c10s7]

Los entornos multiculturales pueden tener un impacto. sobre la dinámica de un grupo. Esto es especialmente verdadero cuando el grupo está geográficamente separado o la comunicación es poco frecuente, ya que tal separación La relación eleva la importancia de cada contacto. La comunicación intercultural es aún más difícil. ficticio si la diferencia en zonas horarias hace oral comunicación menos frecuente.

Los ambientes multiculturales son bastante frecuentes en ingeniería de software, quizás más que en otros campos de la ingeniería, debido a la fuerte tendencia de outsourcing internacional y el envío fácil de componentes de software instantáneamente en el mundo. Por ejemplo, es bastante común para un proyecto de software que se dividirá en partes fronteras nacionales y culturales, y también es bastante común para un equipo de proyecto de software que consiste en personas de diversos orígenes culturales.

Para que un proyecto de software sea un éxito, equipo los miembros deben alcanzar un nivel de tolerancia,

reconociendo que algunas reglas dependen de la sociedad normas sociales y que no todas las sociedades derivan la Las mismas soluciones y expectativas.

Esta tolerancia y comprensión que lo acompaña ing puede ser facilitado por el apoyo del liderazgo y gestión. Comunicación más frecuente, incluyendo reuniones cara a cara, puede ayudar a mitigar puerta divisiones geográficas y culturales, promover cohesión y aumentar la productividad. Además, siendo capaz de comunicarse con compañeros de equipo en su El idioma nativo podría ser muy beneficioso.

3. Habilidades de comunicación

Es vital que un ingeniero de software se comunique bueno, tanto oralmente como en lectura y escritura. Suclogro cuidadoso de los requisitos de software y los plazos dependen del desarrollo de un claro subde pie entre el ingeniero de software y clientes, supervisores, compañeros de trabajo y proveedores ers. La solución óptima de problemas es posible a través de la capacidad de investigar, comprender, y resumir la información. Producto del cliente La aceptación y el uso seguro del producto dependen de provisión de capacitación y documentación relevante. Se deduce que la propia carrera del ingeniero de software el éxito se ve afectado por la capacidad de consistentemente Proporcionar comunicación oral y escrita. activamente y a tiempo.

3.1. Lectura, comprensión y resumen

[5 *, c33s3]

Los ingenieros de software pueden leer y entender soporte de material técnico. Material técnico incluye libros de referencia, manuales, investigación documentos y código fuente del programa.

La lectura no es solo una forma primaria de mejora habilidades, pero también una forma de recopilar información acción necesaria para completar la ingeniería metas. Un ingeniero de software tamiza a través de o reescribir software, es crítico entender tanto su implementación directamente derivada de la código presentado y su diseño, que a menudo debe inferirse.

3.2. Escritura

[3 *, c1s5]

Los ingenieros de software pueden producir por escrito productos según lo requieran las solicitudes del cliente o genpráctica aceptada por vía oral. Estos productos escritos puede incluir código fuente, planes de proyectos de software, documentos de requisitos de software, análisis de riesgos, documentos de diseño de software, planes de prueba de software, manuales de usuario, informes técnicos y evaluaciones, justificaciones, diagramas y gráficos, etc.

Escribir de manera clara y concisa es muy importante porque a menudo es el método principal de comp comunicación entre las partes relevantes. En todos los casos, los productos de ingeniería de software escritos deben ser escrito para que sean accesibles, entiendan capaz y relevante para sus audiencias previstas.

3.3. Comunicación de equipo y grupo

[3 *, c1s6.8] [4 *, c22s3] [5 *, c27s1] [9 *, c10s4]

Comunicación efectiva entre equipo y grupo. los miembros son esenciales para un software colaborativo esfuerzo de ingeniería. Las partes interesadas deben ser consultado, se deben tomar decisiones, y los planes deben ser generado Cuanto mayor sea el número de equipo y miembros del grupo, mayor es la necesidad de comunicar.

El número de vías de comunicación, cómo siempre, crece cuadráticamente con la adición de cada miembro del equipo Además, miembros del equipo es poco probable que se comuniquen con alguien más de dos grados (niveles). Este problema puede ser más serio cuando los esfuerzos de ingeniería de software o

información modulada, filtrando las piezas que
Será de gran ayuda. Los clientes pueden solicitar que
un ingeniero de software resume los resultados de
tal recopilación de información para ellos, simplificando
o explicándolo para que puedan llegar al final
elección entre soluciones competitivas.

Leer y comprender el código fuente es
También un componente de la recopilación de información y
resolución de problemas Al modificar, extender,
Las organizaciones se extienden a nivel nacional y
fronteras continentales

Alguna comunicación se puede lograr en
escritura. La documentación del software es común
sustituto de la interacción directa. El correo electrónico es otro
pero, aunque es útil, no siempre es suficiente;
Además, si uno envía demasiados mensajes, se convierte en
difícil de identificar la información importante.
Cada vez más, las organizaciones utilizan empresas

Página 202

herramientas de colaboración para compartir información. Además, la crea-
ción, el uso de tiendas de información electrónica,
accesible a todos los miembros del equipo, para organizaciones
Políticas, estándares, ingeniería común.
los procedimientos y la información específica del proyecto pueden
Ser más beneficioso.

Algunos equipos de ingeniería de software se centran en
interacción cara a cara y promover tales interacciones
Acción por disposición de espacio de oficina. A pesar de que
las oficinas privadas mejoran la productividad individual,
ubicar a los miembros del equipo en forma física o virtual
formas y proporcionar áreas de trabajo comunales es
importante para los esfuerzos de colaboración.

3.4. Habilidades de presentación

[3 *, c1s5] [4 *, c22] [9 *, c10s7 – c10s8]

Los ingenieros de software confían en su presentación.
habilidades durante los procesos del ciclo de vida del software; por
ejemplo, durante los requisitos de software
Elicitación, los ingenieros de software pueden acompañar a los clientes
y compañeros de equipo a través de requisitos de software
y realizar revisiones formales de requisitos (ver
Revisiones de requisitos en el requisito de software
Medida KA). Durante y después del diseño del software,
construcción de software y mantenimiento de software,
los ingenieros de software lideran las revisiones, el recorrido del producto
a través (ver Revisión y auditorías en el software
Calidad KA) y formación. Todos estos requieren el
capacidad de presentar información técnica a grupos
y solicitar ideas o comentarios.

La capacidad del ingeniero de software para transmitir
conceptos efectivamente en una presentación por lo tanto
influye en la aceptación del producto, la gestión,
y atención al cliente; También influye en la capacidad
Las partes interesadas deben comprender y ayudar en
El esfuerzo del producto. Este conocimiento necesita ser
archivado en forma de diapositivas, conocimiento escrito
en documentos técnicos y cualquier otro material
utilizado para la creación de conocimiento.

203 de 1189.

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	0 0 0 0 [1 *] ott y col. 20 segundo	03 d 20 [3 *] Olan V	ile 2011 erv [4 *] metro om S	04 ell 20 norte [5 *] en cC METRO	M 1999 do (UNA S [6 *] -DO IEEE	06 [7 *] oore 20 METRO	04 ey 20 [8 *] ock T	09 [9 *] airley 20 F
1. Profesionalismo								
1.1. Acreditación, Certificación, y Licencia	c1s4.1, c1s5.1–c1s5.4							
1.2. Códigos de ética y profesional Conducta	c1s6–c1s9	c8	c1s2	c33	* *			
1.3. Naturaleza y Rol de profesional Sociedades	c1s1–c1s2		c1s2	c35s1				
1.4. Naturaleza y Rol del software Ingenieria Normas	c5s3.2, c10s2.1			c32s6		c1s2		
1.5. Económico Impacto del software		c10s8	c1s1.1				c1	
1.6. Empleo Contratos	c7							
1.7. Asuntos legales	c6, c11	c5s3–c5s4						c1s10
1.8. Documentación	c10s5.8	c1s5		c32				
1.9. Compensación Análisis		c1s2, c10						c9s5.10
2. Dinámica de grupo y psicología								
2.1. Dinámica de Trabajando en equipos / Grupos		c1s6						c1s3.5, c10
2.2. Individual Cognición		c1s6.5		c33				
2.3. 2.3 Tratar con Complejidad del problema		c3s2		c33				
2.4. Interactuando con Partes interesadas								c2s3.1

204 de 1189.

	0 0 0 0 [1 *] ott y col. 20 segundo	03 d 20 [3 *] Olan V	ile 2011 erv [4 *] metro om S	04 ell 20 norte [5 *] en cC METRO	M 1999 do (UNA S [6 *] -DO IEEE	06 [7 *] oore 20 METRO	04 ey 20 [8 *] ock T	09 [9 *] airley 20 F

2.5. Tratando con Incertidumbre y Ambigüedad		c24s4, c26s2		c9s4
2.6. Tratando con Multicultural Ambientes				c10s7
3. Comunicación Habilidades				
3.1. Leyendo, Comprensión y Resumiendo			c33s3	
3.2. Escritura	c1s5			
3.3. Equipo y grupo Comunicación	c1s6.8	c22s3	c27s1	c10s4
3.4. Presentación Habilidades	c1s5	c22		c10s7–c10s8

LECTURAS ADICIONALES

Gerald M. Weinberg, *La psicología de Programación informática* [10].

Este fue el primer libro importante para abordar el programa ming como un esfuerzo individual y de equipo y se convirtió un clásico en el campo d.

Kinney y Lange, PA, *Propiedad Intelectual Ley de Abogados de Empresa* [11].

Este libro cubre las leyes de propiedad intelectual en los Estados Unidos. También habla sobre lo que es la ley de propiedad intelectual; también explica por qué se ve como se ve.

Referencias

[1 *] F. Bott et al., *Problemas profesionales en Ingeniería de Software* , 3ª ed., Taylor & Francis, 2000.

[2] *Diccionario colegiado de Merriam-Webster* , 11a ed., 2003.

[3 *] G. Voland, *Ingeniería por diseño* , 2ª ed., Prentice Hall, 2003.

[4 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[5 *] S. McConnell, *Código completo* , 2ª ed., Microsoft Press, 2004.

[6 *] IEEE CS / ACM Joint Task Force sobre Ética de Ingeniería de Software y Prácticas profesionales, "Software Código de Ética de Ingeniería y Práctica profesional (Versión 5.2), "1999;

www.acm.org/serving/se/code.htm .

[7 *] JW Moore, *La hoja de ruta hacia el software Ingeniería: una guía basada en estándares* , Wiley-IEEE Computer Society Press, 2006.

[8 *] S. Tockey, rendimiento *del software: maximización el retorno de su inversión en software* , Addison-Wesley, 2004.

[9 *] RE Fairley, *Gestión y liderazgo Proyectos de software* , computadora Wiley-IEEE Society Press, 2009.

[10] GM Weinberg, *La psicología de programación de computadoras: plata Edición de aniversario* , Dorset House, 1998.

[11] Kinney y Lange, PA, *Intelectual Ley de Propiedad para Abogados Comerciales* , Thomson West, 2013.

CAPITULO 12

ECONOMÍA DE INGENIERÍA DE SOFTWARE

SIGLAS

EVM	Gestion del valor ganado
TIR	Tasa interna de retorno
MARR	Tasa mínima aceptable de Regreso
SDLC	Ciclo de vida del desarrollo de programas
SPLC	Ciclo de vida del producto de software
ROI	Retorno de la inversión
ROCE	Rendimiento del capital invertido
TCO	Costo total de la propiedad

INTRODUCCIÓN

La economía de la ingeniería de software se trata de making decisiones relacionadas con la ingeniería de software en un contexto empresarial. El éxito de un producto de software El servicio, el servicio y la solución dependen de un buen negocio gestión de ness. Sin embargo, en muchas empresas y organizaciones, relaciones comerciales de software para el desarrollo de software y la ingeniería permanecen vago. Esta área de conocimiento (KA) proporciona un Descripción general de la economía de la ingeniería de software.

La economía es el estudio del valor, los costos, recursos y su relación en un contexto dado o situación En la disciplina de software de ingeniería Los objetivos comerciales de la organización. En todo tipos de organizaciones, ya sea "con fines de lucro", "no- con fines de lucro "o gubernamental, esto se traduce en permanecer de manera sostenible en los negocios. En "con fines de lucro" organizaciones esto también se relaciona con el logro un rendimiento tangible del capital invertido tanto activos como capital empleado. Este KA tiene ha sido formulado de una manera para abordar todo tipo de organizaciones independientes de foco, producto y cartera de servicios, o propiedad del capital y taxa- Restricciones

Decisiones como "¿Deberíamos usar un componente específico? nent? "puede parecer fácil desde una perspectiva técnica, pero puede tener serias implicaciones en el negocio viabilidad de un proyecto de software y el resultado producto. A menudo los ingenieros se preguntan si tal las preocupaciones se aplican en absoluto, ya que son "solo ners ". Análisis económico y toma de decisiones son consideraciones importantes de ingeniería porque los ingenieros son capaces de evaluar decisiones tanto Técnicamente y desde una perspectiva empresarial. los Los contenidos de esta área de conocimiento son importantes ics para que los ingenieros de software tengan en cuenta incluso si nunca están realmente involucrados en negocios concretos decisiones de ness; tendrán una vista bien redondeada de cuestiones comerciales y el papel de la consideración técnica Las ediciones juegan en la toma de decisiones comerciales. Muchos propuestas y decisiones de ingeniería, como hacer

neering, las actividades tienen costos, pero el resultado El software en sí también tiene atributos económicos. La economía de la ingeniería de software proporciona un camino para estudiar los atributos de software y software procesa de manera sistemática que los relaciona a medidas económicas. Estas medidas economicas se puede pesar y analizar al tomar decisiones siones que están dentro del alcance de una organización de software nización y aquellos dentro del alcance integrado de Todo un negocio de producción o adquisición.

La economía de la ingeniería de software está preocupada con la alineación de decisiones técnicas de software con versus comprar, tienen profundos impactos económicos intrínsecos eso debe considerarse explícitamente.

Este KA cubre primero los fundamentos, el término clave minología, conceptos básicos y prácticas comunes de economía de ingeniería de software para indicar cómo la toma de decisiones en ingeniería de software incluye, o debería incluir una perspectiva comercial sofware. Luego proporciona una perspectiva del ciclo de vida, destaca la gestión de riesgos e incertidumbres, y muestra cómo se utilizan los métodos de análisis económico. Algunas consideraciones prácticas finalizan el conocimiento. área de borde.

207 de 1189.

Figura 12.1. Desglose de temas para la Ingeniería de Software Economía KA

Page 208

DESGLOSE DE TEMAS PARA ECONOMÍA DE INGENIERÍA DE SOFTWARE

El desglose de temas para el Software Engi-Neering Economics KA se muestra en la Figura 12.1.

1. Economía de la ingeniería de software Fundamentos

1.1. Financiar

[1 *, c2]

Las finanzas son la rama de la economía en cuestión con temas como asignación, gestión, adquisición e inversión de recursos. Financiar es un elemento de cada organización, incluyendo organizaciones de ingeniería de software.

El campo de las finanzas se ocupa de los conceptos de tiempo, dinero, riesgo y cómo están interrelacionados. También trata sobre cómo se gasta el dinero y cómo Geted. Las finanzas corporativas se preocupan por la pro vidriando los fondos para las actividades de una organización. En general, esto implica equilibrar el riesgo y el beneficio. habilidad, al intentar maximizar una organización La riqueza de la nación y el valor de sus acciones. Esta se mantiene principalmente para organizaciones "con fines de lucro", pero también se aplica a organizaciones "sin fines de lucro". Este último necesita finanzas para garantizar la sostenibilidad, sin apuntar a ganancias tangibles. Para hacer esto, un la organización debe

- identificar objetivos organizacionales, horizontes de tiempo, factores de riesgo, consideraciones fiscales y financieras restricciones;
- identificar e implementar el negocio apropiado estrategia de ness, como qué cartera y decisiones de inversión a tomar, cómo gestionar flujo de caja y dónde obtener la financiación;
- medir el desempeño financiero, como flujo de caja y ROI (ver sección 4.3, Retorno en Inversión), y tomar medidas correctivas en caso de desviación de los objetivos y estrategia.

1.2. Contabilidad

[1 *, c15]

La contabilidad es parte de las finanzas. Permite a las personas cuyo dinero se está utilizando para dirigir una organización para conocer los resultados de su inversión: ¿hicieron obtener el beneficio que esperaban? En "con fines de lucro" organizaciones, esto se relaciona con el ROI tangible (consulte la sección 4.3, Retorno de la inversión), mientras que en Organizaciones sin fines de lucro y gubernamentales así como las organizaciones "con fines de lucro", se traduce en permanecer de manera sostenible en los negocios. El primario El papel de la contabilidad es medir la organización desempeño financiero real de la nación comunicar información financiera sobre un negocio entidad a las partes interesadas, como los accionistas, auditores financieros e inversores. Comunicación es generalmente en forma de estados financieros que muestran en términos monetarios los recursos económicos ser controlado. Es importante seleccionar el derecho información que es relevante y confiable para el usuario. La información y su tiempo son parcialmente regido por la gestión de riesgos y la gobernanza políticas Los sistemas contables también son ricos fuente de datos históricos para estimar.

1.3. Controlador

[1 *, c15]

El control es un elemento financiero y contable. Controlar implica medir y corregir El desempeño de las finanzas y la contabilidad. Asegura que los objetivos de una organización y Se cumplen los planes. Controlar el costo es un spe-rama de control cializada utilizada para detectar varices antepasados de los costos reales de los costos planificados.

1.4. Flujo de fondos

[1 *, c3]

El flujo de efectivo es el movimiento de dinero hacia adentro o hacia afuera de un negocio, proyecto o producto financiero sobre un período determinado. Los conceptos de instancias de flujo de efectivo y las corrientes de flujo de efectivo se utilizan para describir la Perspectiva empresarial de una propuesta. Hacer un decisión comercial significativa sobre cualquier específico propuesta, esa propuesta deberá ser evaluada desde una perspectiva empresarial. En una propuesta para desarrollar y lanzar el producto X, el pago por Las nuevas licencias de software son un ejemplo de salida instancia de flujo de caja ing. El dinero necesitaría ser dedicado a llevar a cabo esa propuesta. Los ingresos por ventas del producto X en el undécimo mes después del mercado el lanzamiento es un ejemplo de un flujo de caja entrante

Página 209

Figura 12.2. Un diagrama de flujo de efectivo

ejemplo. El dinero vendría por
llevando a cabo la propuesta.

El término *flujo de caja se* refiere al conjunto de instancias de flujo de efectivo a lo largo del tiempo que son causados por llevando a cabo alguna propuesta dada. El flujo de caja corriente es, en efecto, la imagen financiera completa de esa propuesta. ¿Cuánto dinero sale? Cuando sale? Cuanto dinero viene ¿en? ¿Cuándo entra? Simplemente, si el efectivo la corriente de flujo para la Propuesta A es más deseable que el flujo de caja para la Propuesta B, entonces, todos En igualdad de condiciones, la organización apuesta después de llevar a cabo la Propuesta A que la Propuesta B. Por lo tanto, el flujo de caja es un insumo importante. para la toma de decisiones de inversión. Un flujo de caja instancia es una cantidad específica de dinero que fluye dentro o fuera de la organización en un momento específico como resultado directo de alguna actividad.

Un diagrama de flujo de efectivo es una imagen de un flujo de caja corriente. Le da al lector una descripción rápida de la imagen financiera de la organización en cuestión o proyecto. La figura 12.2 muestra un ejemplo de efectivo diagrama de flujo para una propuesta.

1.5. Proceso de toma de decisiones

[1 *, c2, c4]

Si suponemos que las soluciones candidatas resuelven un dado el problema técnico igualmente bien, ¿por qué debería a la organización le importa cuál es el elegido? los la respuesta es que generalmente hay una gran diferencia influencia en los costos e ingresos de los diferentes soluciones Un objeto comercial, listo para usar, Solicitar el producto del agente puede costar unos pocos miles dólares, pero el esfuerzo por desarrollar una cosecha propia servicio que le da la misma funcionalidad podría cuesta fácilmente cientos de veces esa cantidad.

Si las soluciones candidatas resuelven todas adecuadamente el problema desde una perspectiva técnica, entonces la selección de la alternativa más adecuada debe basarse en factores comerciales como optimizando el costo total de propiedad (TCO) o maximizando el retorno de la inversión a corto plazo (ROI) Costos del ciclo de vida como la corrección de defectos, el servicio de campo y la duración del soporte también son relevantes Consideraciones evasivas. Estos costos deben ser fac- jugado al seleccionar entre tecnología aceptable enfoques nicos, ya que son parte de la vida ROI (ver sección 4.3, Retorno de la inversión).

Un proceso sistemático para tomar decisiones lo garantiza transparencia y permitir una justificación posterior ción Criterios de gobernanza en muchas organizaciones. demanda selección de al menos dos alternativas. Un proceso sistemático se muestra en la Figura 12.3. Comienza con un desafío comercial a la mano y describe los pasos para identificar soluciones alternativas acciones, definir criterios de selección, evaluar las soluciones opciones, implementar una solución seleccionada y monitorear para el desempeño de esa solución.

La Figura 12.3 muestra el proceso como un paso mayormente sabio y serial. El proceso real es más fluido. A veces los pasos se pueden hacer de una manera diferente. orden y, a menudo, se pueden hacer varios de los pasos en paralelo. Lo importante es asegurarse de que

Página 210

Figura 12.3. El proceso básico de toma de decisiones empresariales

ninguno de los pasos se omite ni se reduce. Es también importante entender que este mismo proceso

1.6. Valuación

[1 *, c5, c8]

se aplica a todos los niveles de toma de decisiones: desde un
decisión tan grande como determinar si un software
proyecto debe hacerse en absoluto, para decidir sobre un
algoritmo o estructura de datos para usar en un software
módulo. La diferencia es cómo la firma financiera
importante es la decisión y, por lo tanto, cuánto
se debe invertir el esfuerzo en tomar esa decisión
Sion. La decisión a nivel de proyecto es financieramente sigde
importante y probablemente justifica una relativamente alta
nivel de esfuerzo para tomar la decisión. Seleccionar un
el algoritmo es a menudo mucho menos significativo financieramente
no puede y garantiza un nivel mucho más bajo de esfuerzo para
tomar la decisión, aunque lo mismo básico
Se está utilizando el proceso de toma de decisiones.

Más a menudo que no, una organización podría
realizar más de una propuesta si quisiera
a, y generalmente hay relaciones importantes
entre propuestas. Quizás la Propuesta Y solo puede ser
se lleva a cabo si la Propuesta X también se lleva a cabo. O
quizás la Propuesta P no puede llevarse a cabo si Pro-
la Q posal se lleva a cabo, ni Q podría llevarse a cabo
si P fuera Las elecciones son mucho más fáciles de hacer cuando
hay caminos mutuamente excluyentes, por ejemplo,
ya sea A o B o C o lo que sea elegido. En pre
par decisiones, se recomienda activar cualquier
conjunto de propuestas dado, junto con sus diversos
interrelaciones, en un conjunto de mutuamente excluyentes
Sive alternativas. La elección se puede hacer
entre estas alternativas

En un sentido abstracto, la toma de decisiones pro-
cess, ya sea la toma de decisiones financieras u otra,
se trata de maximizar el valor. La alternativa que
maximiza el valor total siempre se debe elegir.
Una base financiera para la comparación basada en el valor es
Comparar dos o más flujos de efectivo. Varias bases
de comparación están disponibles, incluyendo

- valor actual
- valor futuro
- equivalente anual
- tasa interna de retorno
- período de recuperación (con descuento).

Según el valor temporal del dinero, dos o más
los flujos de efectivo son equivalentes solo cuando son iguales
la misma cantidad de dinero en un punto común
a tiempo. Comparar los flujos de efectivo solo tiene sentido
cuando se expresan en el mismo marco de tiempo.

Tenga en cuenta que el valor no siempre se puede expresar en
Términos de dinero. Por ejemplo, si un artículo
es una marca o no puede afectar significativamente
su valor percibido Valores relevantes que no pueden
ser expresado en términos de dinero todavía debe ser
expresado en términos similares para que puedan ser
evaluado objetivamente.

Página 211

1.7. Inflación [1 *, c13]

La inflación describe las tendencias a largo plazo en los precios.
La inflación significa que las mismas cosas cuestan más
que lo hicieron antes. Si el horizonte de planificación de
una decisión comercial es más larga que unos pocos años, o
si la tasa de inflación supera un par de porcentajes
puntos anualmente, puede causar cambios notables
en el valor de una propuesta. El valor del tiempo presente
por lo tanto, debe ajustarse a las tasas de inflación
y también para las fluctuaciones del tipo de cambio.

1.8. Depreciación [1 *, c14]

La depreciación implica la distribución del costo de un
activo tangible en varios períodos de tiempo;
se usa para determinar cómo las inversiones en capital
los activos contabilizados se cargan contra ingresos sobre
varios años. La depreciación es una parte importante
de determinar el flujo de caja después de impuestos, que es crítico
Cal para abordar con precisión las ganancias y los impuestos. Si
un producto de software se venderá después del desarrollo
se incurre en costos de opciones, esos costos deben ser
capitalizado y depreciado en el tiempo posterior
períodos. El gasto de depreciación por cada vez
período es el costo capitalizado de desarrollar el
software dividido entre el número de períodos
en el que se venderá el software. Un software
La propuesta de proyecto puede compararse con otras
propuestas de software y no software o alternativas
opciones de inversión, por lo que es importante disuadir

1.10. Valor temporal del dinero [1 *, c5, c11]

Uno de los conceptos más fundamentales en
finanzas —y por lo tanto, en decisiones comerciales—
es que el dinero tiene valor temporal: su valor cambia
a través del tiempo. Una cantidad específica de dinero en este momento
casi siempre tiene un valor diferente al mismo
cantidad de dinero en otro momento. Este con-
cept ha estado presente desde el primer registro
historia humana y se conoce comúnmente como *tiempo-valor* . Para comparar propuestas o portfo-
lio elementos, deben normalizarse en costo,
valor y riesgo para el valor presente neto. Moneda
las variaciones de intercambio a lo largo del tiempo deben tomarse
en cuenta basado en datos históricos. Esto es par-
particularmente importante en desarrollos transfronterizos
De todo tipo.

1.11. Eficiencia [2 *, c1]

Eficiencia económica de un proceso, actividad o
tarea es la proporción de recursos realmente consumidos a
recursos que se espera sean consumidos o deseados
ser consumido en la realización del proceso, activ-
ity, o tarea. Eficiencia significa "hacer las cosas bien".
Un comportamiento eficiente, como un comportamiento efectivo,
ofrece resultados, pero mantiene el esfuerzo necesario para
un mínimo. Factores que pueden afectar la eficiencia en
ingeniería de software incluye complejo de producto
ity, requisitos de calidad, presión de tiempo, proceso
capacidad, distribución del equipo, interrupciones, función

mía cómo esas otras propuestas serían despreciables cio y cómo se estimarían las ganancias.

1.9. Impuestos

[1 *, c16, c17]

Los gobiernos cobran impuestos para financiar gastos que la sociedad necesita pero que ninguna organización privada nización invertiría en. Las empresas tienen que pagar impuestos sobre la renta, que pueden tomar una parte sustancial del beneficio bruto de una corporación. Un analisis de decisión que no tiene en cuenta los impuestos puede conducir a la Mala decisión. Una propuesta con un alto beneficio antes de impuestos no se verá tan rentable en términos posteriores a impuestos. No tener en cuenta los impuestos también puede llevar a Alisticamente altas expectativas sobre cuán rentable es un producto propuesto podría ser.

abandono, herramientas y lenguaje de programación.

1.12 Eficacia

[2 *, c1]

La efectividad se trata de tener impacto. Es el relación entre objetivos alcanzados para objetivos definidos La efectividad significa "hacer las cosas correctas ". La efectividad solo mira si se alcanzan los objetivos definidos, no en cómo se alcanzan

1.13 Productividad

[2 *, c23]

La productividad es la relación de salida sobre entrada desde Una perspectiva económica. La salida es el valor

entregado. La entrada cubre todos los recursos (p. Ej., Esfuerzo) gastado para generar la salida. Com productividad bines eficiencia y efectividad desde un valor perspectiva orientada: maximizando la productividad se trata de generar el valor más alto con el más bajo consumo de recursos.

2. Economía del ciclo de vida

2.1. Producto

[2 *, c22] [3 *, c6]

Un producto es un bien económico (o producto) que es creado en un proceso que transforma el producto factors (o entradas) a una salida. Cuando se vende, un producto uct es una entrega que crea tanto un valor como Una experiencia para sus usuarios. Un producto puede ser un combinación de sistemas, soluciones, materiales, y servicios entregados internamente (por ejemplo, en la empresa Solución de TI) o externamente (p. Ej., Aplicación de software) ción), ya sea como es o como un componente para otro producto (p. ej., software integrado).

2.2. Proyecto

[2 *, c22] [3 *, c1]

Un proyecto es "un esfuerzo temporal emprendido para crear un producto, servicio o resultado único ". [1] En ingeniería de software, diferentes tipos de proyectos. se distinguen (por ejemplo, desarrollo de productos, servicios subcontratados, mantenimiento de software, servicios vice creación, y así sucesivamente). Durante su ciclo de vida, un El producto de software puede requerir muchos proyectos. por ejemplo, durante la fase de concepción del producto, se podría realizar un proyecto para determinar el necesidad del cliente y requisitos del mercado; durante mantenimiento, un proyecto podría llevarse a cabo para producir una próxima versión de un producto.

2.3. Programa

Un programa es "un grupo de proyectos relacionados, subprogramas y actividades de programas gestionados en un forma coordinada de obtener beneficios no disponibles de administrarlos individualmente ". [2] Programas A menudo se utilizan para identificar y gestionar diferentes entregas a un solo cliente o mercado a través de un horizonte temporal de varios años.

2.4. portafolio

Las carteras son "proyectos, programas, subportfolios, y operaciones gestionadas como un grupo para lograr objetivos estratégicos. " [3] Las carteras se utilizan para agrupar y luego administrar simultáneamente todos los activos dentro de Una línea de negocio u organización. Mirando a un toda la cartera se asegura de que los impactos de la decise consideran las sesiones, como la asignación de recursos a un proyecto específico, lo que significa que el mismo los recursos no están disponibles para otros proyectos.

2.5. Ciclo de vida del producto

[2 *, c2] [3 *, c2]

Un ciclo de vida del producto de software (SPLC) incluye todas las actividades necesarias para definir, construir, operar, mantener y retirar un producto o servicio de software y sus variantes. Las actividades de SPLC de "opercomieron "," mantienen "y" retiran "generalmente ocurren en un marco de tiempo mucho más largo que el software inicial desarrollo (la vida de desarrollo de software cycle — SDLC — vea Software Life Cycle Model-Els en el proceso de ingeniería de software KA). También las actividades de operación-mantenimiento-retiro de un SPLC generalmente consume más esfuerzo total y otros recursos que las actividades de SDLC (ver La mayoría de costos de mantenimiento en el software Mantenimiento KA). El valor aportado por un producto de software o servicios asociados pueden ser determinado objetivamente durante el "operar y mantener "marco de tiempo. Ingeniería de software ecolos nomics deben preocuparse por todas las actividades de SPLC ities, incluidas las actividades posteriores al producto inicial lanzamiento.

2.6. Ciclo de vida del proyecto

[2 *, c2] [3 *, c2]

Las actividades del ciclo de vida del proyecto generalmente involucran cinco grupos de procesos (iniciación, planificación, ejecución) ing, Monitoreo y control, y cierre [4]

1 Project Management Institute, Inc., *Léxico PMI*

de Términos de Gestión de Proyectos, 2012, www.pmi.org/PMBOK-Guide-and-Standards / ~ / media / Registered / PMI_Lexicon_Final.ashx.

2 Ibid.

3 Ibid.

Page 213

(Ver el Software Engineering Management KA).
Las actividades dentro del ciclo de vida de un proyecto de software.
a menudo se entrelazan, superponen e iteran
de varias maneras [3 *, c2] [5] (vea el Software
Proceso de Ingeniería KA). Por ejemplo, ágil
El desarrollo de productos dentro de un SPLC implica
iteraciones múltiples que producen incrementos de
Software entregable. Un SPLC debe incluir
gestión de riesgos y sincronización con diferentes
proveedores diferentes (si los hay), mientras que proporcionan la propia
información adecuada para la toma de decisiones (por ejemplo, revisión
ing con necesidades de responsabilidad del producto o gobierno
regulaciones). El ciclo de vida del proyecto de software y
el ciclo de vida del producto de software está interrelacionado; un
SPLC puede incluir varios SDLC.

2.7. Propuestas

[1 *, c3]

Tomar una decisión comercial comienza con el
noción de una *propuesta* . Las propuestas se relacionan con la empresa
un objetivo comercial: en el proyecto, producto o
nivel de cartera Una propuesta es una sola, separada
opción que se está considerando, como llevar a cabo
un proyecto de desarrollo de software en particular o no.
Otra propuesta podría ser mejorar una existente.
ing componente de software, y aún otro poder
ser para volver a desarrollar ese mismo software desde cero.
Cada propuesta representa una unidad de elección, ya sea
puedes elegir llevar a cabo esa propuesta o
puede elegir no hacerlo. Todo el propósito de los negocios
la toma de decisiones es averiguar, dada la actual
circunstancias comerciales, qué propuestas deberían
llevarse a cabo y que no debería.

2.8. Decisiones de inversión

[1 *, c4]

Los inversores toman decisiones de inversión para gastar
dinero y recursos para lograr un objetivo objetivo
tive Los inversores están dentro (p. Ej., Finanzas,
junta) o fuera (p. ej., bancos) de la organización.
El objetivo se relaciona con algunos criterios económicos, como
como lograr un alto retorno de la inversión,
fortaleciendo las capacidades de la organización,
o mejorar el valor de la empresa. Intangi-
aspectos flexibles como la buena voluntad, la cultura y la conveniencia
las tenencias deben ser consideradas.

2.9. Planeando el horizonte

[1 *, c11]

Cuando una organización elige invertir en un par-
propuesta ticular, el dinero queda atado en ese pro-
posal: los llamados "activos congelados".
El impacto de los activos congelados tiende a comenzar alto y
disminuye con el tiempo. Por otro lado, opera-
costos de mantenimiento e ing de elementos asociados
con la propuesta tienden a comenzar bajo pero aumentar
con el tiempo. El costo total de la propuesta: que
es, poseer y operar un producto, es la suma
de esos dos costos. Al principio, costos de activos congelados
dominar; luego, la operación y mantenimiento
Los costos dominan. Hay un punto en el tiempo donde el
se minimiza la suma de los costos; esto se llama el
Costo mínimo de por vida .

Para comparar adecuadamente una propuesta con un cuatro
vida útil de un año a una propuesta con una vida de seis años
lapso, los efectos económicos de cortar el
propuesta de seis años por dos años o invirtiendo el
se beneficia de la propuesta de cuatro años para otro
Dos años deben ser abordados. La planificación
horizonte, a veces conocido como el período de estudio,
es el marco de tiempo consistente sobre el cual
También se tienen en cuenta. Efectos como la vida del software
habrá que tener en cuenta el tiempo para establecer un
planeando el horizonte. Una vez que el horizonte de planificación es
establecido, varias técnicas están disponibles para
poniendo propuestas con diferentes vidas en
ese horizonte de planificación.

2.10. Precio y precio

[1 *, c13]

Un precio es lo que se paga a cambio de un bien o
Servicio. El precio es un aspecto fundamental de las finanzas
modelado y es una de las cuatro Ps del marketing
mezcla. Las otras tres P son producto, promoción,
y lugar. El precio es el único elemento generador de ingresos.
ment entre las cuatro Ps; El resto son costos.

El precio es un elemento de las finanzas y el marketing.
Es el proceso de determinar lo que una empresa
recibirá a cambio de sus productos. Precios
los factores incluyen el costo de fabricación, el mercado
ment, competencia, condición del mercado y calidad
deprecio. Los precios aplican precios a productos y
servicios basados en factores como cantidad fija,
descanso por cantidad, promoción o campaña de ventas,

Página 214

presupuesto específico del proveedor, fecha de envío o factura,
parámetros utilizados para determinar si los programas,

combinación de múltiples pedidos, ofertas de servicios, y muchos otros. Las necesidades del consumidor pueden convertirse en demanda solo si el consumidor tiene la voluntad y la capacidad de comprar el producto uct. Por lo tanto, el precio es muy importante en el marketing. El precio se realiza inicialmente durante el inicio del proyecto. La fase de transición y es parte de la toma de decisiones "ir"

2.11. Costo y Costeo

[1 *, c15]

Un costo es el valor del dinero que se ha agotado. para producir algo y, por lo tanto, no está disponible para usar más. En economía, un costo es una alternativa nativo que se abandona como resultado de una decisión.

Un costo hundido son los gastos antes de cierto tiempo, típicamente utilizado para abstraer decisiones de gastos en el pasado, que pueden causar obstáculos para mirar hacia adelante. De un tradicional punto de vista económico, los costos hundidos no deberían ser considerado en la toma de decisiones. Oportunidad el costo es el costo de una alternativa que debe ser para-desaparecido para buscar otra alternativa.

El costeo forma parte de la gestión financiera y de producción ment. Es el proceso para determinar el costo basado en gastos (por ejemplo, producción, ingeniero de software) ing, distribución, retrabajo) y en el costo objetivo ser competitivo y exitoso en un mercado. El costo objetivo puede ser inferior al estimado real costo. La planificación y el control de estos costos. (llamado *gestión de costos*) es importante y debe siempre se incluirá en el costo.

Un concepto importante en el costeo es el costo total. de propiedad (TCO). Esto vale especialmente para software, porque hay muchos no tan obvios costos relacionados con las actividades de SPLC después de la adquisición. Desarrollo de uct. TCO para un producto de software es definido como el costo total de adquisición, activación, y mantener ese producto funcionando. Estos costos se pueden agrupar como costos directos e indirectos. TCO es un método de contabilidad que es crucial para hacer decisiones económicas acertadas.

2.12 Medición del desempeño

[3 *, c7, c8]

La medición del rendimiento es el proceso mediante el cual una organización establece y mide el resultados de inversiones y adquisiciones están logrando el resultados deseados. Se utiliza para evaluar si los objetivos de desempeño se logran realmente; a controlar presupuestos, recursos, progreso y decisión siones y para mejorar el rendimiento.

2.13. Gestion del valor ganado

[3 *, c8]

La gestión del valor ganado (EVM) es un proyecto técnica de gestión para medir el progreso basado en el valor creado. En un momento dado, el Los resultados logrados hasta la fecha en un proyecto son comparado con el presupuesto proyectado y el planificado Programar el progreso para esa fecha. El progreso se relaciona recursos ya consumidos y logrados resultados en un momento dado con el respeto tive valores planificados para la misma fecha. Ayuda para identificar posibles problemas de rendimiento en un Etapa temprana. Un principio clave en EVM es el seguimiento variaciones de costo y cronograma a través de la comparación de planificado versus calendario real y presupuesto versus costo real El seguimiento EVM ofrece visiones mucho más tempranas bility a las desviaciones y por lo tanto permite correcciones antes del costo clásico y el seguimiento del horario que solo analiza los documentos y productos entregados.

2.14. Decisiones de rescisión

[1 *, c11, c12] [2 *, c9]

Terminación significa finalizar un proyecto o producto. La terminación puede planificarse previamente para el final de un larga vida útil del producto (por ejemplo, cuando se prevé que un el producto alcanzará su vida útil) o puede venir más bien espontáneamente durante el desarrollo del producto (por ejemplo, cuando los objetivos de rendimiento del proyecto no son lograd). En ambos casos, la decisión debe ser cuidadosamente preparado, considerando siempre el alter- nativos de continuar versus terminar. Costo de se deben estimar diferentes alternativas (cobertura) Temas como el reemplazo, la recopilación de información selección, proveedores, alternativas, activos y utilidades ing recursos para otras oportunidades. Costos hundidos no debe ser considerado en tal toma de decisiones porque se han gastado y no cosecharán pera como valor.

Figura 12.4. Metas, Estimaciones y Planes

2.15. Decisiones de reemplazo y jubilación

[1 *, c12] [2 *, c9]

Se toma una decisión de reemplazo cuando una organización-
zation ya tiene un activo particular y son
considerando reemplazarlo con otra cosa; para
ejemplo, decidir entre mantener y apoyar
portar un producto de software heredado o volver a desarrollar-
desde cero. Uso de decisiones de reemplazo
el mismo proceso de decisión comercial descrito
arriba, pero hay desafíos adicionales: hundido
Costo y valor de recuperación. Las decisiones de jubilación son
también sobre salir de una actividad por completo,
como cuando una compañía de software considera que no
vender un producto de software más o un hardware
el fabricante considera no construir y vender un
modelo particular de computadora por más tiempo. Retirarse-
La decisión de la decisión puede verse influenciada por el hardware-
tors como dependencia tecnológica y alta salida
costos

3. Riesgo e incertidumbre

3.1. Metas, Estimaciones y Planes

[3 *, c6]

Los objetivos en la economía de la ingeniería de software son
principalmente objetivos comerciales (u objetivos comerciales):
Un objetivo comercial relaciona las necesidades comerciales (como
aumento de la rentabilidad) para invertir recursos
(como comenzar un proyecto o lanzar un producto-
uct con un presupuesto, contenido y tiempo dados).
Los objetivos se aplican a la planificación operativa (por ejemplo,
alcanzar un determinado hito en una fecha determinada o
extender las pruebas de software por algún tiempo para lograr un
nivel de calidad deseado: consulte Temas clave en Soft-
Ware Testing KA) y al nivel estratégico (como
como alcanzar una cierta rentabilidad o cuota de mercado
en un período de tiempo establecido).
Una estimación es una evaluación bien fundada de
recursos y tiempo que se necesitarán para lograr
metas establecidas (ver Esfuerzo, Calendario y Costo estimado)
mation en la Gerencia de Ingeniería de Software
Estimación de costos de mantenimiento y KA en el software
Mantenimiento de mercancías KA). Una estimación de software es
utilizado para determinar si los objetivos del proyecto pueden
ser alcanzado dentro de las limitaciones en el horario,
Presupuesto, características y atributos de calidad. Estimados
son típicamente generados internamente y no son
necesariamente visible externamente. Las estimaciones deberían
no ser impulsado exclusivamente por los objetivos del proyecto
porque esto podría hacer una estimación demasiado optimista
mistic La estimación es una actividad periódica; estimados
debe revisarse continuamente durante un proyecto.
Un plan describe las actividades y los hitos.
que son necesarios para alcanzar los objetivos de

un proyecto (consulte Planificación de proyectos de software en
Gerencia de Ingeniería de Software KA). los
el plan debe estar en línea con la meta y la estimación
mate, que no es necesariamente fácil y obvio
ous — como cuando un proyecto de software con dado
los requisitos tomarían más tiempo que el objetivo
fecha prevista por el cliente. En tales casos, los planes
exigir una revisión de los objetivos iniciales, así como la estimación-
compañeros y las incertidumbres subyacentes e inac-
curaciones Soluciones creativas con lo subyacente
Las razones para lograr una posición de ganar-ganar son
aplicado para resolver conflictos.

Para ser valioso, la planificación debe involucrar
consideración de las restricciones y el compromiso del proyecto-
Ment a los interesados. La figura 12.4 muestra cómo
Los objetivos se definen inicialmente. Las estimaciones están hechas
basado en los objetivos iniciales. El plan intenta coincidir
los objetivos y las estimaciones Este es un iterativo
proceso, porque una estimación inicial generalmente lo hace-
No cumplir con los objetivos iniciales.

3.2. Técnicas de estimación

[3 *, c6]

Las estimaciones se utilizan para analizar y pronosticar el
recursos o tiempo necesarios para implementar
mentos (ver Esfuerzo, Programa y Estimación de costos
en el Software Engineering Management KA
y estimación de costos de mantenimiento en el software

3.3. Abordar la incertidumbre

[3 *, c6]

Debido a los muchos factores desconocidos durante
iniciación y planificación del proyecto, las estimaciones son
inherentemente incierto; esa incertidumbre debería ser
abordado en las decisiones comerciales. Técnicas para
abordar la incertidumbre incluye

- considerar rangos de estimaciones
- analizar la sensibilidad a los cambios de supuestos
- retrasar las decisiones finales.

3.4. Priorización

[3 *, c6]

La priorización implica clasificar alternativas basadas
en criterios comunes para entregar el mejor posible
valor. En proyectos de ingeniería de software, software
los requisitos a menudo se priorizan para
entregar el mayor valor al cliente dentro de
limitaciones de horario, presupuesto, recursos y tecnología
nología, o para proporcionar el incremento de productos de construcción
ments, donde los primeros incrementos proporcionan el
valor más alto para el cliente (ver Requisitos
Clasificación y negociación de requisitos en
los requisitos de software KA y software
Modelos de ciclo de vida en la ingeniería de software
Proceso KA).

Mantenimiento KA). Cinco familias de estimación existen técnicas:

- Juicio experto
- analogía
- Estimación por partes.
- métodos paramétricos
- Métodos de estadística.

Ninguna técnica de estimación única es perfecta, así que Usar la técnica de estimación múltiple es útil. Convergencia entre las estimaciones producidas por diferentes técnicas indican que las estimaciones son probablemente precisos Difundido entre los esti- compañeros indica que ciertos factores podrían tener Ha sido pasado por alto. Encontrar los factores que causaron la propagación y luego volver a estimar nuevamente para pro- Duce resultados que convergen podrían conducir a un mejor estimar.

3.5. Decisiones bajo riesgo

[1 *, c24] [3 *, c9]

Las decisiones bajo técnicas de riesgo se utilizan cuando el tomador de decisiones puede asignar probabilidades a diferentes resultados posibles (ver Gestión de riesgos- ment en la Gestión de Ingeniería de Software KA). Las técnicas específicas incluyen

- toma de decisiones sobre el valor esperado
- variación de expectativas y toma de decisiones
- Análisis de Monte Carlo
- árboles de decisión
- valor esperado de información perfecta.

Página 217

Figura 12.5. El proceso de toma de decisiones con fines de lucro

3.6. Decisiones bajo incertidumbre

[1 *, c25] [3 *, c9]

Se utilizan decisiones bajo técnicas de incertidumbre. cuando el tomador de decisiones no puede asignar probabilidades se vincula con los diferentes resultados posibles porque la información necesaria no está disponible (ver Riesgo Gestión en el Software Engineering Man- agement KA). Las técnicas específicas incluyen

- Regla de Laplace
- Regla de Maximin
- Regla de Maximax

4. Métodos de análisis económico

4.1. Análisis de decisiones con fines de lucro

[1 *, c10]

La figura 12.5 describe un proceso para identificar La mejor alternativa de un conjunto de Sive alternativas. Los criterios de decisión dependen de objetivos comerciales y típicamente incluyen ROI (ver sección 4.3, Retorno de la inversión) o Retorno sobre Capital Empleado (ROCE) (ver sección 4.4, Rendimiento del capital invertido).

Las técnicas de decisión con fines de lucro no aplican

- Regla de Hurwicz
- Regla de arrepentimiento Minimax.

organizaciones gubernamentales y sin fines de lucro. En estos casos, las organizaciones tienen objetivos diferentes, que significa que un conjunto diferente de técnicas de decisión son necesarios, como costo-beneficio o costo-efectivo Análisis de ness.

Página 218

4.2. Tasa de retorno mínima aceptable

[1 *, c10]

La tasa de rendimiento mínima aceptable (MARR) es la tasa interna de retorno más baja que la organización zation consideraría una buena inversión. En términos generales, no sería inteligente invertir en una actividad con un retorno del 10% cuando hay Otra actividad que se sabe que devuelve el 20%. El MARR es una declaración de que una organización confía en que puede lograr al menos esa tasa de regreso. El MARR representa a la organización. costo de oportunidad para inversiones. Por elección para invertir en alguna actividad, la organización es decidiendo explícitamente no invertir ese mismo dinero en algún otro lugar. Si la organización ya está seguro de que puede obtener una tasa de rendimiento conocida, otras alternativas deben elegirse solo si su La tasa de rendimiento es al menos tan alta. Una manera simple para tener en cuenta ese costo de oportunidad es utilizar el MARR como la tasa de interés en las decisiones comerciales. El valor presente de una alternativa evaluado en el MARR muestra cuánto más o menos (en pres- términos en efectivo del día entrante) esa alternativa vale más que invirtiendo en el MARR.

4.3. Retorno de la inversión

[1 *, c10]

El retorno de la inversión (ROI) es una medida de rentabilidad de una empresa o unidad de negocio. Eso se define como la proporción de dinero ganado o perdido (ya sea realizado o no realizado) en una inversión en relación con la cantidad de dinero invertido. los El propósito del ROI varía e incluye, por ejemplo, proporcionando una justificación para futuras inversiones y decisiones de adquisición

4.4. Rendimiento del capital invertido

El rendimiento del capital empleado (ROCE) es una medida seguro de la rentabilidad de una empresa o negocio unidad. Se define como la relación de una ganancia bruta antes de impuestos e intereses (EBIT) a los activos totales menos pasivos corrientes. Describe el retorno de El capital utilizado.

4.5. Análisis coste-beneficio

[1 *, c18]

El análisis de costo-beneficio es uno de los más ampliamente métodos utilizados para evaluar propuestas individuales als. Cualquier propuesta con una relación costo-beneficio de menos que 1.0 generalmente se puede rechazar sin más análisis porque costaría más que el ben- efit. Las propuestas con una proporción más alta deben tener en cuenta considerar el riesgo asociado de una inversión y compare los beneficios con la opción de invertir el dinero a una tasa de interés garantizada (ver sección 4.2, Tasa de rendimiento mínima aceptable).

4.6. Análisis de costo-efectividad

[1 *, c18]

El análisis de costo-efectividad es similar al costo análisis de beneficios. Hay dos versiones de costo. análisis de efectividad: la versión de *costo fijo* maximiza el beneficio dado un límite superior en costo; la versión de *efectividad fija* minimiza El costo necesario para lograr un objetivo fijo.

4.7. Punto de equilibrio de analisis

[1 *, c19]

El análisis de equilibrio identifica el punto donde los costos de desarrollar un producto y los ingresos para ser generados son iguales. Tal análisis puede ser utilizado para elegir entre diferentes propuestas en Costos e ingresos estimados diferentes. Dado el estimado costos e ingresos asociados de dos o más propuestas También, el análisis de equilibrio ayuda a elegir entre ellos.

4.8. Caso de negocios

[1 *, c3]

El caso de negocio es la información consolidada. resumiendo y explicando una propuesta de negocio desde diferentes perspectivas para un tomador de decisiones (costo, beneficio, riesgo, etc.). Es de uso frecuente para evaluar el valor potencial de un producto, que puede usarse como base en la decisión de inversión proceso de fabricación. A diferencia de una mera ganancia cálculo de pérdidas, el caso de negocio es un "caso" de planes y análisis que son propiedad del producto

Page 219

gerente y utilizado en apoyo de lograr el objetivos de negocios.

4.9. Evaluación de atributos múltiples

[1 *, c26]

Los temas discutidos hasta ahora se utilizan para tomar decisiones. siones basadas en un criterio de decisión único: dinero. La alternativa con el mejor valor presente, el mejor ROI, y así sucesivamente es el seleccionado. Aparte de viabilidad técnica, el dinero es casi siempre el criterio de decisión más importante, pero No siempre es el único. Muy a menudo hay otros criterios, otros "atributos" que deben ser considerado, y esos atributos no se pueden lanzar Términos de dinero. Tecnología de decisión de atributos múltiples. las niques permiten que otros criterios no financieros sean factores investigó la decisión.

Hay dos familias de atributos múltiples. técnicas de decisión que difieren en cómo usan Los atributos en la decisión. Una familia es la Tecnología "compensatoria" o unidimensional niques Esta familia colapsa todos los atributos. en una sola figura de mérito. La familia se llama compensatorio porque, para cualquier alternativa dada, se puede compensar una puntuación más baja en un atributo por (o intercambiado) un puntaje más alto en otro atributos. Las técnicas compensatorias incluyen

- escalamiento no dimensional
- ponderación aditiva
- proceso de jerarquía analítica.

En contraste, la otra familia es la "no técnicas pensativas "o totalmente dimensionadas. Esta familia no permite compensaciones entre los atributos. Cada atributo se trata por separado entidad en el proceso de decisión. La no compensación técnicas tory incluyen

- dominio
- satisficing
- lexicografía.

4.10. Análisis de optimización

[1 *, c20]

El uso típico del análisis de optimización es estudiar una función de costo en un rango de valores para encuentre el punto donde el rendimiento general es el mejor. El compromiso clásico del espacio-tiempo del software es un ejemplo de optimización; un algoritmo que se ejecuta más rápido a menudo usará más memoria. Mejoramiento equilibra el valor del tiempo de ejecución más rápido contra El costo de la memoria adicional.

El análisis de opciones reales se puede usar para cuantificar El valor de las opciones del proyecto, incluido el valor de retrasar una decisión. Tales opciones son difíciles para calcular con precisión. Sin embargo, la conciencia que las opciones tienen un valor monetario proporciona conocimiento sobre el momento de las decisiones, como el aumento ing personal del proyecto o alargar el tiempo de comercialización mejorar calidad.

5. Consideraciones prácticas

5.1. El principio de "lo suficientemente bueno"

[1 *, c21]

A menudo, proyectos y productos de ingeniería de software. no son precisos sobre los objetivos que deberían ser logrado. Los requisitos de software están establecidos, pero el valor marginal de agregar un poco más de función No se puede medir la identidad. El resultado podría llegar tarde. entrega o costo demasiado alto. El "suficientemente bueno" principio relaciona el valor marginal con el costo marginal y proporciona orientación para determinar los criterios cuando un entregable es "suficientemente bueno" para ser entregado. Estos criterios dependen de los objetivos comerciales y en la priorización de diferentes alternativas, como requisitos de software de clasificación, calidad medible Atributos, o cronograma relacionado con el producto Carpa y costo.

El principio RACE (reducir accidentes y esencia de control) es una regla popular hacia el bien suficiente software Los accidentes implican innecesarios gastos generales tales como chapado en oro y retrabajo debido a la eliminación tardía de defectos o demasiados requisitos cambios La esencia es lo que pagan los clientes. Suave- la economía de la ingeniería de artículos proporciona el mech- mecanismos para definir criterios que determinan cuándo un entregable es "suficientemente bueno" para ser entregado. También destaca que ambas palabras son relevantes: "Bueno" y "suficiente". Calidad o cantidad insuficiente no es lo suficientemente bueno.

Los métodos ágiles son ejemplos de "suficientemente bueno" que intentan optimizar el valor reduciendo el exceso jefe de retrabajo retrasado y el chapado en oro que

resulta de agregar características que tienen un margen bajo valor ginal para los usuarios (ver Métodos ágiles en los modelos y métodos de ingeniería de software Modelos de ciclo de vida de software y KA en Soft- Proceso de Ingeniería de mercancías KA). En metanfetamina ods, planificación detallada y desarrollo prolongado las fases se reemplazan por planificación incremental y entrega frecuente de pequeños incrementos de una entrega producto que ha sido probado y evaluado por el usuario representantes.

5.2. Economía sin fricción

En un ecosistema típico, hay productores y consumidores, donde los consumidores agregan valor a Los recursos consumidos. Tenga en cuenta que un consumidor es no el usuario final sino una organización que usa el producto para mejorarlo. Un ecosistema de software es, por ejemplo, un proveedor de una aplicación trabajando con empresas que realizan la instalación y el soporte puerto en diferentes regiones. Ninguno de los dos podría existir sin el otro Los ecosistemas pueden ser permanentes o temporal. Economía de ingeniería de software proporciona los mecanismos para evaluar alternativas al establecer o extender un ecosistema, para

La fricción económica es todo lo que mantiene el mercado. Kets por tener una competencia perfecta. Implica distancia, costo de entrega, regulaciones restrictivas, y / o información imperfecta. En alta fricción mercados, los clientes no tienen muchos proveedores de donde elegir. Haber estado en un negocio por un tiempo o ser dueño de una tienda en una buena ubicación determina la posición económica. Es difícil para nuevos competidores para iniciar negocios y competir. El mercado se mueve lenta y previsiblemente. Los mercados libres de fricción son justo lo contrario. Nuevo surgen competidores y los clientes se apresuran a responder. El mercado es todo menos predecir poder. Teóricamente, el software y la TI son fricciones. gratis. Nuevas empresas pueden crear productos fácilmente y a menudo lo hacen a un costo mucho menor que el establecido empresas encuestadas, ya que no necesitan considerar cualquier legado. Se puede hacer marketing y ventas a través de Internet y redes sociales, y basi Los mecanismos de distribución gratuitos pueden permitir una subir a un negocio global. Ingeniero de software- La economía inglesa tiene como objetivo proporcionar bases para juzgar cómo funciona un negocio de software y cómo un mercado libre de fricción en realidad lo es. Por ejemplo, la competencia entre los desarrolladores de aplicaciones de software inhibido cuando las aplicaciones deben venderse a través de una tienda almacenar y cumplir con las reglas de esa tienda.

5.3. Ecosistemas

Un ecosistema es un entorno que consiste en todos las partes interesadas mutuamente dependientes, negocios unidades y empresas que trabajan en un área en particular.

ejemplo, evaluar si trabajar con un distribuidor específico o que la distribución sea realizada por un empresa que presta servicio en un área.

5.4. Deslocalización y Outsourcing

La deslocalización significa ejecutar una actividad comercial más allá de las ventas y el marketing fuera del hogar País de una empresa. Empresas típicamente o tienen sus ramas de deslocalización en baja países de costo o preguntan a empresas especializadas en el extranjero para ejecutar la actividad respectiva. Offshor- Por lo tanto, no debe confundirse con out- abastecimiento La deslocalización dentro de una empresa se llama deslocalización cautiva. El outsourcing es el resultado ori- relación establecida con un proveedor que ejecuta actividades comerciales para una empresa cuando, tra- opcionalmente, esas actividades se ejecutaron dentro la empresa. El outsourcing es independiente del sitio. El proveedor puede residir en el vecindario de la empresa o en el extranjero (subcontratación externa) En g). La economía de la ingeniería de software proporciona Los criterios básicos y las herramientas comerciales para evaluar diferentes mecanismos de abastecimiento y controlan sus actuación. Por ejemplo, usando un outsourcing proveedor de desarrollo de software y mantenimiento una empresa podría reducir el costo por hora de software desarrollo, pero aumenta el número de horas y gastos de capital debido a una mayor necesidad de monitoreo y comunicación. (Para más información En relación con la deslocalización y la subcontratación, ver "Out- abastecimiento "en Problemas de gestión en el Software Mantenimiento KA.)

Page 221

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	05		**09**
	ey 20 [1 *]	**ile 2011 erv [2 *]**	**[3 *]**
	ock T	**metro gm S**	**airley 20 F**
1. Economía de la ingeniería de software Fundamentos			
1.1. Financiar	c2		
1.2. Contabilidad	c15		
1.3. Controlador	c15		
1.4. Flujo de fondos	c3		
1.5. Proceso de toma de decisiones	c2, c4		
1.6. Valuación	c5, c8		
1.7. Inflación	c13		
1.8. Depreciación	c14		
1.9. Impuestos	c16, c17		
1.10. Valor temporal del dinero	c5, c11		
1.11. Eficiencia		c1	

1.12 Eficacia		c1	
1.13. Productividad		c23	
2. Economía del ciclo de vida			
2.1. Producto		c22	c6
2.2. Proyecto		c22	c1
2.3. Programa			
2.4. portafolio			
2.5. Ciclo de vida del producto		c2	c2
2.6. Ciclo de vida del proyecto		c2	c2
2.7. Propuestas	c3		
2.8. Decisiones de inversión	c4		
2.9. Planeando el horizonte	c11		
2.10. Precio y precio	c13		
2.11. Costo y Costeo	c15		
2.12 Medición del desempeño			c7, c8
2.13. Gestion del valor ganado			c8
2.14. Decisiones de rescisión	c11, c12	c9	
2.15. Decisiones de reemplazo y jubilación	c12	c9	

Página 222

	05			**09**
	ey 20 [1 *]	**ile 2011**		
	ock	**erv [2 *]**		**[3 *]**
	T	**metro**		**airley 20**
		om		**F**
		S		
3. Riesgo e incertidumbre				
3.1. Metas, Estimaciones y Planes				c6
3.2. Técnicas de estimación				c6
3.3. Abordar la incertidumbre				c6
3.4. Priorización				c6
3.5. Decisiones bajo riesgo	c24			c9
3.6. Decisiones bajo incertidumbre	c25			c9
4. Métodos de análisis económico				
4.1. Análisis de decisiones con fines de lucro	c10			
4.2. Tasa de retorno mínima aceptable	c10			
4.3. Retorno de la inversión	c10			
4.4. Rendimiento del capital invertido				
4.5. Análisis coste-beneficio	c18			
4.6. Análisis de costo-efectividad	c18			
4.7. Punto de equilibrio de analisis	c19			
4.8. Caso de negocios	c3			
4.9. Evaluación de atributos múltiples	c26			
4.10. Análisis de optimización	c20			
5. Consideraciones prácticas				
5.1. El principio de "lo suficientemente bueno"	c21			
5.2. Economía sin fricción				
5.3. Ecosistemas				
5.4. Deslocalización y Outsourcing				

Página 223

LECTURAS ADICIONALES

Una guía para el organismo de gestión de proyectos de Conocimiento (Guía PMBOK®) [4].

La *guía PMBOK®* proporciona pautas para gestiona proyectos individuales y define proyecto conceptos relacionados con la gestión. También describe el ciclo de vida de la gestión de proyectos y sus relacionados procesos, así como el ciclo de vida del proyecto. Es una guía reconocida a nivel mundial para el proyecto man- profesión de agente.

Extensión de software a la guía del proyecto Cuerpo de conocimiento de gestión (SWX) [5].

SWX proporciona adaptaciones y extensiones a la prácticas genéricas de gestión de proyectos docu- mencionado en la *Guía PMBOK®* para gestionar proyectos de software. La principal contribución de esta extensión de la *Guía PMBOK®* es descriptiva ción de procesos que son aplicables para gestionar proyectos de software de ciclo de vida adaptativo.

BW Boehm, *Economía de la Ingeniería del Software* [6]

Este libro es la lectura clásica sobre software. economía de la ingeniería Proporciona una visión general. del pensamiento empresarial en ingeniería de software. Aunque los ejemplos y las cifras están fechados, Todavía vale la pena leerlo.

C. Ebert y R. Dumke, *Medición de software* [7]

Este libro proporciona una visión general sobre la cuantificación tive métodos en ingeniería de software, comenzando con la teoría de la medición y proceder a gestión del desempeño y decisión comercial fabricación.

DJ Reifer, *Making the Software Business Case: Mejora por los números* [8].

Este libro es una lectura clásica sobre cómo hacer un negocio. ness case en las empresas de software y TI. Muchos ejemplos útiles ilustran cómo el caso de negocios está formulado y cuantificado.

Referencias

[1 *] S. Tockey, rendimiento *del software: maximización el retorno de su inversión en software* , Addison-Wesley, 2004.

[2 *] JH Allen et al., *Seguridad de software Ingeniería: una guía para el proyecto Gerentes* , Addison-Wesley, 2008.

[3 *] RE Fairley, *Gestión y liderazgo Proyectos de software* , computadora Wiley-IEEE Society Press, 2009.

[4] Project Management Institute, *una guía al Órgano de Gestión del Proyecto de Conocimiento (Guía PMBOK (R))* , 5ª ed., Instituto de Gestión de Proyectos, 2013.

[5] Instituto de Gestión de Proyectos e IEEE Sociedad de Computación, *Extensión de Software a la Quinta Edición de la Guía PMBOK®* , ed .: Instituto de Gestión de Proyectos, 2013.

[6] BW Boehm, *Ingeniería de software Economía* , Prentice-Hall, 1981.

[7] C. Ebert y R. Dumke, *Software Medición* , Springer, 2007.

[8] DJ Reifer, *haciendo el negocio del software Caso: mejora por los números* , Addison Wesley, 2002.

CAPITULO 13

FUNDAMENTOS INFORMÁTICOS

SIGLAS

AOP	Programación Orientada a Aspectos
ALU	Unidad aritmética lógica
API	Programación de aplicaciones Interfaz
Cajero automático	Máquina de Transferencia Asíncrona Modo
B / S	Navegador-Servidor
CERT	Respuesta de emergencia informática Equipo
Cunas	Comercial fuera de la plataforma
CRUDO	Crear, leer, actualizar, eliminar
C / S	Servidor de cliente
CS	Ciencias de la Computación
DBMS	Sistema de administración de base de datos
FPU	Unidad de punto flotante
I / O	Entrada y salida
ES UN	Set de instrucciones arquitectura
YO ASI	Organización internacional para Normalización
ISP	Proveedor de servicios de Internet
LAN	Red de área local
MUX	Multiplexor
NIC	Tarjeta de interfaz de red
OOP	Programación orientada a objetos
OS	Sistema operativo
OSI	Sistemas abiertos de interconexión
ordenador personal	Computadora personal
PDA	Asistente personal digital
PPP	Protocolo punto a punto
RFID	Identificación de frecuencia de radio
RAM	Memoria de acceso aleatorio
ROM	Memoria de sólo lectura
SCSI	Interfaz de sistema de computadora pequeña
SQL	lenguaje de consulta estructurado
TCP	Protocolo de control de transporte
UDP	Protocolo de datagramas de usuario
VPN	Red privada virtual
PÁLIDO	Red de área amplia

INTRODUCCIÓN

El alcance del conocimiento de las Fundaciones de Computación área de borde (KA) abarca el desarrollo y entorno operativo en el que el software evoluciona y se ejecuta. Porque ningún software puede existir en el vacío o funcionar sin una computadora, el núcleo de dicho entorno es la computadora y Sus diversos componentes. Conocimiento sobre el computadora y sus principios subyacentes de hard- El software y el software sirven como marco para qué ingeniería de software está anclada. Por lo tanto, todos los ingenieros de software deben tener una buena comprensión ing de las Fundaciones de Computación KA.

En general se acepta que el software engi- Neering se basa en la informática. por ejemplo, “Ingeniería de software 2004: Cur- Pautas de Riculum para Licenciatura Programas en Ingeniería de Software ”[1] claramente declara: "Un aspecto particularmente importante es que *la ingeniería de software se basa en la informática* y matemáticas "(cursiva agregada).

Steve Tockey escribió en su libro *Return on Software* :

> Tanto la informática como la ingeniería del software trato de neering con computadoras, computación, y software. La ciencia de la computación, como un cuerpo de conocimiento, está en el centro de ambos.

Figura 13.1. Desglose de Temas para las Fundaciones de Computación KA

> ... La ingeniería de software se preocupa por la aplicación de computadoras, computación, y software para fines prácticos, específicos En términos generales, el diseño, la construcción y el funcionamiento, software eficiente y económico sistemas.

Por lo tanto, en el núcleo de la ingeniería de software es un comprensión de la informática.

Mientras que pocas personas negarán el rol de computadora la ciencia juega en el desarrollo de software ingeniería tanto como disciplina y como cuerpo de conocimiento, la importancia de la informática a la ingeniería de software no se puede exagerar tamaño así, este KA de Fundamentos de Computación es siendo escrito

La mayoría de los temas discutidos en el Comité Poner Fundamentos KA también son temas de discusión. sión en cursos básicos impartidos en informática programas de pregrado y posgrado. Tal los cursos incluyen programación, estructura de datos, algoritmos, organización informática, funcionamiento sistemas, compiladores, bases de datos, redes, dis- sistemas tributarios, y así sucesivamente. Por lo tanto, cuando se compone Al descifrar temas, puede ser tentador descomponer los fundamentos de computación KA de acuerdo con estas divisiones a menudo encontradas en cursos relevantes.

Sin embargo, una división puramente basada en el curso de Los temas sufren serios inconvenientes. Por un lado, no todos los cursos de informática están relacionados o igualmente importante para la ingeniería de software. Así, algunos temas que de otro modo estarían cubiertos en un curso de ciencias de la computación no están cubiertos en esta KA. Por ejemplo, gráficos por computadora, mientras que un curso importante en una licenciatura en informática programa: no está incluido en este KA.

Segundo, algunos temas discutidos en esta guía: la línea no existe como cursos independientes en sub- programas de posgrado o posgrado en ciencias de la computación. En consecuencia, tales temas pueden no ser adecuadamente cubierto en un desglose puramente basado en el curso. por ejemplo, la abstracción es un tema incorporado en varios cursos de informática diferentes; está no está claro a qué curso debe pertenecer la abstracción en un desglose de temas basado en cursos.

Las Fundaciones de Computación KA se dividen en diecisiete temas diferentes. La utilidad directa de un tema El criterio utilizado para los ingenieros de software es seleccionar temas para incluir en este KA (ver Figura 13.1). La ventaja de este desglose basado en temas es su fundamento en la creencia de que Computing Foun- las fechas, si se quiere comprender firmemente, deben ser consideradas echado a un lado como una colección de temas conectados lógicamente apuntalar la ingeniería de software en general y construcción de software en particular.

Las Fundaciones de Computación KA están relacionadas este componente al Diseño de Software, Software- estructura, prueba de software, software principal tenencia, calidad de software y matemática Fundaciones KAs.

DESGLOSE DE TEMAS PARA FUNDAMENTOS INFORMÁTICOS

El desglose de temas para la informática Fundamentos KA se muestra en la Figura 13.1.

Página 226

1. Técnicas de resolución de problemas [2 *, s3.2, c4] [3 *, c5]

Los conceptos, nociones y terminología introducidos aquí forman una base subyacente para la comprensión El papel y el alcance de las técnicas de resolución de problemas.

1.1. Definición de resolución de problemas

La resolución de problemas se refiere al pensamiento y la actividad vínculos realizados para responder o derivar una solución a un problema. Hay muchas formas de acercarse a un problema, y cada manera emplea diferentes herramientas y utiliza diferentes procesos. Estos diferentes formas de abordar los problemas se expanden gradualmente y definirse y finalmente dar lugar a diferencias diferentes disciplinas Por ejemplo, el software de ingeniería Neering se enfoca en resolver problemas usando computadoras y software.

Mientras que diferentes problemas justifican diferentes soluciones y pueden requerir diferentes herramientas y procesos, metodología y técnicas utilizadas

1.3. Analiza el problema

Una vez que el enunciado del problema está disponible, el siguiente el paso es analizar el enunciado del problema o la situación Acción para ayudar a estructurar nuestra búsqueda de una solución. Cuatro tipos de análisis incluyen *análisis de situación,* en el que los aspectos más urgentes o críticos de un la situación se identifica primero; *análisis de problemas,* en cuál la causa del problema debe ser disuadida mined; *análisis de decisiones,* en el que la (s) acción (es) necesario para corregir el problema o eliminar su la causa debe ser determinada; y *problema potencial análisis,* en el cual las acciones necesarias para prevenir cualquier recurrencia del problema o el desarrollo Se deben determinar los nuevos problemas.

1.4. Diseñar una estrategia de búsqueda de soluciones

Una vez que se completa el análisis del problema, podemos centrarse en estructurar una estrategia de búsqueda para encontrar el solución. Para encontrar la "mejor" solución (aquí, "Mejor" podría significar cosas diferentes para diferentes

para resolver problemas, siga algunas pautas y a menudo se puede generalizar como resolución de problemas técnicas Por ejemplo, una directriz general para resolver un problema genérico de ingeniería es usar El proceso de tres pasos que figura a continuación [2 *].

- Formular el verdadero problema.
- Analizar el problema.
- Diseñar una estrategia de búsqueda de soluciones.

1.2. Formulando el problema real

Gerard Voland escribe: "Es importante reconocer Tenga en cuenta que se debe formular un problema específico si se trata de desarrollar una solución específica "[2 *]. Esta formulación se llama enunciado del problema, que especifica explícitamente cuál es el problema y el resultado deseado son.

Aunque no hay una forma universal de estadística Si hay un problema, en general un problema debería ser expresado de tal manera que facilite el desarrollo Opción de soluciones. Algunas técnicas generales para ayudar a formular el problema real incluyen declaración-reexpresión, determinando la fuente y la causa, revisando la declaración, analizando estado presente y deseado, y usando el ojo fresco enfoque.

personas, como más rápido, más barato, más utilizable, diferentes capacidades, etc.), necesitamos eliminar caminos que no conducen a soluciones viables, diseño tareas de una manera que proporciona la mayor orientación en buscar una solución y usar varios atributos del estado de la solución final para guiar nuestras elecciones en El proceso de resolución de problemas.

1.5. Resolución de problemas usando programas

La singularidad del software da problemas lem resolviendo un sabor que es distinto del general resolución de problemas de ingeniería. Resolver un problema usando computadoras, debemos responder lo siguiente preguntas

- ¿Cómo averiguamos qué decirle a la empresa? ¿Qué hacer?
- ¿Cómo convertimos el enunciado del problema? en un algoritmo?
- ¿Cómo convertimos el algoritmo en instrucciones de la máquina?

La primera tarea para resolver un problema usando un equipo lo más importante es determinar qué decirle a la computadora hacer. Puede haber muchas formas de contar la historia, pero todos deberían tomar la perspectiva de una computadora como

Página 227

que la computadora eventualmente puede resolver el problema lem. En general, un problema debe expresarse de tal manera que facilite el desarrollo de algoritmos y estructuras de datos para resolverlo.

El resultado de la primera tarea es un problema ment. El siguiente paso es convertir el estado del problema. ment en algoritmos que resuelven el problema. Una vez se encuentra un algoritmo, el paso final convierte el algoritmo en instrucciones de máquina que forman el solución final: software que resuelve el problema.

En términos abstractos, la resolución de problemas utilizando la computadora puede considerarse como un proceso de problema transformación de lem, en otras palabras, el paso a paso paso de transformación de una declaración de problema en Una solución del problema. A la disciplina del software ingeniería, el objetivo final del problema resolver es transformar un problema expresado en lenguaje natural en electrones corriendo Un circuito. En general, esta transformación puede ser dividido en tres fases:

a) Desarrollo de algoritmos a partir del problema declaración de lem.
b) Aplicación de algoritmos al problema.
c) Transformación de algoritmos a programa código.

La conversión de una declaración de problema en algoritmos y algoritmos en códigos de programa generalmente sigue un "refinamiento gradual" (también conocido como descomposición sistemática) en la que comenzamos con un enunciado del problema, reescríbalo como una tarea, y descomponer recursivamente la tarea en unos pocos subtareas más simples hasta que la tarea sea tan simple que Las soluciones son sencillas. Hay tres formas básicas de descomposición: secuencial, condi-

"A través de la abstracción", según Voland, "Vemos el problema y su posible solución caminos desde un nivel superior de subconcepto conceptual en pie. Como resultado, podemos ser mejores antes recortado para reconocer posibles relaciones entre diferentes aspectos del problema y, por lo tanto, genborrar soluciones de diseño más creativas "[2 *]. Esta es particularmente cierto en informática en general (como hardware frente a software) y en software ingeniería en particular (estructura de datos vs. datos flujo, y así sucesivamente).

2.1. Niveles de abstracción

Al abstraernos, nos concentramos en un "nivel" del panorama general a la vez con la confianza de que entonces podemos conectarnos efectivamente con niveles superiores y por debajo. Aunque nos centramos en un nivel, abstracción no significa no saber nada sobre Los niveles vecinos. Los niveles de abstracción no necesariamente corresponde a componentes discretos en realidad o en el dominio del problema, pero bien interfaces estándar definidas como la programación APIs. Las ventajas que tienen las interfaces estándar proporcionar incluir portabilidad, software más fácil / hardintegración de software y uso más amplio.

2.2. Encapsulamiento

La encapsulación es un mecanismo utilizado para implementar abstracción mental. Cuando estamos tratando con uno nivel de abstracción, la información relativa a los niveles por debajo y por encima de ese nivel es encapsuLated Esta información puede ser el concepto, el problema lem, o fenómeno observable; o puede ser el operaciones permitidas en estas entidades relevantes.

tional e iterativo.

2. abstracción

[3 *, s5.2–5.4]

La abstracción es una técnica indispensable indispensable. ayudado con la resolución de problemas. Se refiere tanto a la proceso y resultado de la generalización al reducir la información de un concepto, un problema o un fenómeno observable para que uno pueda concentrarse en el "panorama general". Uno de los más importantes habilidades en cualquier empresa de ingeniería está enmarcando los niveles de abstracción adecuadamente.

La encapsulación generalmente viene con cierto grado de información oculta en la que algunos o todos los detalles subyacentes están ocultos del nivel encima de la interfaz proporcionada por la abstracción. Para un objeto, ocultar información significa que no necesita saber los detalles de cómo se representa el objeto resentido o cómo las operaciones en esos objetos son implementados.

2.3. Jerarquía

Cuando usamos la abstracción en nuestra fórmula del problema ción y solución, podemos usar diferentes abstracciones

Página 228

en diferentes momentos, en otras palabras, trabajamos en diferentes diferentes niveles de abstracción según lo requiera la situación. La mayoría de las veces, estos diferentes niveles de abstracción se organizan en una jerarquía. Hay muchos formas de estructurar una jerarquía particular y la criterios utilizados para determinar el contenido específico de cada capa en la jerarquía varía según el individuos que realizan el trabajo.

A veces, una jerarquía de abstracción es una secuencia. tial, lo que significa que cada capa tiene una y solo una capa predecesora (inferior) y una y solo una capa sucesora (superior), excepto la capa superior capa (que no tiene sucesor) y la más inferior capa (que no tiene predecesor). A veces, sin embargo, la jerarquía está organizada en forma de árbol estructura, lo que significa que cada capa puede tener más de una capa predecesora pero solo una sucesora capa. Ocasionalmente, una jerarquía puede tener muchas estructura de muchos, en la que cada capa puede tener Múltiples predecesores y sucesores. En ningún momento, debe haber algún bucle en una jerarquía.

Una jerarquía a menudo se forma naturalmente en la descomposición de la descomposición. A menudo, un análisis de tareas puede descomponerse de manera jerárquica, comenzando con el más grande tareas y objetivos de la organización y ruptura cada uno de ellos en subtareas más pequeñas que pueden nuevamente se subdividirá más Esta división continua Sión de tareas en las más pequeñas produciría una estructura jerárquica de tareas-subtareas.

2.4. Abstracciones Alternas

A veces es útil tener múltiples alternativas abstracciones para el mismo problema para que uno pueda tenga en cuenta diferentes perspectivas. Para examen- ple, podemos tener un diagrama de clase, un gráfico de estado y un diagrama de secuencia para el mismo software en el mismo nivel de abstracción. Estos alternan las abstracciones no forman una jerarquía sino más bien se complementan para ayudar a comprender El problema y su solución. Aunque beneficioso, es como tiempos difíciles de mantener abstracciones alternativas en sintonía.

3. Fundamentos de programación

[3 *, c6-19]

La programación se compone de las metodologías. o actividades para crear programas de computadora que realizan una función deseada. Es indispensable parte en la construcción de software. En general, programática puede considerarse como el proceso de diseño, escritura, prueba, depuración y mantenimiento conteniendo el código fuente. Este código fuente está escrito diez en un lenguaje de programación.

El proceso de escribir código fuente a menudo requiere experiencia en muchos temas diferentes áreas, incluido el conocimiento de la aplicación dominio, estructuras de datos apropiadas, especiales algoritmos ized, varias construcciones de lenguaje, buenas técnicas de programación y software Ingenieria.

3.1. El proceso de programación

La programación implica diseño, escritura, pruebas, depuración y mantenimiento. *El diseño* es el aceptación o invención de un esquema para convertir un requisito del cliente para software de computadora en software operativo Es la actividad que une composición de aplicación para codificación y depuración ging. *Escribir* es la codificación real del diseño en un lenguaje de programación apropiado. *Pruebas* es la actividad para verificar que el código que uno escribe en realidad hace lo que se supone que debe hacer. *Depurar-ging* es la actividad para encontrar y corregir errores (fallas) en El código fuente (o diseño). *El mantenimiento* es el actividad para actualizar, corregir y mejorar las existentes programas Cada una de estas actividades es un gran tema. y a menudo garantiza la explicación de todo KA en la *Guía SWEBOK* y muchos libros.

3.2. Paradigmas de programación

La programación es altamente creativa y, por lo tanto, que personal Diferentes personas a menudo escriben diferentes diferentes programas para los mismos requisitos. Esta la diversidad de programación causa mucha dificultad en la construcción y mantenimiento de grandes software complejo Varios parámetros de programación digms se han desarrollado a lo largo de los años para poner cierta estandarización en este altamente creativo y actividad personal Cuando uno programa, él o ella puede usar uno de varios paradigmas de programación para Escribe el código. Los principales tipos de programación. Los paradigmas se analizan a continuación.

Programación desestructurada *:* en desestructurada programación, un programador sigue su

Página 229

presentimiento para escribir el código de cualquier manera que lo haga. Me gusta siempre que la función esté operativa. A menudo, la práctica es escribir código para cumplir con un específico utilidad sin tener en cuenta nada más. Programas escrito de esta manera no exhibe ninguna estructura particular de ahí el nombre de "programación no estructurada". La programación no estructurada también es a veces llamado programación ad hoc.

Programa Estructurado / Procesal / Imperativo ming: un sello distintivo de la programación estructurada es El uso de estructuras de control bien definidas, incluyendo ing procedimientos (y / o funciones) con cada procedure (o función) realizando una tarea específica. Existen interfaces entre procedimientos para facilitar operaciones de llamada correctas y sin problemas del programos Bajo programación estructurada, programa- Los clientes a menudo siguen los protocolos y reglas establecidos de pulgar al escribir código. Estos protocolos y las reglas pueden ser numerosas y abarcar casi todo el alcance de la programación, desde problema más simple (como cómo nombrar variables, funciones, procedimientos, etc.) a más componentes problemas complejos (como cómo estructurar una interfaz, cómo manejar excepciones, y así sucesivamente).

Programación Orientada a Objetos: Mientras continúa- la programación dural organiza programas alrededor procedimientos, programación orientada a objetos (OOP) organizar un programa alrededor de objetos, que son estructuras de datos abstractas que combinan ambos datos y métodos utilizados para acceder o manipular el datos. Las características principales de OOP son que los objetos representando varias entidades abstractas y concretas son creados y estos objetos interactúan con cada uno otro para cumplir colectivamente las funciones deseadas.

Programación orientada a aspectos *:* aspecto-ori- La programación ented (AOP) es una programación paradigma que se construye sobre OOP. AOP apunta para aislar funciones secundarias o de apoyo de la lógica de negocios del programa principal al enfocarse en las secciones transversales (preocupaciones) de los objetos. La motivación principal para AOP es resolver el objeto enredado y dispersión asociado con OOP, en el que las interacciones entre objetos se vuelve muy complejo. La esencia de AOP es la separación de preocupaciones muy enfatizada, que separa las preocupaciones funcionales no esenciales o lógica en varios aspectos.

Programación funcional : aunque menos popu- Lar, la programación funcional es tan viable como los otros paradigmas en la resolución de la programación problemas. En la programación funcional, todos los las putaciones se tratan como la evaluación de las matemáticas funciones ematicas. En contraste con el imperativo programación que enfatiza cambios en el estado, la programación funcional enfatiza la aplicación ción de funciones, evita estado y datos mutables, y proporciona transparencia referencial.

4. Conceptos básicos del lenguaje de programación [4 *, c6]

Usar computadoras para resolver problemas implica programación, que es escritura y organización Instrucciones que le dicen a la computadora qué hacer en cada paso Los programas deben estar escritos en alguna lenguaje de programación con el cual ya través qué describimos los cálculos necesarios. En en otras palabras, utilizamos las instalaciones proporcionadas por un lenguaje de programación para describir problemas, desarrollar algoritmos y razonar sobre el problema soluciones Para escribir cualquier programa, uno debe soportar al menos un lenguaje de programación.

4.1. Descripción general del lenguaje de programación

Un lenguaje de programación está diseñado para expresar cálculos que puede realizar una empresa puter En un sentido práctico, un lenguaje de programación guage es una notación para escribir programas y por lo tanto debería poder expresar la mayoría de las estructuras de datos y algoritmos Algunas, pero no todas, las personas restringen término "lenguaje de programación" a esos lenguajes que puede expresar todos los algoritmos posibles.

No todos los idiomas tienen la misma importancia. y popularidad. Los más populares son a menudo definido por un documento de especificación establecido por una organización conocida y respetada. por ejemplo, el lenguaje de programación C es especi cumplido por un estándar ISO llamado ISO / IEC 9899. Otros lenguajes, como Perl y Python, no disfruta de tal tratamiento y a menudo tiene un dominante implementación que se utiliza como referencia.

4.2. Sintaxis y Semántica de Programación Idiomas

Al igual que los lenguajes naturales, muchas programaciones los idiomas tienen alguna forma de especificación escrita ción de su sintaxis (forma) y semántica (media- En g). Dichas especificaciones incluyen, por ejemplo,

230 de 1189.

requisitos específicos para la definición de vari ables y constantes (en otras palabras, declaración y tipos) y requisitos de formato para instrucciones en sí.

4.4. Lenguajes de programación de alto nivel

Un lenguaje de programación de alto nivel tiene un fuerte abstracción de los detalles de la computadora

En general, un lenguaje de programación soporta construcciones tales como variables, tipos de datos, constantes, literales, declaraciones de asignación, control declaraciones, procedimientos, funciones y comentarios. La sintaxis y la semántica de cada construcción deben estar claramente especificado

4.3. Lenguajes de programación de bajo nivel

El lenguaje de programación se puede clasificar en dos clases: idiomas de bajo nivel y lenguaje de alto nivel medidores Se pueden entender idiomas de bajo nivel por una computadora con o sin asistencia mínima y normalmente incluyen lenguajes de máquina y ensamblaje bly languages. Un lenguaje de máquina usa unos y ceros para representar instrucciones y variables, y es directamente comprensible por una computadora. Un lenguaje ensamblador contiene las mismas instrucciones como lenguaje de máquina pero las instrucciones y las variables tienen nombres simbólicos que son más fáciles para humanos para recordar.

Los lenguajes de ensamblaje no pueden estar directamente habilitados parado junto a una computadora y debe ser traducido a un lenguaje de máquina por un programa de utilidad llamado *ensamblador.* A menudo existe una correspondencia entre las instrucciones de un lenguaje ensamblador y un lenguaje de máquina, y la traducción de el código de ensamblaje al código de máquina es sencillo sala. Por ejemplo, "agregar r1, r2, r3" es un ensamblado instrucciones bly para agregar el contenido del registro r2 y r3 y almacenar la suma en el registro r1. Esta la instrucción se puede traducir fácilmente a la máquina código "0001 0001 0010 0011 **.** "(Supongamos que el El código de acción para la adición es 0001, consulte la Figura 13.2).

añadir	r1,	r2,	r3
0001	0001	0010	0011

Figura 13.2. Traducciones de ensamblado a binario

Un rasgo común compartido por estos dos tipos del lenguaje es su estrecha asociación con el detalles de un tipo de computadora o conjunto de instrucciones arquitectura (ISA).

ES UN. En comparación con la programación de bajo nivel. idiomas, a menudo usa elementos de lenguaje natural ments y por lo tanto es mucho más fácil para los humanos entender. Tales lenguajes permiten nombres simbólicos. ing de variables, proporcionar expresividad, y permitir la abstracción del hardware subyacente. Por ejemplo, mientras cada microprocesador tiene su ISA propio, código escrito en un programa de alto nivel El lenguaje ming suele ser portátil entre muchos diferentes plataformas de hardware Por estas razones, la mayoría de los programadores usan y la mayoría del software son escrito en lenguajes de programación de alto nivel. Ejemplos de lenguajes de programación de alto nivel. incluyen C, C ++, C # y Java.

4.5. Programación declarativa versus programación imperativa Idiomas

La mayoría de los lenguajes de programación (de alto o bajo nivel) nivel) permite a los programadores especificar el indiinstrucciones visuales que una computadora debe ejecutar. Dichos lenguajes de programación se denominan imperative lenguajes de programación porque uno tiene que especifique cada paso claramente a la computadora. Pero algunos lenguajes de programación permiten el programa solo describe la función que se va a realizar formado sin especificar la instrucción exacta secuencias a ejecutar. Tal programación los lenguajes se llaman programación declarativa idiomas Los lenguajes declarativos son de alto nivel. idiomas La implementación real de la el cálculo escrito en dicho lenguaje está oculto de los programadores y por lo tanto no es una preocupación para ellos.

El punto clave a tener en cuenta es que la declaración declarativa gramática solo describe *lo que* el programa debe lograr sin describir *la forma en* que lograrlo Por esta razón, mucha gente cree que la programación declarativa facilita Desarrollo de software más fácil. Pro declarativa los lenguajes de gramática incluyen Lisp (también una función lenguaje de programación nacional) y Prolog, mientras los lenguajes de programación imperativos incluyen C, C ++ y JAVA.

Página 231

5. Herramientas y técnicas de depuración [3 *, c23]

Una vez que un programa está codificado y compilado (compilación- se discutirá en la sección 10), el siguiente paso es depuración, que es un proceso metódico de encontrar y reducir el número de errores o fallas en un programa El propósito de la depuración es encontrar averiguar por qué un programa no funciona o produce un Resultado o resultado incorrecto. Excepto por muy simple programas, la depuración siempre es necesaria.

5.1. Tipos de errores

Cuando un programa no funciona, a menudo es porque el programa contiene errores o errores que pueden ser ya sea errores sintácticos, errores lógicos o errores de datos. Los errores lógicos y los errores de datos también se conocen como

5.2. Técnicas de depuración

La depuración implica muchas actividades y puede ser estática, dinámica o post mortem. *Depuración estática ging* generalmente toma la forma de revisión de código, mientras que *la depuración dinámica* generalmente toma la forma de seguimiento y está estrechamente asociado con las pruebas. *La depuración post mortem* es el acto de depuración el volcado del núcleo (volcado de memoria) de un proceso. Núcleo los volcados a menudo se generan después de que un proceso ha terminado mined debido a una excepción no manejada. Los tres Las técnicas se utilizan en varias etapas del programa. desarrollo.

La actividad principal de la depuración dinámica es seguimiento, que está ejecutando el programa de una pieza a la vez, examinando el contenido de los registros y memoria, para examinar los resultados en cada paso. Hay tres formas de rastrear un programa.

dos categorías de "fallas" en ingeniería de software terminología (ver tema 1.1, Término relacionado con las pruebas) minología, en el Software Testing KA).

Los errores de sintaxis son simplemente cualquier error que ventila el traductor (compilador / intérprete) de analizando correctamente la declaración. Cada estado En un programa, debe poder analizarse antes de el significado puede ser entendido e interpretado (y, por lo tanto, ejecutado). En programación de alto nivel idiomas, los errores de sintaxis se detectan durante el compilación o traducción del alto nivel lenguaje en código máquina. Por ejemplo, en el Lenguaje de programación C / C ++, la declaración "123 = constante;" contiene un error de sintaxis que ser atrapado por el compilador durante la compilación.

Los errores lógicos son *errores* semánticos que resultan en cálculos incorrectos o comportamientos del programa. ¡Tu programa es legal, pero está mal! Entonces los resultados no coincide con la declaración del problema o expectations. Por ejemplo, en la programación C / C ++ lenguaje, la función en línea "int f (int *x*) {return f (*x* -1);} "para calcular factorial x! es legal pero Lógicamente incorrecto. Este tipo de error no puede ser atrapado por un compilador durante la compilación y es a menudo descubierto a través del rastreo de la ejecución de el programa (los verificadores estáticos modernos identifican Algunos de estos errores. Sin embargo, el punto sigue siendo que estos no son comprobables por máquina en general).

Los errores de datos son *errores de* entrada que resultan en datos de entrada que son diferentes de lo que el programa espera o en el procesamiento de datos incorrectos.

- *Un solo paso:* ejecute una instrucción en un tiempo para asegurarse de que cada instrucción sea ejecutable Cuted correctamente. Este método es tedioso pero útil para verificar cada paso de un programa.
- *Puntos* de interrupción *:* dígale al programa que deje de ejecutar ing cuando alcanza una instrucción específica. Esta técnica permite ejecutar rápidamente secuencias de código seleccionadas para obtener un alto nivel Resumen del comportamiento de ejecución.
- *Puntos de observación:* indique al programa que se detenga cuando registro o cambios en la ubicación de la memoria o cuando es igual a un valor específico. Esta tecnica es útil cuando uno no sabe dónde o cuando se cambia un valor y cuando este valor El cambio probablemente causa el error.

5.3. Herramientas de depuración

La depuración puede ser compleja, difícil y tediosa. Al igual que la programación, la depuración también es altamente creativa activo (a veces más creativo que el programa) ming). Por lo tanto, se necesita ayuda de las herramientas. por depuración dinámica, los *depuradores* son ampliamente utilizados y permitir al programador monitorear la ejecución ción de un programa, detener la ejecución, reiniciar el ejecución, establecer puntos de interrupción, cambiar valores en mem- Ory, e incluso, en algunos casos, retroceden en el tiempo.

Para la depuración estática, hay muchos *estáticos herramientas de análisis de código* , que buscan un específico conjunto de problemas conocidos dentro del código fuente.

Existen herramientas comerciales y gratuitas en varios idiomas Estas herramientas pueden ser extremadamente útiles cuando se verifican árboles fuente muy grandes, dónde está poco práctico para hacer tutoriales de código. El UNIX *El* programa de *pelusa* es un ejemplo temprano.

6. Estructura de datos y representación [5 *, s2.1–2.6]

Los programas trabajan en datos. Pero los datos deben ser expresado y organizado dentro de las computadoras antes siendo procesado por programas. Esta organizacion y la expresión de datos para el uso de programas es el sujeto de estructura de datos y representación. Sim- En pocas palabras, una estructura de datos intenta almacenar y organizar datos en una computadora de tal manera que los datos puedan ser utilizado de manera eficiente. Hay muchos tipos de datos estructuras y cada tipo de estructura es adecuada para algunos tipos de aplicaciones Por ejemplo, B / Los árboles B + son muy adecuados para implementar mas- sive sistemas de archivos y bases de datos.

6.1. Descripción general de la estructura de datos

Las estructuras de datos son representaciones informáticas de datos. Las estructuras de datos se utilizan en casi todos los programa. En cierto sentido, ningún programa significativo puede ser construido sin el uso de algún tipo de datos estructura. Algunos métodos de diseño y programas ming languages incluso organizan todo un software sistema alrededor de estructuras de datos. Fundamentalmente, las estructuras de datos son abstracciones definidas en una colección

6.2. Tipos de estructura de datos

Como se mencionó anteriormente, las diferentes perspectivas pueden ser usado para clasificar estructuras de datos. sin embargo, el perspectiva predominante utilizada en la clasificación se centra en el ordenamiento físico y lógico entre elementos de datos Esta clasificación divide la estructura de datos Tures en estructuras lineales y no lineales. Lineal Las estructuras organizan los elementos de datos en una sola dimensión. Sión en la que cada entrada de datos tiene una (física o lógico) predecesor y un sucesor con La excepción de la primera y última entrada. El primero la entrada no tiene predecesor y la última entrada tiene sin sucesor Las estructuras no lineales organizan datos en dos o más dimensiones, en cuyo caso una entrada puede tener múltiples predecesores y sucesores Los ejemplos de estructuras lineales incluyen listas, pilas y colas. Ejemplos de no lineales Las estructuras incluyen montones, tablas hash y árboles. (como árboles binarios, árboles de equilibrio, árboles B y etc.)

Otro tipo de estructura de datos que a menudo es encontrado en la programación es el compuesto estructura. Una estructura de datos compuesta se basa en parte superior de otras estructuras de datos (más primitivas) y, de alguna manera, puede verse como la misma estructura como la estructura subyacente. Ejemplos de comp las estructuras de la libra incluyen conjuntos, gráficos y parti iones Por ejemplo, una partición se puede ver como Un conjunto de conjuntos.

6.3. Operaciones sobre estructuras de datos

Página 232

selección de datos y sus operaciones asociadas.

A menudo, las estructuras de datos están diseñadas para manejar Programa de eficiencia del programa o algoritmo. Ejemplos de tales estructuras de datos incluyen pilas, colas y muchísimo. En otras ocasiones, las estructuras de datos se utilizan para unidad conceptual (tipo de datos abstractos), como el nombre y dirección de una persona. A menudo, una estructura ture puede determinar si un programa se ejecuta en un unos segundos o en unas pocas horas o incluso unos pocos días.

Desde la perspectiva física y logística. ordenamiento cal, una estructura de datos es lineal o no lineal Otras perspectivas dan lugar a diferencias clasificaciones diferentes que incluyen homogénea vs. heterogéneo, estático vs. dinámico, persistente vs. transitoria, externa vs. interna, primitiva vs. agregado, recursivo versus no recursivo; pasivo vs. activo; y estructuras con estado vs. sin estado.

Todas las estructuras de datos admiten algunas operaciones que producir una estructura y ordenamiento específicos, o recuperar datos relevantes de la estructura, almacenar datos utilizar estructura, o eliminar datos de la estructura. Operaciones básicas compatibles con todas las estructuras de datos. incluyen buscar, leer, actualizar y eliminar (CRUD).

- Crear: inserte una nueva entrada de datos en el estructura.
- Leer: recupera una entrada de datos de la estructura.
- Actualizar: modifique una entrada de datos existente.
- Eliminar: elimine una entrada de datos del estructura.

Algunas estructuras de datos también admiten operaciones:

Página 233

- Encuentra un elemento particular en la estructura.
- Ordenar todos los elementos de acuerdo con algún orden.
- Recorrer todos los elementos en un orden específico.
- Reorganizar o reequilibrar la estructura.

Diferentes estructuras soportan diferentes operaciones con diferentes eficiencias. La diferencia La eficiencia entre operaciones puede ser significativa. Por ejemplo, es fácil recuperar el último elemento. insertado en una pila, pero encontrando un elemento particular Ment dentro de una pila es bastante lento y tedioso.

7. Algoritmos y Complejidad

[5 *, s1.1–1.3, s3.3–3.6, s4.1–4.8, s5.1–5.7, s6.1–6.3, s7.1–7.6, s11.1, s12.1]

Los programas no son piezas de código al azar: son escrito meticulosamente para realizar lo esperado por el usuario comportamiento. La guía que se usa para componer programas son algoritmos, que organizan varias funciones en una serie de pasos y tener en cuenta el dominio de la aplicación, la estrategia de solución y Las estructuras de datos que se utilizan. Un algoritmo puede Ser muy simple o muy complejo.

7.1. Descripción general de los algoritmos

En términos abstractos, los algoritmos guían la operafunciones de las computadoras y consisten en una secuencia de acciones compuestas para resolver un problema. Alternativa las definiciones incluyen pero no se limitan a:

- Un algoritmo es cualquier cálculo bien definido. procedimiento nacional que toma algún valor o conjunto de valores como entrada y produce algún valor o conjunto de valores como salida.
- Un algoritmo es una secuencia de cálculo pasos que transforman la entrada en la salida.
- Un algoritmo es una herramienta para resolver un pozo. problema de cálculo especificado

Por supuesto, se prefieren diferentes definiciones por diferentes personas. Aunque no hay univerdefinición aceptada por Sally, existe algún acuerdo que un algoritmo necesita ser correcto, finito (en en otras palabras, terminar eventualmente o uno debe ser

7.2. Atributos de Algoritmos

Los atributos de los algoritmos son muchos y a menudo incluyen modularidad, corrección, mantenibilidad ity, funcionalidad, robustez, facilidad de uso (es decir, fácil de entender por las personas), programa-Tiempo, simplicidad y extensibilidad. Un com El atributo más enfatizado es "rendimiento" o "eficiencia" con lo que nos referimos a ambos tiempos y eficiencia en el uso de recursos mientras que generalmente enfatizando el eje del tiempo. Hasta cierto punto, efila competencia determina si un algoritmo es factible o poco práctico. Por ejemplo, un algoritmo que toma cien años para terminar es prácticamente uso menos e incluso se considera incorrecto.

7.3. Análisis Algorítmico

El análisis de algoritmos es el estudio teórico. del rendimiento y los recursos del programa de computadora uso; hasta cierto punto determina la bondad de un algoritmo Tal análisis usualmente resúmenes lejos los detalles particulares de una computadora específica y se centra en lo asintótico, independiente de la máquina análisis de abolladuras.

Hay tres tipos básicos de análisis. En *En el peor de los casos,* se determina el máximo Mamá tiempo o recursos requeridos por el algoritmo de cualquier entrada de tamaño n . En el *análisis de casos promedio,* uno determina el tiempo o los recursos esperados requerido por el algoritmo sobre todas las entradas de tamaño n ; en la realización de análisis de casos promedio, uno a menudo necesita hacer suposiciones sobre la distribución estadística Aportación de insumos. El tercer tipo de análisis es El *mejor análisis de caso,* en el que se determina el tiempo mínimo o los recursos requeridos por el algoritmo en cualquier entrada de tamaño n . Entre el tres tipos de análisis, el análisis de casos promedio es el más relevante pero también el más difícil de realizar.

Además de los métodos de análisis básicos, hay también el *análisis amortizado,* en el que uno disuade minas el tiempo máximo requerido por un algoritmo ritmo sobre una secuencia de operaciones; y el *análisis competitivo,* en el que se determina El mérito relativo de rendimiento de un algoritmo

capaz de escribirlo en un número finito de pasos), y inequívoco

contra el algoritmo óptimo (que puede no ser conocido) en la misma categoría (para el mismo operaciones).

Página 234

7.4. Estrategias de diseño algorítmico

El diseño de algoritmos generalmente sigue uno de las siguientes estrategias: fuerza bruta, división y conquistar, programación dinámica y codiciosa selección. La *estrategia de fuerza bruta* es en realidad una sin estrategia. Intenta exhaustivamente todo lo posible manera de abordar un problema. Si un problema tiene una solución Además, esta estrategia está garantizada para encontrarla; sin embargo, El gasto de tiempo puede ser demasiado alto. La *división y la estrategia de conquista* mejora la fuerza bruta estrategia dividiendo un gran problema en más pequeño, problemas homogéneos Resuelve el gran problema lem resolviendo recursivamente los problemas más pequeños y peinando las soluciones al problema más pequeño lems para formar la solución al gran problema. los la suposición subyacente para dividir y conquistar es que problemas más pequeños son más fáciles de resolver.

La *estrategia de programación dinámica* mejora sobre la estrategia de divide y vencerás al reconocer ing que algunos de los subproblemas producidos por la división puede ser la misma y por lo tanto evita resolver los mismos problemas una y otra vez Esto elimina ción de subproblemas redundantes puede dramáticamente mejorar la eficiencia.

La *estrategia de selección codiciosa* mejora aún más en programación dinámica al reconocer que no todos los subproblemas contribuyen a la solución ción del gran problema. Al eliminar todo menos un subproblema, la estrategia de selección codiciosa logra la mayor eficiencia entre todos estrategias de diseño de ritmos. A veces el uso de *la aleatorización* puede mejorar la codiciosa selección estrategia al eliminar la complejidad en determinar la elección codiciosa a través del lanzamiento desmedido ping o aleatorización.

7.5. Estrategias de análisis algorítmico

Las estrategias de análisis de algoritmos incluyen *análisis de conteo básico,* en el cual uno realmente cuenta el número de pasos que toma un algoritmo para completar su tarea; *análisis asintótico,* en el cual uno solo considera el orden de magnitud de El número de pasos que toma un algoritmo para completar su tarea; *análisis probabilístico,* en el cual uno hace uso de probabilidades en el análisis de rendimiento promedio de un algoritmo; *amor análisis adaptado,* en el cual uno usa los métodos de agregación, potencial y contabilidad para ana- analizar el peor rendimiento de un algoritmo en un secuencia de operaciones; y *análisis competitivo,* en el cual uno usa métodos tales como potencial y contabilidad para analizar el rendimiento relativo de Un algoritmo para el algoritmo óptimo.

Para problemas y algoritmos complejos, uno puede necesitar usar una combinación de los anteriores estrategias de análisis mencionadas.

8. Concepto básico de un sistema

[6 *, c10]

Ian Sommerville escribe: "un sistema es un propósito colección de componentes interrelacionados que funcionan juntos para lograr algún objetivo "[6 *]. Un sistema El tema puede ser muy simple e incluir solo algunos componentes, como una pluma de tinta, o más bien complejos, como un avión Dependiendo de si los humanos son parte del sistema, los sistemas se pueden dividir en sistemas técnicos basados en computadora y socio- sistemas técnicos Un técnico basado en computadora funciones del sistema sin participación humana, como televisores, teléfonos móviles, termostato, y algo de software; un sistema sociotécnico no funcionará sin la participación humana. Ejemplos de dicho sistema incluyen espacio tripulado vehículos, chips incrustados dentro de un humano, y así adelante.

8.1. Propiedades del sistema emergente

Un sistema es más que simplemente la suma de sus partes. Por lo tanto, las propiedades de un sistema no son simplemente una de las propiedades de sus componentes. En lugar, un sistema a menudo exhibe propiedades que son apropiadas lazos del sistema en su conjunto. Estas propiedades son llamadas *propiedades emergentes* porque se desarrollan solo después de la integración de las partes constituyentes en el sistema. Las propiedades emergentes del sistema pueden ser ya sea funcional o no funcional. Funcional Las propiedades describen las cosas que hace un sistema. Por ejemplo, las propiedades funcionales de una aeronave. incluyen flotación en el aire, transporte de personas o carga, y usar como arma de destrucción masiva. No- propiedades funcionales describen cómo funciona el sistema se comporta en su entorno operativo. Estas puede incluir cualidades tales como consistencia, capacidad ity, peso, seguridad, etc.

Page 235

Figura 13.3. Componentes básicos de un sistema informático basado en el modelo von Neumann

8.2. Ingeniería de Sistemas

"La ingeniería de sistemas es la interdisciplinaria enfoque que rige el total técnico y administrativo esfuerzo inicial requerido para transformar un conjunto de clientes Las necesidades, expectativas y limitaciones de los participantes una solución y para apoyar esa solución a través de fuera de su vida ". [7]. Las etapas del ciclo de vida de los sistemas la ingeniería varía según el sistema que se esté construido pero, en general, incluye requisitos del sistema definición, diseño del sistema, desarrollo del subsistema ment, integración del sistema, prueba del sistema, sistema instalación temática, evolución del sistema y sistema desmantelamiento

Se han producido muchas pautas prácticas en el pasado para ayudar a las personas a realizar la actividad Lazos de cada fase. Por ejemplo, diseño del sistema puede dividirse en tareas más pequeñas de identificación de subsistemas, la asignación del sistema requiere ments to subsystems, especificación del subsistema funcionalidad, definición de interfaces de subsistema, Etcétera.

8.3. Descripción general de un sistema informático

Entre todos los sistemas, uno que obviamente es evasivo para la comunidad de ingeniería de software es El sistema informático. Una computadora es una máquina que ejecuta programas o software. Consiste en una colección intencional de mecánica, electricidad, y componentes electrónicos con cada componente realizando una función preestablecida. En conjunto, estos componentes los ponentes pueden ejecutar las instrucciones que son dados por el programa.

En términos abstractos, una computadora recibe algunos interesa, almacena y manipula algunos datos, y proporciona algo de salida. La característica más distintiva de una computadora es su capacidad de almacenar y ejecutar secuencias de instrucciones llamadas *programas.* Un fenómeno interesante sobre la computadora es la equivalencia universal en funcionalidad. Según Turing, todas las computadoras con un cierto la capacidad mínima es equivalente en su habilidad ity para realizar tareas de computación. En otras palabras, dado suficiente tiempo y memoria, todas las computadoras que van desde una netbook a una supercomputadora, son capaz de calcular exactamente las mismas cosas, independientemente de la velocidad, el tamaño, el costo o cualquier otra cosa.

La mayoría de los sistemas informáticos tienen una estructura que es conocido como el "modelo von Neumann", que consta de cinco componentes: una *memoria* para almacenar instrucciones y datos, una *unidad central de procesamiento* para realizar operaciones aritméticas y lógicas, una *unidad de control* para secuenciar e interpretar instrucciones, *entrada* para obtener información externa ción en la memoria y *salida* para producir Resultados para el usuario. Los componentes básicos de un sistema informático basado en el von Neumann modelo se representan en la figura 13.3.

Página 236

9. Organización de la computadora [8 *, c1 – c4]

Desde la perspectiva de una computadora, una amplia existe una brecha semántica entre su comportamiento previsi ior y el funcionamiento de la electrónica subyacente dispositivos que realmente hacen el trabajo dentro de la comunidad puter Esta brecha se cierra a través de la organización informática nización, que combina varias eléctricas, eléctricas dispositivos tronic y mecánicos en un solo dispositivo eso forma una computadora. Los objetos que la computadora La organización trata con los dispositivos, conexión funciones y controles. La abstracción construida en comp La organización informática es la computadora. ISA, que especifica cosas como el nativo tipos de datos, instrucciones, registros, direccionamiento modos, la arquitectura de memoria, interrupción y manejo de excepciones y las E / S. En general, el ISA especifica la capacidad de una computadora y qué se puede hacer en la computadora con programación.

9.1. Sistemas digitales

En el nivel más bajo, se realizan cálculos por los dispositivos eléctricos y electrónicos dentro de un computadora. La computadora usa circuitos y memoria ory para mantener cargos que representan la presencia o ausencia de voltaje. La presencia de voltaje

9.1. Descripción de la organización informática

Una computadora generalmente consiste en una CPU, memory, dispositivos de entrada y dispositivos de salida. Abstractamente hablando, la organización de una computadora puede ser dividido en cuatro niveles (Figura 13.4). La *macro* nivel de *arquitectura* es la especificación formal de todos las funciones que puede realizar una máquina en particular y se conoce como la arquitectura del conjunto de instrucciones (ES UN). El nivel de *micro arquitectura* es el implemento mentación de la ISA en una CPU específica, en otra palabras, la forma en que las especificaciones de la ISA En realidad se llevan a cabo. El nivel de los *circuitos lógicos* es el nivel donde cada componente funcional de la microarquitectura está formada por circuitos que toman decisiones basadas en reglas simples. los nivel de *dispositivos* es el nivel donde, finalmente, cada lógica el circuito está hecho de dispositivos electrónicos como como semiconductores complementarios de óxido de metal (CMOS), semiconductores de óxido de metal de canal n (NMOS), o transistores de arseniuro de galio (GaAs), Etcétera.

Nivel de arquitectura macro (ISA)
Nivel de micro arquitectura
Nivel de circuitos lógicos
Nivel de dispositivos

Figura 13.4. Niveles de arquitectura de máquina

Cada nivel proporciona una abstracción al nivel arriba y depende del nivel inferior. A un programador, la abstracción más importante es

es igual a 1 mientras que la ausencia de voltaje es un cero. En el disco, la polaridad del voltaje es representativa sembrado por 0s y 1s que a su vez representa los datos almacenado Todo, incluidas las instrucciones y datos se expresan o codifican usando ceros digitales y unos En este sentido, una computadora se convierte en un sistema digital Por ejemplo, el valor decimal 6 puede puede codificarse como 110, la instrucción de adición puede ser codificado como 0001, y así sucesivamente. El componente de la computadora como la unidad de control, ALU, memoria y E / S usan la información para calcular las instrucciones.

9.3. Lógica digital

Obviamente, se necesitan lógicas para manipular los datos. y para controlar el funcionamiento de las computadoras. Esta lógica, que está detrás de la función adecuada de una computadora ción, se llama *lógica digital* porque trata con las operaciones de ceros digitales y unos. Digital La lógica especifica las reglas tanto para construir varios dispositivos digitales de los elementos más simples (como como transistores) y para gobernar el funcionamiento de dispositivos digitales. Por ejemplo, hechizos de lógica digital cuál será el valor si un cero y uno es ANDed, ORed u ORED exclusivamente juntos. Eso también especifica cómo construir decodificadores, multiplexers (MUX), memoria y sumadores que se utilizan para ensamblar la computadora.

9.4. Expresión informática de datos

Como se mencionó anteriormente, una computadora expresa datos con señales eléctricas o ceros digitales y unos. Como solo se usan dos dígitos diferentes en

Página 237

expresión de datos, dicho sistema se llama *binario sistema de expresión* . Debido a la naturaleza inherente de un sistema binario, el valor numérico máximo expresable por un código binario de n bits es $2^n - 1$. Específicamente, el número binario $a_n a_{n-1} \ldots a_1 a_0$ corresponde a $un_n \times 2^n + a_{n-1} \times 2^{n-1} + \ldots + a_1 \times 2^1 + a_0 \times 2^0$. Por lo tanto, el valor numérico del binario la expresión de 1011 es $1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 11$. Para expresar un valor no numérico, nosotros necesita decidir el número de ceros y unos para uso y el orden en que esos ceros y unos están arreglados

Por supuesto, hay diferentes maneras de hacer codificación, y esto da lugar a diferentes datos esquemas de expresión y subesquemas. Por ejemplo, los enteros se pueden expresar en forma de sin signo, el complemento de uno o el complemento de dos. por caracteres, hay ASCII, Unicode e IBM Normas EBCDIC. Para números de coma flotante, existen estándares IEEE-754 FP 1, 2 y 3.

9.5. La Unidad Central de Procesamiento (CPU)

La unidad central de procesamiento es el lugar donde las instrucciones (o programas) se ejecutan realmente. La ejecución generalmente toma varios pasos, incluyendo: ing ir a buscar la instrucción del programa, decodificar la instrucción, buscar operandos, realizar

- Celdas de memoria y chips
- Tarjetas de memoria y módulos
- Jerarquía de memoria y caché
- La memoria como subsistema de la computadora.

Las celdas de memoria y los chips tratan con un solo digital almacenamiento y montaje de unidades de un solo dígito en matrices de memoria unidimensionales también como el montaje de almacenamiento unidimensional matrices en memoria de almacenamiento multidimensional papas fritas. *Las tarjetas de memoria y los módulos se* refieren a montaje de chips de memoria en sistemas de memoria temas, con el foco puesto en la organización, operación y gestión del individuo chips en el sistema. *Jerarquía de memoria y caché* se utilizan para admitir operaciones de memoria eficientes. *La memoria como un subsistema* trata con la interfaz entre el sistema de memoria y otras partes de el ordenador.

9.7 Entrada y salida (E / S)

Una computadora es inútil sin E / S. Común los dispositivos de entrada incluyen el teclado y el mouse; Los dispositivos de salida comunes incluyen el disco, el pantalla, la impresora y los altavoces. E / S diferentes los dispositivos funcionan a diferentes velocidades de datos y reli habilidades. Cómo se conectan y administran las computadoras

operaciones aritméticas y lógicas en la operación
ands, y almacenando el resultado. El componente principal
Las necesidades de una CPU consisten en registros donde las instrucciones
a menudo se leen y escriben en
la unidad aritmética y lógica (ALU) que realiza
la aritmética real (como suma, resta)
ción, multiplicación y división) y lógica (como
como operaciones AND, OR, shift, etc.), el
unidad de control que se encarga de producir
señales apropiadas para controlar las operaciones, y vari-
Autobuses (datos, dirección y control) que enlazan
componentes juntos y transportar datos hacia y
de estos componentes.

9.6. Organización del sistema de memoria

La memoria es la unidad de almacenamiento de una computadora que en
se refiere al ensamblaje de una memoria a gran escala
sistema de almacenamiento más pequeño y de un solo dígito
unidades. Los principales temas cubiertos por el sistema de memoria
La arquitectura tem incluye lo siguiente:

varios dispositivos de entrada y salida para facilitar la
interacción entre computadoras y humanos (o
otras computadoras) es el foco de los temas en E / S.
Los principales problemas que deben resolverse en la entrada
y salida son las formas en que las E / S pueden y deben ser
realizado.

En general, la E / S se realiza tanto en hard-
niveles de software y software. La E / S de hardware puede ser
realizado en cualquiera de las tres formas. *E / S dedicada*
dedica la CPU a la entrada y salida reales
operaciones durante E / S; trata de *E / S con mapeo de memoria*
Operaciones de E / S como operaciones de memoria; e *híbrido*
I / O combina *E / S* dedicada y mapeada en memoria
E / S en un único modo de operación de E / S holístico.

Casualmente, el software de E / S también puede ser per-
formado en una de tres formas. *E / S programada*
deja que la CPU espere mientras el dispositivo de E / S está funcionando
I / O; *La E / S controlada por interrupción* permite el manejo de la CPU
de E / S impulsadas por el dispositivo de E / S; y *directo*
Él mismo acceso a memoria (*DMA)* permite que las E / S sean manejadas por un
CPU secundaria integrada en un dispositivo DMA (o

Página 238

canal). (Excepto durante la configuración inicial, el
la CPU principal no se ve afectada durante una E / S DMA
operación.)

Independientemente de los tipos de esquema de E / S
utilizado, los principales problemas involucrados en E / S incluyen *E / S*
direccionamiento (que trata el tema de cómo
identificar el dispositivo de E / S para una operación de E / S específica
ción), *sincronización* (que trata el problema
de cómo hacer que la CPU y el dispositivo de E / S funcionen
en armonía durante E / S), y *detección de errores y*
corrección (que se ocupa de la aparición de
errores de transmisión).

10. Conceptos básicos del compilador

[4 *, s6.4] [8 *, s8.4]

10.1 Resumen del compilador / intérprete

Los programadores suelen escribir programas en alto
código de idioma de nivel, que la CPU no puede ejecutar
linda; entonces este código fuente tiene que ser convertido a
código de máquina para ser entendido por una computadora.
Debido a las diferencias entre diferentes ISA,
la traducción debe hacerse para cada ISA o spe-
lenguaje de máquina específico bajo consideración.

La traducción generalmente es realizada por una pieza
de software llamado compilador o intérprete .
Este proceso de traducción desde un lenguaje de alto nivel
el lenguaje de máquina se llama compilación
ción o, a veces, interpretación.

10.2 Interpretación y compilación

Hay dos formas de traducir un programa escrito
diez en un lenguaje de nivel superior en código máquina:
interpretación y compilación. *Interpretación*
traduce el código fuente una declaración a la vez
en lenguaje máquina, lo ejecuta en el acto,
y luego vuelve para otra declaración. Ambos
el código fuente de lenguaje de alto nivel y el inter
se requieren preter cada vez que se ejecuta el programa.

La compilación traduce el lenguaje de alto nivel

hay algunas diferencias importantes entre el
dos métodos Primero, un compilador hace la conversión
solo una vez, mientras que un intérprete generalmente consulta
lo verte cada vez que se ejecuta un programa. Segundo,
interpretar el código es más lento que ejecutar el comando
código apilado, porque el intérprete debe analizar
cada declaración en el programa cuando se ejecuta
y luego realice la acción deseada, mientras que el
el código compilado solo realiza la acción dentro de
Un contexto fijo determinado por la compilación.
Tercero, el acceso a las variables también es más lento en un
intérprete porque la asignación de identificadores a
las ubicaciones de almacenamiento deben realizarse repetidamente en la ejecución
tiempo en lugar de en tiempo de compilación.

Las tareas principales de un compilador pueden incluir
preprocesamiento, análisis léxico, análisis, semántico
análisis, generación de código y optimización de código
ción Fallos del programa causados por un compilador incorrecto
El comportamiento puede ser muy difícil de rastrear. por
Por esta razón, los implementadores del compilador invierten mucho
tiempo asegurando la corrección de su software.

10.3 El proceso de compilación

La compilación es una tarea compleja. La mayoría de los compiladores
Divida el proceso de compilación en muchas fases.
Un desglose típico es el siguiente:

- Análisis léxico
- Análisis de sintaxis o análisis
- Análisis semántico
- Codigo de GENERACION

El análisis léxico divide el texto de entrada (el
código fuente), que es una secuencia de caracteres,
en *comentarios* separados , que deben ser ignorados
en acciones posteriores y *símbolos básicos, que*
tienen significados léxicos. Estos simbolos basicos
debe corresponder a algunos símbolos terminales de
la gramática del lenguaje de programación particular
calibrador Aquí los símbolos terminales se refieren a los elementos
símbolos mentales (o tokens) en la gramática que

código fuente en un programa completo de lenguaje máquina no puede ser cambiado
grama (una imagen ejecutable) por un programa llamado
compilador. Después de la compilación, solo el ejecutable
Se necesita una imagen para ejecutar el programa. La mayoría de las aplicaciones
El software de cationes se vende de esta forma.

Mientras tanto la compilación como la interpretación con-
vert código de lenguaje de alto nivel en código de máquina,

- *El análisis de sintaxis* se basa en los resultados de análisis léxico y descubre la estructura en el programa y determina si un texto o no se ajusta a un formato esperado. *¿Es esto un textu-Ally correcto programa de C ++?* o *¿Esta entrada es tex-tually correcto?* son preguntas típicas que pueden ser

Página 239

respondido por análisis de sintaxis. Análisis de sintaxis determina si el código fuente de un programa es correcto rect y lo convierte en un representante más estructurado resentación (árbol de análisis) para análisis semántico o transformación.

El análisis semántico agrega información semántica al árbol de análisis construido durante el análisis de sintaxis y construye la tabla de símbolos. Realiza vari-Nuestras verificaciones semánticas que incluyen verificación de tipos, enlace de objeto (asociando variable y función referencias con sus definiciones) y definidas asignación (que requiere que todas las variables locales sean inicializado antes de su uso). Si se encuentran errores, el las declaraciones semánticamente incorrectas del programa son rechazado y marcado como errores.

Una vez que se completa el análisis semántico, la fase de *generación* de *código* comienza y transforma el código intermedio producido en el anterior fases en el lenguaje de máquina nativo de la computadora bajo consideración. Esto involucra decisiones de recursos y almacenamiento, como decidir qué variables encajan en registros y memoria y la selección y programación de los apropiados instrucciones de la máquina, junto con sus asociados modos de direccionamiento.

A menudo es posible combinar múltiples fases. en una pasada sobre el código en un compilador imple-Mentación. Algunos compiladores también tienen un prepro-fase de cesación al comienzo o después del léxico análisis que hace el trabajo de limpieza necesario, como procesar las instrucciones del programa para El compilador (directivas). Algunos compiladores pro-vide una fase de optimización opcional al final de toda la compilación para optimizar el código (como como el reordenamiento de la secuencia de instrucciones) por eficiencia y otros objetivos deseables solicitado por los usuarios.

11. Conceptos básicos de los sistemas operativos

[4 *, c3]

Todo sistema de complejidad significativa necesita para ser manejado Una computadora, como un complejo sistema electromecánico, necesita su propio hombre-ira por gestionar los recursos y actividades ocurriendo en él. Ese gerente se llama una *operación Sistema de ing* (OS) .

11.1 Resumen de sistemas operativos

Los sistemas operativos son una colección de software y firmware, que controla la ejecución de la computadora programas y presta servicios tales como computadoras asignación de recursos, control de trabajo, con- tacto de entrada / salida trol y gestión de archivos en un sistema informático. Conceptualmente, un sistema operativo es una computadora programa que gestiona los recursos de hardware y hace que sea más fácil de usar por aplicaciones previas enviando buenas abstracciones. Esta bonita abstracción a menudo se llama la máquina virtual e incluye cosas como procesos, memoria virtual y sistemas de archivos Un sistema operativo oculta la complejidad de hardware subyacente y se encuentra en todos los modernos ordenadores.

Los roles principales que desempeñan los sistemas operativos son: ment e ilusión. *La gestión se* refiere a los SO gestión (asignación y recuperación) de fisioterapia recursos cal entre múltiples usuarios competidores / aplicaciones / tareas. *Ilusión se* refiere a lo agradable abstracciones que proporciona el sistema operativo.

11.2 Tareas de un sistema operativo

Las tareas de un sistema operativo difieren significativamente depende de la máquina y el tiempo de su invención. Sin embargo, los sistemas operativos modernos han llegado a un acuerdo sobre las tareas que deben ser realizado por un sistema operativo. Estas tareas incluyen CPU gestión, gestión de memoria, manejo de disco gestión (sistema de archivos), gestión de dispositivos de E / S, y seguridad y protección. Cada tarea del sistema operativo envejece un tipo de recurso físico.

Específicamente, la gestión de la CPU se ocupa de asignación y liberación de la CPU entre empresas programas peting (llamados procesos / hilos en el sistema operativo jerga), incluido el propio sistema operativo. los abstracción principal proporcionada por la gestión de la CPU es El modelo de proceso / hilo. Gestión de la memoria se ocupa de la asignación y liberación de memoria espacio entre procesos competitivos, y los principales abstracción proporcionada por la gestión de memoria es memoria virtual La gestión de discos se ocupa de el intercambio de estado magnético u óptico o sólido discos entre múltiples programas / usuarios y sus principales La abstracción es el sistema de archivos. Dispositivo de E / S gestionado ment trata con la asignación y lanzamientos de varios dispositivos de E / S entre procesos competidores.

Trato de seguridad y protección con la protección de recursos informáticos por uso ilegal.

11.3 Abstracciones del sistema operativo

El arsenal de los sistemas operativos es la abstracción. Correspondiente
Para las cinco tareas físicas, los sistemas operativos usan cinco abstracciones
opciones: proceso / hilo, memoria virtual, sistema de archivos
tems, entrada / salida y dominios de protección. los
La abstracción general del sistema operativo es la máquina virtual.

Para cada área de tareas del sistema operativo, hay tanto un Realidad cal y abstracción conceptual. El fis-
La realidad ical se refiere al recurso de hardware bajo administración; la abstracción conceptual se refiere a la interfaz que el SO presenta a los usuarios / programas por encima. Por ejemplo, en el modelo de hilo del sistema operativo, la realidad física es la CPU y el La abstracción es múltiples CPU. Por lo tanto, un usuario no tiene que preocuparse por compartir la CPU con otros cuando se trabaja en la abstracción proporcionada por un OS. En la abstracción de memoria virtual de un sistema operativo, la realidad física es la RAM física o ROM (lo que sea), la abstracción es múltiple ilimitada espacio de memoria ocupado. Por lo tanto, un usuario no tiene que preocuparse por compartir memoria física con otros o sobre un tamaño de memoria física limitado.

Las abstracciones pueden ser virtuales o transparentes; en este contexto virtual se aplica a algo que parece estar allí, pero no lo está (como la memoria utilizable más allá de lo físico), mientras que se aplica transparente a algo que está ahí, pero parece no ser allí (como recuperar contenido de la memoria del disco o memoria física).

11.4 Clasificación de sistemas operativos

Diferentes sistemas operativos pueden tener diferentes Implementación de funcionalidad. En dias tempranos de la era de la computadora, los sistemas operativos se relacionaban Totalmente simple. A medida que pasa el tiempo, la complejidad y la sofisticación de los sistemas operativos aumenta significativamente. Desde una perspectiva histórica, un sistema operativo se puede clasificar como uno de los siguiendo.

- *Sistema operativo por lotes:* organiza y procesa el trabajo en lotes. Los ejemplos de tales sistemas operativos incluyen IBM FMS, IBSYS y la Universidad de UMES de Michigan.
- *Sistema operativo de procesamiento por lotes multiprogramado:* agrega mulcapacidad de tareas en lotes anteriores simples OS. Un ejemplo de tal sistema operativo es el de IBM OS / 360.
- *Sistema operativo de tiempo compartido:* agrega tareas múltiples e interdispositivos activos en el sistema operativo. Ejemplos de tales sistemas operativos incluyen UNIX, Linux y NT.
- *Sistema operativo en tiempo real:* agrega previsibilidad de tiempo ity en el sistema operativo mediante la programación individual tareas de acuerdo con la finalización de cada tarea plazos de entrega. Los ejemplos de dicho sistema operativo incluyen VxWorks (WindRiver) y DART (EMC).
- *Sistema operativo distribuido:* agrega la capacidad de envejecimiento de una red de computadoras en el sistema operativo.
- *Sistema operativo incorporado:* tiene una funcionalidad limitada y se utiliza para sistemas integrados como automóviles y PDAs. Los ejemplos de tales sistemas operativos incluyen Palm OS, Windows CE y TOPPER.

Alternativamente, un sistema operativo se puede clasificar por su máquina / entorno objetivo aplicable en el siguiendo.

- *Sistema operativo Mainframe: se* ejecuta en la computadora mainframe incluye e incluye OS / 360, OS / 390, AS / 400, MVS y VM.
- *SO del servidor: se* ejecuta en estaciones de trabajo o servidores e incluye sistemas como UNIX, Windows, Linux y VMS.
- *Sistema operativo multicomputador: se* ejecuta en múltiples equipos puters e incluyen ejemplos como Novell Netware
- *SO de computadoras personales: se* ejecuta en personal computadoras e incluyen ejemplos tales como DOS, Windows, Mac OS y Linux.
- *SO del dispositivo móvil: se* ejecuta en dispositivos personales como teléfonos celulares, iPad e incluyen tales ejemplos de iOS, Android, Symbian, etc.

12. Bases de datos y gestión de datos

[4 *, c9]

Una base de datos consiste en una colección organizada de datos para uno o más usos. En cierto sentido, una base de datos es Una generalización y expansión de las estructuras de datos. Pero la diferencia es que una base de datos es usualmente extensos a programas individuales y permanentes en existencia en comparación con las estructuras de datos. Bases de datos se usan cuando el volumen de datos es grande o lógico

Página 241

Las relaciones entre los elementos de datos son importantes. Los Los factores considerados en el diseño de la base de datos incluyen formance, concurrencia, integridad y recuperación de fallas de hardware.

12.1 Entidad y esquema

Las cosas que una base de datos intenta modelar y almacenar son llamadas entidades Las entidades pueden ser objetos del mundo real como personas, automóviles, casas, etc., o pueden ser conceptos abstractos como personas, salario,

12.3 Lenguaje de consulta de base de datos

Los usuarios / aplicaciones interactúan con una base de datos a través de un lenguaje de consulta de base de datos, que es un spelenguaje de programación especializado adaptado a datos uso base. El modelo de base de datos tiende a determinar los idiomas de consulta que están disponibles para acceder a base de datos. Una consulta comúnmente utilizada El estimulador para la base de datos relacional es el estructurado lenguaje de consulta, más comúnmente abreviado como SQL Un lenguaje de consulta común para datos de objetos

nombres, etc. Una entidad puede ser primitiva como un nombre o compuesto como un empleado que consiste en un nombre, número de identificación, salario, dirección, etc.

El concepto más importante en una base de datos. es el *esquema* , que es una descripción de todo estructura de base de datos desde la cual todas las demás bases de datos Se construyen actividades. Un esquema define la relación se envía entre las diversas entidades que componen un base de datos. Por ejemplo, un esquema para una empresa. sistema de nómina consistiría en cosas tales como Identificación del empleado, nombre, salario, dirección, etc. adelante. El software de base de datos mantiene la base de datos De acuerdo con el esquema.

Otro concepto importante en la base de datos es el *modelo de base de datos* que describe el tipo de relación Relación entre varias entidades. El comúnmente los modelos usados incluyen relacional, de red y modelos de objetos

12.2 Sistemas de gestión de bases de datos (DBMS)

Componente del sistema de gestión de bases de datos (DBMS) nents incluyen aplicaciones de bases de datos para el almacenamiento era de los datos estructurados y no estructurados y la funciones de gestión de bases de datos requeridas para ver, recopilar, almacenar y recuperar datos del bases de datos Un DBMS controla la creación, mantenencia y uso de la base de datos y generalmente es categorizado de acuerdo con el modelo de base de datos que apoyos, como los relacionales, de red o modelo de objeto Por ejemplo, una base de datos relacional sistema de gestión (RDBMS) implementa fea- Tures del modelo relacional. Una base de datos de objetos sistema de gestión (ODBMS) implementa features del modelo de objeto.

bases es el lenguaje de consulta de objetos (abreviado como OQL). Hay tres componentes de SQL: datos Lenguaje de definición (DDL), manipulación de datos Idioma (DML) y lenguaje de control de datos (DCL). Un ejemplo de una consulta DML puede verse como el siguiente:

```
SELECT Component_No, Cantidad
DE COMPONENTE
DONDE Artículo_No = 100
```

La consulta anterior selecciona todos los componentes_No y su cantidad correspondiente de una base de datos tabla llamada COMPONENT, donde el Item_No es igual a 100

12.4 Tareas de los paquetes DBMS

Un sistema DBMS proporciona lo siguiente capacidades:

- El *desarrollo de bases de datos* se utiliza para definir y organizar el contenido, las relaciones y la estructura Tura de los datos necesarios para construir una base de datos.
- La *interrogación de la base de datos* se usa para acceder los datos en una base de datos para recuperar información y generación de informes. Los usuarios finales pueden seleccionar Recuperar y mostrar información y Producir informes impresos. Esta es la operación que la mayoría de los usuarios conocen sobre bases de datos.
- El *mantenimiento de la base de datos* se usa para agregar, eliminar, actualizar y corregir los datos en una base de datos.
- El *desarrollo de aplicaciones* se utiliza para desarrollar prototipos de pantallas de entrada de datos, consultas, formularios, informes, tablas y etiquetas para un protocolo aplicación escrita También se refiere al uso de Lengua de cuarta generación o aplicación generadores para desarrollar o generar código de programa.

Página 242

12.5 Gestión de datos

Una base de datos debe administrar los datos almacenados en ella. Esta gestión incluye tanto la organización como almacenamiento.

La organización de los datos reales en una base de datos. depende del modelo de base de datos. En un relacional modelo, los datos se organizan como tablas con diferentes tablas que representan diferentes entidades o relaciones entre un conjunto de entidades. El almacenamiento de ofertas de datos con el almacenamiento de estas tablas de bases de datos en discos. Las formas comunes para lograr esto es usar archivos. Los archivos secuenciales, indexados y hash se utilizan en este propósito con diferentes estructuras de archivos que proporcionan Diferentes prestaciones de acceso y conveniencia.

12.6 Minería de datos

A menudo hay que saber qué buscar antes consultar una base de datos. Este tipo de "localización" el acceso no hace uso completo de la gran cantidad de información almacenada en la base de datos, y de hecho reduce la base de datos en una colección de discretos archivos. Para aprovechar al máximo una base de datos, uno puede realizar análisis estadísticos y patrones de distorsión

proporcionado por redes informáticas. Estos paradigmas incluyen computación distribuida, computación en cuadrícula, Computación en Internet y computación en la nube.

13.1 Tipos de red

Las redes de computadoras no son todas iguales y puede clasificarse de acuerdo con una amplia variedad de características, incluida la conexión de red Topología de acción, tecnologías cableadas, tecnología inalámbrica nologías, escala, topología de red, funciones y velocidad. Pero la clasificación que es familiar para la mayoría se basa en la escala de las redes.

- *La red de área personal / red doméstica* es una red informática utilizada para la comunicación entre computadora (s) e información diferente dispositivos tecnológicos cercanos a una perhijo. Los dispositivos conectados a dicha red el trabajo puede incluir PC, faxes, PDA y Televisores Esta es la base sobre la cual Internet de las cosas se construye.
- *La red de área local* (LAN) se conecta computadoras y dispositivos en un área geográfica limitada área, como un campus escolar, laboratorio de computación

cobertura del contenido de una base de datos utilizando una tecnología
Nique llama *minería de datos.* Tales operaciones pueden ser
solía apoyar una serie de actividades comerciales
que incluyen, entre otros, marketing,
detección de fraude y análisis de tendencias.

Numerosas formas de realizar minería de datos
han sido inventados en la última década e incluyen
técnicas comunes como la descripción de la clase,
discriminación de clase, análisis de agrupamiento, asociación
análisis y análisis atípico.

13. Conceptos básicos de comunicación de red
[8 *, c12]

Una red informática conecta una colección de
computadoras y permite a los usuarios de diferentes computadoras
ers para compartir recursos con otros usuarios. Una red
facilita las comunicaciones entre todos los
computadoras conectadas y pueden dar la ilusión
de una sola computadora omnipresente. Cada com
se llama computadora o dispositivo conectado a una red
Un *nodo de red.*

Han surgido varios paradigmas informáticos
para beneficiarse de las funciones y capacidades
de un edificio de oficinas, o bien ubicado
grupo de edificios

- *Campus Network* es una red informática hecha de una interconexión de redes de área local (LAN) dentro de un área geográfica limitada.
- *La red de área amplia* (WAN) es una computadora red que cubre un área geográfica grande, como una ciudad o país o incluso a través de distancias continentales Una WAN limitada a un la ciudad a veces se llama Área Metropolitana Red.
- *Internet* es la red global que conecta computadoras ubicadas en muchos (quizás todos) países.

Otras clasificaciones pueden dividir las redes en
redes de control, redes de almacenamiento, pri virtual
redes de vate (VPN), redes inalámbricas, punto-
redes punto a punto e Internet de las cosas.

13.2 Componentes de red básicos

Todas las redes están formadas por el mismo hardware básico.
componentes de hardware, incluidas computadoras, red

Página 243

tarjetas de interfaz (NIC), puentes, concentradores, conmutadores,
y enrutadores. Todos estos componentes se llaman *nodos.*
en la jerga de las redes. Cada componente por
forma una función distintiva que es esencial para
el embalaje, conexión, transmisión, amplificación
catión, control, desembalaje e interpretación
de los datos. Por ejemplo, un repetidor amplifica el
señales, un interruptor realiza conexiones de muchos a muchos
iones, un centro realiza conexiones de uno a muchos,
una tarjeta de interfaz está conectada a la computadora y
realiza empaquetamiento y transmisión de datos, un puente
conecta una red con otra, y un enrutador es
una computadora y realiza análisis de datos y
control de flujo para regular los datos de la red.

Las funciones realizadas por varias redes
los componentes corresponden a las funciones especificadas
por uno o más niveles del Open de siete capas
Modelo de red de interconexión de sistemas (OSI),
que se discute a continuación.

13.3 Protocolos y estándares de redes

Las computadoras se comunican entre sí usando
protocolos, que especifican el formato y la regulación
se utilizan para empacar y desempacar datos. Facilitar
comunicación más fácil y mejor estructura, red-
los protocolos de trabajo se dividen en diferentes capas
con cada capa que trata con un aspecto de la
comunicación. Por ejemplo, la disposición física
Los ers se ocupan de la conexión física entre
las partes que se van a comunicar, el enlace de datos
la capa se ocupa de la transmisión de datos sin procesar y
control de flujo, y la capa de red se ocupa de la
Empaque y desempaquetado de datos en un determinado
formato que sea comprensible para el par-
corbatas. La red OSI más utilizada
modelo organiza protocolos de red en siete
capas, como se muestra en la figura 13.5.

Una cosa a tener en cuenta es que no todos los protocolos de
los protocolos de capa de enlace incluyen frame-relay, asyn-
modo de transferencia crónica (ATM) y punto a
Protocolo de puntos (PPP). Protocolos de capa de aplicación
Incluye canal de fibra, sistema de computadora pequeña
Interfaz (SCSI) y Bluetooth. Para cada capa
o incluso cada protocolo individual, puede haber
normas establecidas por nacionales o internacionales
organizaciones para guiar el diseño y desarrollo
ment de los protocolos correspondientes.

Capa de aplicación
Capa de presentación
Capa de sesión
Capa de transporte
Capa de red
Capa de enlace de datos
Capa fisica

Figura 13.5. El modelo de red OSI de siete capas

13.4 La Internet

Internet es un sistema global de interconectado
gubernamental, académica, corporativa, pública y
redes de computadoras privadas. En el dominio público
el acceso a internet es a través de organizaciones
conocidos como proveedores de servicios de internet (ISP). los
ISP mantiene uno o más centros de conmutación
llamado un punto de presencia, que en realidad
conecta a los usuarios a Internet.

13.5 Internet de las Cosas

Internet de las cosas se refiere a la creación de redes
de objetos cotidianos, como automóviles, teléfonos celulares,
IP. Así televisores, refrigeradores e incluso edificios:

cols implementan todas las capas del modelo OSI. por ejemplo, el protocolo TCP / IP no implementa la capa de presentación ni la capa de sesión.

Puede haber más de un protocolo para cada capa. Por ejemplo, UDP y TCP funcionan en la capa de transporte por encima de la capa de red de IP, proviendo el mejor esfuerzo, transporte no confiable (UDP) vs. Función de transporte confiable (TCP). Capa física los protocolos incluyen token ring, Ethernet, Fast Ethernet, gigabit Ethernet y Ethernet inalámbrica. Datos utilizando tecnologías de red cableadas o inalámbricas.

La función y el propósito de *Internet de las cosas* es interconectar todas las cosas para facilitar la autonomous y mejor vida. Tecnologías utilizadas en el Internet de las cosas incluye RFID, inalámbrica y redes cableadas, tecnología de sensores y mucho software por supuesto. Como paradigma de internet de las cosas todavía está tomando forma, mucho trabajo es necesario para que Internet de las cosas gane una amplia difusión aceptación.

Página 244

13.6 Red privada virtual (VPN)

Una red privada virtual es una red virtual previamente planificada conexión entre nodos en una LAN / WAN o en La Internet. Permite al administrador de la red. para separar el tráfico de red en grupos de usuarios que tienen una afinidad común entre sí, como todos los usuarios en la misma organización o grupo de trabajo. Este tipo de circuito puede mejorar el rendimiento. y seguridad entre nodos y permite la facilidad ier mantenimiento de circuitos en la resolución de problemas.

14. Computación paralela y distribuida

[8 *, c9]

La computación paralela es un paradigma informático que surge con el desarrollo de múltiples funciones Unidades nacionales dentro de una computadora. El objetivo principal tive de computación paralela es ejecutar varios tareas simultáneamente en diferentes unidades funcionales y así mejorar el rendimiento o la respuesta o ambos. La informática distribuida, por otro lado, es un paradigma informático que emerge con el desarrollo Opción de redes informáticas. Su objetivo principal es hacer uso de varias computadoras en el red para lograr cosas que de otra manera no serían posibles posible dentro de una sola computadora o mejorar eficiencia de putación aprovechando el poder de Múltiples computadoras.

14.1 Computación Paralela y Distribuida Visión general

Tradicionalmente, la computación paralela investiga formas de maximizar la concurrencia (la simultánea ejecución de múltiples tareas) dentro del límite aria de una computadora. Estudios de computación distribuida sistemas distribuidos, que consisten en múltiples computadoras *autónomas* que se comunican a través de Una red informática. Alternativamente, distribuido la informática también puede referirse al uso de distribuido sistemas para resolver computacionales o transaccionales problemas. En la primera definición, distribuida la informática investiga los protocolos, mecanismos y estrategias que proporcionan la base para computación distribuida; en la última definición, la computación distribuida estudia las formas de dividir un problema en muchas tareas y asignar tales tareas a varias computadoras involucradas en el cómputo.

Fundamentalmente, la informática distribuida es otra forma de computación paralela, aunque en un escala más grande En informática distribuida, la función las unidades nacionales no son ALU, FPU o núcleos separados, pero computadoras individuales. Por esta razón, algunos la gente considera que la informática distribuida es la igual que la computación paralela. Porque ambos distribuir computación en paralelo y paralela implica alguna forma de concurrencia, ambos también se llaman concurrentes alquilar informática.

14.2 Diferencia entre Paralelo y Distribuir-Computación

Aunque la computación paralela y distribuida se parece se mezclan en la superficie, hay un sutil pero distinción real entre ellos: computación paralela ing no necesariamente se refiere a la ejecución de programas en diferentes computadoras; en cambio, se puede ejecutar en diferentes procesadores dentro de un solo computadora. De hecho, consenso entre la informática los profesionales limitan el alcance de la computación paralela En el caso en que una memoria compartida es utilizada por todos los procesadores involucrados en la informática, mientras computación distribuida se refiere a computaciones donde existe memoria privada para cada procesador involucrado en los cálculos.

Otra sutil diferencia entre paralelo y la computación distribuida es esa computación paralela requiere la ejecución concurrente de varias tareas mientras que la informática distribuida no tiene esto necesidad.

Basado en la discusión anterior, es posible clasificar los sistemas concurrentes como "paralelos" o "distribuido" basado en la existencia o no presencia de memoria compartida entre todos los procesos sar: ofertas de computación paralela con computaciones dentro de una sola computadora; Computación distribuída se ocupa de los cálculos dentro de un conjunto de computadoras ers. Según esta opinión, la informática multinúcleo Es una forma de computación paralela.

14.3 Computación Paralela y Distribuida Modelos

Dado que múltiples computadoras / procesadores / núcleos son involucrado en la computación distribuida / paralela, algunos la coordinación entre las partes involucradas es necensayar para asegurar el correcto comportamiento del sistema.

Página 245

Las diferentes formas de coordinación dan lugar a diferencias
modelos de computación ent. El mod más común
A este respecto, los elementos son la memoria compartida (parál-
lel) modelo y el mensaje que pasa (distribuido)
modelo.

En un modelo de *memoria compartida (paralelo)* , todos los
los ordenadores tienen acceso a una memoria central compartida
donde se utilizan cachés locales para acelerar el
poder de procesamiento. Estas cachés usan un protocolo
para asegurar que los datos localizados estén actualizados y
fecha, típicamente el protocolo MESI. El algoritmo
el diseñador elige el programa para su ejecución por
cada computadora El acceso a la memoria central puede
ser síncrono o asíncrono, y debe ser
coordinado de manera que se mantenga la coherencia.
Se han inventado diferentes modelos de acceso para
Tal propósito.

En un modelo de *paso de mensajes (distribuido)* , todos
las computadoras ejecutan algunos programas que colectivamente
lograr algún propósito El sistema debe funcionar.
correctamente independientemente de la estructura de la red
trabajo. Este modelo puede clasificarse adicionalmente en
cliente-servidor (C / S), navegador-servidor (B / S) y
Modelos de n niveles. En el modelo C / S, el servidor pro-
vides servicios y el cliente solicita servicios
del servidor En el modelo B / S, el servidor pro-
vides servicios y el cliente es el navegador. En el
modelo de n niveles, cada nivel (es decir, capa) proporciona servicios
vicios al nivel inmediatamente superior y solicita
servicios del nivel inmediatamente debajo de él. En
De hecho, el modelo de n niveles puede verse como una cadena de
modelos cliente-servidor. A menudo, los niveles entre el
el nivel inferior y el nivel superior se denominan
middleware, que es un tema de estudio distinto
en su propio derecho.

14.4 Problemas principales en computación distribuida

Coordinación entre todos los componentes en un dis-
El entorno informático tributario es a menudo complejo
y consume mucho tiempo. Como el número de núcleos /
CPU / computadoras aumenta, la complejidad de
la computación distribuida también aumenta. Entre
los muchos problemas enfrentados, la coherencia de la memoria y
El consenso entre todas las computadoras son las más difíciles.
los ficticios Muchos paradigmas de computación tienen
ha sido inventado para resolver estos problemas y son
Los principales temas de discusión en distribuido / paralelo
informática.

15. Factores humanos del usuario básico [3 *, c8] [9 *, c5]

El software está desarrollado para satisfacer los deseos humanos o
necesariamente. Por lo tanto, todo el diseño y desarrollo de software
los componentes tienen en cuenta el usuario humano
factores como la forma en que las personas usan el software, cómo
la gente ve el software y lo que los humanos esperan
del software Hay numerosos factores en el
interacción hombre-máquina y documentación ISO 9241
ment series definen todos los estándares detallados de
tales interacciones. [10] Pero el usuario humano básico
los factores considerados aquí incluyen entrada / salida, el
manejo de mensajes de error y la solidez de
El software en general.

15.1 Entrada y salida

Entrada y salida son las interfaces entre usuarios
y software. El software es inútil sin entrada
y salida. Los humanos diseñan software para procesar
alguna entrada y producir salida deseable. Todos
los ingenieros de software deben considerar la entrada y salida
poner como parte integral del producto de software
ellos ingenian o desarrollan. Cuestiones consideradas para
la entrada incluye (pero no se limita a):

- ¿Qué entrada se requiere?
- ¿Cómo se pasa la entrada de los usuarios a
  ¿ordenadores?
- ¿Cuál es la forma más conveniente para que los usuarios
  ingresar entrada?
- ¿Qué formato requiere la computadora?
  los datos de entrada?

El diseñador debe solicitar el mínimo
datos de entrada humana, solo cuando los datos no son
ya almacenado en el sistema. El diseñador debe
formatear y editar los datos en el momento de la entrada a
reducir los errores derivados de incorrectos o maliciosos
entrada de datos.

Para la salida, debemos considerar lo que los usuarios
deseo ver:

- ¿En qué formato les gustaría ver a los usuarios?
  ¿salida?
- ¿Cuál es la forma más agradable de mostrar
  ¿salida?

Página 246

Si la parte que interactúa con el software no es
humano pero otro software o computadora o equipo
sistema de control, entonces debemos considerar la entrada /
tipo de salida y formato que el software debería
producir para garantizar un intercambio de datos adecuado entre
sistemas.

Hay muchas reglas generales para los desarrolladores.

15.3 Robustez del software

La robustez del software se refiere a la capacidad de software
Ware para tolerar entradas erróneas. Se dice software
ser robusto si continúa funcionando incluso cuando
Se dan entradas erróneas. Por lo tanto, es inaceptable
capaz de que el software simplemente se bloquee cuando se encuentra

seguir para producir una buena entrada / salida para un soft-
mercancía. Estas reglas generales incluyen simples y
diálogo natural, lenguaje de usuario hablado, mini-
mizing carga de memoria del usuario, consistencia, mínima
sorpresa, conformidad con los estándares (si
acordado o no: por ejemplo, los automóviles tienen un estándar
interfaz dard para acelerador, freno, dirección).

15.2 Error de mensajes

Es comprensible que la mayoría de los programas de software
detecta fallas y falla de vez en cuando. Pero
los usuarios deben ser notificados si hay algo que
impide la ejecución sin problemas del programa.
Nada es más frustrante que un inesperado
terminación o desviación de comportamiento del software
sin ninguna advertencia o explicación. Ser usuario
amigable, el software debe informar todos los errores
diciones a los usuarios o aplicaciones de nivel superior
para que se pueda tomar alguna medida para rectificar el
situación o salir con gracia. Hay varios
pautas que definen lo que constituye un buen
mensaje de error: los mensajes de error deben ser claros, para
El punto, y oportuno.

Primero, los mensajes de error deben explicar claramente
qué está sucediendo para que los usuarios sepan qué es
pasando en el software. Segundo, el mensaje de error
los sabios deben determinar la causa del error, si
todo lo posible, para que se puedan tomar las acciones adecuadas
Tercero, los mensajes de error deben mostrarse a la derecha
cuando ocurre la condición de error. De acuerdo a
Jakob Nielsen, "Deben ser buenos mensajes de error
expresado en lenguaje sencillo (sin códigos), precisamente
indicar el problema y sugerir constructivamente
una solución "[9 *]. Cuarto, los mensajes de error deberían
no sobrecargar a los usuarios con demasiada información
ción y hacer que ignoren todos los mensajes
juntos.

Sin embargo, los mensajes relacionados con el acceso de seguridad
los errores no deben proporcionar información adicional que
ayudaría a personas no autorizadas a entrar.

declarar un problema de entrada ya que esto puede causar
consecuencias esperadas, como la pérdida de valiosos
datos. El software que exhibe tal comportamiento es con-
echado a un lado para carecer de robustez.

Nielsen da una descripción más simple del software
robustez: "El software debe tener un bajo
tasa de error, para que los usuarios cometan pocos errores durante
el uso del sistema y para que si lo hacen
errores que pueden recuperarse fácilmente de ellos. Promover,
no deben ocurrir errores catastróficos "[9 *].
Hay muchas formas de evaluar la robustez
ness de software y tantas formas de hacer
software más robusto. Por ejemplo, para mejorar
robustez, siempre se debe verificar la validez
de las entradas y valores de retorno antes del progreso
ing más allá; siempre se debe lanzar una excepción
ción cuando ocurre algo inesperado, y
nunca se debe salir de un programa sin primero
dando a los usuarios / aplicaciones la oportunidad de corregir el
condición.

16. Factores humanos del desarrollador básico

[3 *, c31–32]

Los factores humanos del desarrollador se refieren a la consideración
Aciones de factores humanos tomadas al desarrollar
software. El software es desarrollado por humanos, lea
por humanos, y mantenido por humanos. Si alguna-
algo está mal, los humanos son responsables de cor
Rectificando esos errores. Por lo tanto, es esencial escribir
software de una manera fácilmente comprensible
por humanos o, al menos, por otro software
desarrolladores Un programa que es fácil de leer y
entender exhibiciones legibilidad.

Los medios para asegurar que el software cumpla con esto
Los objetivos son numerosos y van desde
arquitectura a nivel macro a lo particular
estilo de codificación y uso variable a nivel micro.
Pero dos factores destacados son la *estructura* (o
diseños del programa) y *comentarios* (documentación).

16.1 Estructura

Los programas bien estructurados son más fáciles de entender.
y modificar Si un programa está mal estructurado, entonces
ninguna cantidad de explicaciones o comentarios es suficiente
para que sea comprensible. Las formas de organizar un
programa son numerosos y van desde el adecuado
uso de espacios en blanco, sangría y paréntesis para
arreglos agradables de agrupaciones, líneas en blanco y
tirantes. Cualquiera sea el estilo que uno elija, debería ser
consistente en todo el programa.

16.2 Comentarios

Para la mayoría de las personas, la programación es codificación y
la gente no se da cuenta de que la programación también
incluye escribir comentarios y que los comentarios son
Una parte integral de la programación. Es cierto, comentarios
no son utilizados por la computadora y ciertamente no
constituyen instrucciones finales para la computadora, pero
mejoran la legibilidad de los programas por

- Dentro de una función, los comentarios deben ser
  dado para cada sección lógica de codificación para
  explicar el significado y el propósito (intención)
  de la sección.
- Los comentarios deben estipular qué libertad
  hace (o no) el programa de mantenimiento
  mers tienen respecto a hacer cambios a
  ese código
- Rara vez se requieren comentarios para indi-
  declaraciones visuales. Si una declaración necesita comp
  De hecho, uno debería reconsiderar la declaración.

17. Desarrollo seguro de software y Mantenimiento

[11 *, c29]

Debido al aumento de actividades maliciosas dirigidas
En los sistemas informáticos, la seguridad se ha convertido en una señal
problema importante en el desarrollo de software. En
Además de la corrección y fiabilidad habituales,

explicando el significado y la lógica de las declaraciones o secciones de código. Debe recordarse que los programas no solo están destinados a computadoras, sino también también son leídos, escritos y modificados por humanos.

Los tipos de comentarios incluyen la repetición de código, explicación del código, marcador del código, resumen del código, descripción del intención del código e información que no puede ser posiblebly se expresará por el propio código. Algunos com los comentarios son buenos, algunos no. Los buenos son los que explican la intención del código y justifique por qué este código se ve como se ve. los los malos son una repetición del código y dicen irrel-Información evasiva. Los mejores comentarios son auto-código de documentación Si el código está escrito en tal de manera clara y precisa que su significado es auto-proclamado, entonces no se necesita ningún comentario. Pero esto Es más fácil decirlo que hacerlo. La mayoría de los programas se explica por sí mismo y a menudo son difíciles de leer y entienda si no se hacen comentarios.

Aquí hay algunas pautas generales para escribir buenos comentarios:

- Los comentarios deben ser consistentes en todo el programa completo
- Cada función debe estar asociada con comentarios que explican el propósito de la función y su papel en el programa general.

los desarrolladores de software también deben prestar atención a La seguridad del software que desarrollan. Seguro el desarrollo de software crea seguridad en el software siguiendo un conjunto de recomendaciones establecidas y / o reglas y prácticas reparadas en el desarrollo de software ment. Complementos seguros de mantenimiento de software desarrollo de software seguro garantizando el no ese introducen problemas de seguridad durante el software mantenimiento.

Una visión generalmente aceptada sobre el software la seguridad es que es mucho mejor diseñar la seguridad en el software que parcharlo después de que el software es desarrollado. Para diseñar la seguridad en software, uno debe tener en cuenta todas las etapas del soft-ciclo de vida de desarrollo de artículos. En particular, seguro el desarrollo de software implica *software requerido seguridad* , *diseño de* software *seguridad* , *software seguridad en la construcción* y *seguridad de pruebas de software Rity* . Además, la seguridad también debe tenerse en consideración al realizar mantenimiento de software nance como fallas de seguridad y lagunas pueden ser y a menudo se introducen durante el mantenimiento.

17.1 Seguridad de requisitos de software

Los requisitos de software de seguridad se ocupan de aclaración y especificación de la política de seguridad y objetivos en los requisitos de software, que

Página 248

sienta las bases para consideraciones de seguridad en El desarrollo de software. Factores a considerar en esta fase incluyen requisitos de software y amenazas / riesgos. El primero se refiere a lo específico funciones que son necesarias en aras de la seguridad rity este último se refiere a las posibles formas en que el La seguridad del software está amenazada.

17.2 Seguridad de diseño de software

El diseño de software se ocupa de la seguridad del diseño de módulos de software que se unen para cumplir los objetivos de seguridad especificados en la seguridad requisitos Este paso aclara los detalles de consideraciones de seguridad y desarrolla el específico pasos para la implementación. Factores considerados puede incluir marcos y modos de acceso que configurar el monitoreo de seguridad general / hacer cumplir estrategias de ment, así como la política individual mecanismos de ejecución.

17.3 Seguridad de construcción de software

La seguridad de la construcción de software se refiere a las preguntas ción de cómo escribir código de programación real para situaciones específicas tales que consideraciones de seguridad son atendidos El término "construcción de software Seguridad "podría significar cosas diferentes para diferentes gente. Puede significar la forma en que una función específica es codificado, de modo que la codificación en sí sea segura, o puede significa la codificación de seguridad en software.

La mayoría de las personas enredan a los dos juntos sin distinción. Una razón para tal enredo es que no está claro cómo uno puede asegurarse de que un La codificación específica es segura. Por ejemplo, en C pro-

- Estructurar el proceso para que todas las secciones que requieren privilegios adicionales son módulos. los los módulos deben ser lo más pequeños posible y debe realizar solo aquellas tareas que requieren esos privilegios
- Asegúrese de que cualquier suposición en el programa son validados Si esto no es posible, docu-Mencionarlos para los instaladores y mantenedores para que sepan los supuestos que los atacantes intentará invalidar.
- Asegúrese de que el programa no comparta objetos en memoria con cualquier otro programa.
- El estado de error de cada función debe ser comprobado. No intente recuperarse a menos que ninguno la causa del error ni sus efectos afectan Cualquier consideración de seguridad. El programa debería restaurar el estado del software a el estado que tenía antes de que comenzara el proceso, y luego terminar.

17.4 Seguridad de pruebas de software

La seguridad de las pruebas de software determina que el software Vigorpro protege datos y mantiene especificaciones de seguridad ficación según lo dado. Para más información, por favor consulte el Software Testing KA.

17.5 Incorporar seguridad a la ingeniería de software Proceso

El software es tan seguro como su desarrollo. proceso va Para garantizar la seguridad del software, la seguridad debe estar integrada en el ingeniero de software proceso de ing. Una tendencia que emerge a este respecto es el contrato de Secure Development Lifecycle (SDL)

lenguaje gramatical, la expresión de i << 1 (shift la representación binaria del valor de i a la izquierda por un bit) y 2 * i (multiplica el valor de la variable i por constante 2) significan lo mismo semánticamente, ¿Pero tienen la misma ramificación de seguridad? La respuesta podría ser diferente para diferentes empresas combinaciones de ISA y compiladores. Debido a esta falta de comprensión, la construcción de software asegura rity, en su estado actual de existencia, principalmente se refiere al segundo aspecto mencionado anteriormente: el codificación de seguridad en software.

La codificación de la seguridad en el software puede ser logrado siguiendo las reglas recomendadas. Unos pocos tales reglas siguen:

cept, que es un modelo espiral clásico que toma Una visión holística de la seguridad desde la perspectiva del ciclo de vida del software y garantiza que la seguridad sea inherente al diseño y desarrollo de software, no una idea de último momento en la producción. El programa SDL se afirma que cess reduce el mantenimiento del software costos y aumentar la confiabilidad del software ing fallas relacionadas con la seguridad del software.

17.6. Pautas de seguridad del software

Aunque no hay formas a prueba de balas para la seguridad desarrollo de software, algunas pautas generales existen que pueden utilizarse para ayudar a tal esfuerzo. Estas

Página 249

las pautas abarcan cada fase del software ciclo de vida de desarrollo. Alguna guía de buena reputación las líneas son publicadas por Computer Emergency El Equipo de respuesta (CERT) y abajo son sus principales 10 prácticas de seguridad de software (los detalles pueden ser encontrado en [12]:

1. Validar entrada.
2. Preste atención a las advertencias del compilador.
3. Arquitecto y diseño de políticas de seguridad.
4. Mantenlo simple.
5. Denegación predeterminada.
6. Adherirse al principio del menor privilegio.
7. Desinfecte los datos enviados a otro software.
8. Practica la defensa en profundidad.
9. Use técnicas efectivas de aseguramiento de la calidad.
10. Adoptar una seguridad de construcción de software estándar.

Page 250

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	03 d 20 [2 *] Olan V	04 ell 20 norte [3 *] en cC METRO	08 oreja 20 sh [4 *] Arroyo	07 itz y col. 20 [5 *] orow H	ile 2011 erv [6 *] metro gm S	06 r 20 tu transmisión exterior d L [8 *] ll an tu norte	[9 *] ielsen 1993 norte	02 op 20 sh [11 *] segundo
1. Resolución de problemas Tecnicas	s3.2, s4.2							
1.1. Definicion de Resolución de problemas	s3.2							
1.2. Formulando el Problema real	s3.2							
1.3. Analiza el Problema	s3.2							
1.4. Diseñar un Búsqueda de soluciones Estrategia	s4.2							
1.5. Resolución de problemas Usando programas		c5						
2. abstracción		s5.2–5.4						
2.1. Niveles de Abstracción		s5.2–5.3						
2.2. Encapsulamiento		s5.3						
2.3. Jerarquía		s5.2						
3. Programación Fundamentos		c6-19						
3.1. los Programación Proceso		c6 – c19						
3.2. Programación Paradigmas		c6 – c19						
3.3. Defensivo Programación		c8						
4. Programación Conceptos básicos de lenguaje			c6					
4.1. Programación Descripción general del idioma			s6.1					
4.2. Sintaxis y Semántica de Programación Idioma			s6.2					

Page 251

	03	04	08	07	06	02
					r 20 tu	

	d 20 [2 *] Olan V	ell 20 norte en cC METRO [3 *]	oreja 20 sh [4 *] Arroyo	itz y col. 20 [5 *] orow H	ile 2011 erv metro [6 *] om S	transmisión exterior d L [8 *] ll an tu norte	[9 *] ielsen 1993 norte	op 20 sh segundo [11 *]
4.3. Nivel bajo Programación Idioma			s6.5–6.7					
4.4. Nivel alto Programación Idioma			s6.5–6.7					
4.5. Declarativo vs. Imperativo Programación Idioma			s6.5–6.7					
5. Herramientas de depuración y técnicas	c23							
5.1. Tipos de errores	s23.1							
5.2. Depuración Técnicas:	s23.2							
5.3. Depuración Herramientas	s23.5							
6. Estructura de datos y Representación			s2.1–2.6					
6.1. Estructura de datos Visión general			s2.1–2.6					
6.2. Tipos de datos Estructura			s2.1–2.6					
6.3. Operaciones en Estructuras de datos			s2.1–2.6					
7. Algoritmos y Complejidad			s1.1–1.3, s3.3–3.6, s4.1–4.8, s5.1–5.7, s6.1–6.3, s7.1–7.6, s11.1, s12.1					

Page 252

	03 d 20 [2 *] Olan V	04 ell 20 norte en [3 *] cC METRO	08 oreja 20 sh [4 *] Arroyo	07 itz y col. 20 [5 *] orow H	ile 2011 erv metro [6 *] om S	06 r 20 tu transmisión exterior d L [8 *] ll an tu norte	[9 *] ielsen 1993 norte	02 op 20 sh segundo [11 *]
7.1. Visión general de Algoritmos			s1.1–1.2					
7.2. Atributos de Algoritmos			s1.3					
7.3. Algorítmico Análisis			s1.3					
			s3.3–3.6,					

7.4. Algorítmico Estrategias de diseño	s4.1– 4.8, s5.1– 5.7, s6.1– 6.3, s7.1– 7.6, s11.1, s12.1	
7.5. Algorítmico Estrategias de análisis	s3.3– 3.6, s4.1– 4.8, s5.1– 5.7, s6.1– 6.3, s7.1– 7.6, s11.1, s12.1	
8. Concepto básico de un Sistema		c10
8.1. Emergente Propiedades del sistema		s10.1
8.2. Sistema Ingenieria		s10.2
8.3. Resumen de un Sistema informático		

Page 253

	03 d 20 Olan V [2 *]	**04 ell 20 norte en cC METRO [3 *]**	**08 oreja 20 sh Arroyo [4 *]**	**07 itz y col. 20 orow H [5 *]**	**ile 2011 erv metro om S [6 *]**	**06 r 20 tu transmisión exterior d L ll an tu norte [8 *]**	**ielsen 1993 norte [9 *]**	**02 op 20 ish segundo [11 *]**
9. Computadora Organización						c1–4		
9.1. Computadora Organización Visión general						s1.1–1.2		
9.2. Sistemas digitales						c3		
9.3. Lógica digital						c3		
9.4. Computadora Expresión de datos						c2		
9.5. El central Unidad de procesamiento (UPC)						s4.1– 4.2 4.2		
9.6. Sistema de memoria Organización						s4.6		
9.7 Entrada y salida (E / S)						s4.5		
10. Conceptos básicos del compilador			s6.4			s8.4		
10.1 Compilador Visión general						s8.4		
10.2 Interpretación						s8.4		

y compilación 10.3 los Proceso de compilación	s6.4	s8.4
11. Operativo Conceptos básicos de sistemas	c3	
11.1 Operando Resumen de sistemas	s3.2	
11.2 Tareas de Sistema operativo	s3.3	
11.3 Operando Sistema de abstracciones	s3.2	
11.4 Operando Sistemas Clasificación	s3.1	

Page 254

03 d 20 [2 *] Olan V	**04 ell 20 norte [3 *] en cC METRO**	**08 oreja 20 sh [4 *] Arroyo**	**07 [5 *] itz y col. 20 orow H**	**ile 2011 [6 *] rv metro om S**	**06 r 20 tu transmisión exterior [8 *] d L ll an tu norte**	**[9 *] ielsen 1993 norte**	**02 op 20 [11 *] sh segundo**	
12. Base de datos Conceptos básicos y datos administración		c9						
12.1 Entidad y Esquema		s9.1						
12.2 Base de datos administración Sistemas (DBMS)		s9.1						
12.3 Base de datos Lenguaje de consulta		s9.2						
12.4 Tareas de Paquetes DBMS		s9.2						
12.5 Datos administración		s9.5						
12.6 Minería de datos		s9.6						
13. red Comunicación Lo esencial					c12			
13.1 Tipos de Red					s12.2– 12,3			
13.2 Red básica Componentes					s12.6			
13.3 Redes Protocolos y Normas					s12.4– 12,5			
13.4 La Internet								
13.5 Internet de Cosas					s12.8			
13.6 Privado virtual Red								
14. Paralelo y Repartido Informática					c9			

14.1 Paralela y distribuido Informática Visión general							s9.4.1–9.4.3		

255 de 1189.

	03 d 20 Olan V [2 *]	04 ell 20 norte en cC METRO [3 *]	08 oreja 20 sh [4 *] Arroyo	07 itz y col. 20 orow H [5 *]	ile 2011 orv metro om S [6 *]	06 r 20 tu transmisión exterior d L ll an tu norte [8 *]	[9 *] ielsen 1993 norte	02 op 20 ish segundo [11 *]
14.2 Las diferencias entre paralelo y distribuido Informática						s9.4.4–9.4.5		
14.3 Paralela y distribuido Modelos de computación						s9.4.4–9.4.5		
14.4 Temas principales en distribuido Informática								
15. Usuario básico Factores humanos		c8					c5	
15.1 Entrada y Salida							s5.1, s5.3	
15.2 Error de mensajes							s5.2, s5.8	
15.3 Software Robustez							s5.5–5.6	
16. Desarrollador básico Factores humanos		c31–32						
16.1 Estructura		c31						
16.2 Comentarios		c32						
17. Software seguro Desarrollo y Mantenimiento								c29
17.1 Dos aspectos de Codificación segura								s29.1
17.2 Codificación Seguridad en Software								s29.4
17.3 Requisito Seguridad								s29.2
17.4 Diseño Seguridad								s29.3
17.5 Implementación Seguridad								s29.5

Page 256

Referencias

[1] Grupo de trabajo conjunto sobre planes de estudio de informática de la IEEE Computer Society and Association para maquinaria informática, *software Ingeniería 2004: Directrices curriculares para programas de pregrado en Ingeniería de Software* , 2004;http: // sitios. computer.org/ccse/SE2004Volume.pdf .

[2 *] G. Voland, *Ingeniería por diseño* , 2ª ed., Prentice Hall, 2003.

[3 *] S. McConnell, *Código completo* , 2ª ed., Microsoft Press, 2004.

[4 *] JG Brookshear, *Ciencias de la computación: un Descripción general* , décima edición, Addison-Wesley, 2009.

[5 *] E. Horowitz et al., *Algoritmos informáticos* , 2a ed., Silicon Press, 2007.

[6 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[7] *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010.

[8 *] L. Null y J. Lobur, *Los fundamentos de Organización y arquitectura de computadoras* , 2ª ed., Jones and Bartlett Publishers, 2006

[9 *] J. Nielsen, *Ingeniería de usabilidad* , Morgan Kaufmann, 1993.

[10] ISO 9241-420: 2011 Ergonomía del ser humano. Interacción del sistema, ISO, 2011.

[11] M. Bishop, *Seguridad informática: arte y Science* , Addison-Wesley, 2002.

[12] RC Seacord, la *codificación segura CERT C Estándar* , Addison-Wesley Professional, 2008

CAPITULO 14

FUNDAMENTOS MATEMÁTICOS

INTRODUCCIÓN

Los profesionales del software viven con programas. en un lenguaje muy simple, solo se puede programar para en resumen, puede escribir un programa solo para un problema Si sigue alguna lógica. El objetivo de este KA es ayudarlo a desarrollar la habilidad para identificar y Describe esa lógica. El énfasis está en ayudar

algo que sigue a un bien entendido, no lógica ambigua Los fundamentos matemáticos área de conocimiento (KA) ayuda a ingenieros de software comprender esta lógica, que a su vez se traduce en código de lenguaje de programación. Las matemáticas ics que es el enfoque principal en este KA es bastante diferente de la aritmética típica, donde los números son tratados y discutidos. Lógica y razón ing son la esencia de las matemáticas que un software El ingeniero debe abordar.

La matemática, en cierto sentido, es el estudio de la educación formal. [1 *, c2] sistemas. La palabra "formal" está asociada con precisión, por lo que no puede haber ninguna ambigüedad o interpretación errónea del hecho. Matemáticas ics es por lo tanto el estudio de todos y cada uno verdades sobre cualquier concepto. Este concepto puede ser sobre números, así como sobre símbolos, imágenes, sonidos, video, casi cualquier cosa. En resumen, no solo los números y las ecuaciones numéricas son sub-Objeto a la precisión. Por el contrario, un software el ingeniero necesita tener una abstracción precisa en un dominio de aplicación diversa.

La base matemática de la *guía SWEBOK* iones KA cubre técnicas básicas para identificar un conjunto de reglas para razonar en el contexto del sistema en estudio Cualquier cosa que se pueda deducir bajar estas reglas es una certeza absoluta dentro de El contexto de ese sistema. En este KA, las técnicas que puede representar y llevar adelante el razonamiento y juicio de un ingeniero de software en un preciso (y, por lo tanto, matemática) se definen las formas y discutido El lenguaje y los métodos de la lógica. que se discuten aquí nos permiten describir las matemáticas pruebas ematicas para inferir de manera concluyente lo absoluto verdad de ciertos conceptos más allá de los números. En entiendes los conceptos básicos en lugar de en desafiando tus habilidades aritméticas.

DESGLOSE DE TEMAS PARA FUNDAMENTOS MATEMÁTICOS

El desglose de temas para la matemática Fundamentos KA se muestra en la Figura 14.1.

1. Conjunto, relaciones, funciones

Conjunto. Un conjunto es una colección de objetos, llamados elementos. del conjunto. Un conjunto puede representarse enumerando su elementos entre llaves, por ejemplo, S = {1, 2, 3}.

El símbolo ∈ se usa para expresar que un element pertenece a un conjunto o, en otras palabras, es un miembro del conjunto. Su negación está representada por ∉, por ejemplo, 1 ∈ S, pero 4 ∉ S.

En una representación más compacta del conjunto usando establecer notación de constructor, {x | P (x)} es el conjunto de todas las x tal que P (x) para cualquier proposición P (x) sobre cualquier universo del discurso Ejemplos de alguna importancia Los conjuntos de tant incluyen lo siguiente:

N = {0, 1, 2, 3, ...} = el conjunto de elementos no negativos enteros

Z = {…, −3, −2, −1, 0, 1, 2, 3,…} = el conjunto de enteros

Conjunto finito e infinito. Un conjunto con un número finito El conjunto de elementos se denomina conjunto finito. A la inversa, cualquier conjunto que no tenga un número finito de elementos ments in it is a *infinite set* . El conjunto de todo natural números, por ejemplo, es un conjunto infinito.

Figura 14.1. Desglose de temas para los fundamentos matemáticos KA

Cardinalidad. La cardinalidad de un conjunto finito S es El número de elementos en S. Esto está representado | S |, por ejemplo, si S = {1, 2, 3}, entonces | S | = 3.

Conjunto universal. En general S = {x ∈ U | p (x)}, donde U es el universo del discurso en el que el predicado P (x) debe ser interpretado. La UNIverso del discurso "para un predicado dado es a menudo referido como el conjunto universal. Alternativamente, uno puede definir conjunto universal como el conjunto de todos los elementos.

Establecer igualdad. Dos conjuntos son iguales si y solo si tienen los mismos elementos, es decir:

$$X = Y \equiv \forall p\,(p \in X \leftrightarrow p \in Y).$$

Conjunto vacio. Un conjunto sin elementos se llama *conjunto vacío* . Un conjunto vacío, denotado por ∅, también es referido como un conjunto nulo o nulo.

Set de poder. El conjunto de todos los subconjuntos de un conjunto X es llamado el *conjunto* de *potencia* de X. Se representa como $\wp$ (X).

Por ejemplo, si X = {a, b, c}, entonces $\wp$ (X) = {∅, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}}. Si X = n, entonces | $\wp$ (X) | = 2^n.

Diagramas de Venn. Los diagramas de Venn son representaciones gráficas representaciones de conjuntos como áreas cerradas en el plano.

Por ejemplo, en la Figura 14.2, el rectángulo representa resiente el conjunto universal y la región sombreada representa un conjunto X.

Subconjunto. X es un subconjunto del conjunto Y, o X está contenido en Y, si todos los elementos de X están incluidos en Y. Esto es denotado por X ⊆ Y. En otras palabras, X ⊆ Y si y solo si ∀p (p ∈ X → p ∈ Y).

Por ejemplo, si X = {1, 2, 3} e Y = {1, 2, 3, 4, 5}, luego X ⊆ Y.

Si X no es un subconjunto de Y, se denota como X Y.

Subconjunto propio. X es un subconjunto apropiado de Y (denotado por X ⊂ Y) si X es un subconjunto de Y pero no es igual a Y, es decir, hay algún elemento en Y que no está en X.

En otras palabras, X ⊂ Y si (X ⊆ Y) ∧ (X ≠ Y).

Por ejemplo, si X = {1, 2, 3}, Y = {1, 2, 3, 4}, y Z = {1, 2, 3}, entonces X ⊂ Y, pero X no es un subconjunto apropiado de Z. Los conjuntos X y Z son conjuntos iguales.

Si X no es un subconjunto adecuado de Y, se denota como X ⊄ Y.

Superconjunto Si X es un subconjunto de Y, entonces se llama Y un *superconjunto* de X. Esto se denota por Y ⊇ X, es decir, Y ⊇ X si y solo si X ⊆ Y.

Por ejemplo, si X = {1, 2, 3} e Y = {1, 2, 3, 4, 5}, luego Y ⊇ X.

Figura 14.2. Diagrama de Venn para el set X

1.1 Establecer operaciones

Intersección. La intersección de dos conjuntos X y Y, denotado por X ∩ Y, es el conjunto de elementos comunes Mentiones tanto en X como en Y.

En otras palabras, X ∩ Y = {p | (p ∈ X) ∧ (p ∈ Y)}.

Como, por ejemplo, {1, 2, 3} ∩ {3, 4, 6} = {3}

Si X ∩ Y = f, entonces se dicen los dos conjuntos X e Y ser un par de conjuntos disjuntos.

Page 259

Un diagrama de Venn para establecer la intersección se muestra en Figura 14.3. La parte común de los dos conjuntos. representa la intersección establecida.

Figura 14.3. Intersección de conjuntos X e Y

Unión. La unión de dos conjuntos X e Y, denotada por X ∪ Y, es el conjunto de todos los elementos en X o en Y, o en ambos.

En otras palabras, X ∪ Y = {p | (p ∈ X) ∨ (p ∈ Y)}.

Como, por ejemplo, {1, 2, 3} ∪ {3, 4, 6} = {1, 2, 3, 4, 6}.

Figura 14.4. Unión de conjuntos X e Y

Cabe señalar que | X ∪ Y | = | X | + | Y | - | X ∩ Y |.

Un diagrama de Venn que ilustra la unión de dos los conjuntos están representados por la región sombreada en la figura 14.4

Complemento. El conjunto de elementos en el universo el conjunto sal que no pertenece a un conjunto dado X se llama su complemento conjunto X '.

En otras palabras, X '= {p | (p ∈ U) ∧ (p ∉ X)}.

La parte sombreada del diagrama de Venn en la figura ure 14.5 representa el conjunto de complemento de X.

Establecer diferencia o complemento relativo. El conjunto de elementos que pertenecen al conjunto X pero no al conjunto Y construye la diferencia establecida de Y de X. Esto es representado por X - Y.

En otras palabras, X - Y = {p | (p ∈ X) ∧ (p ∉ Y)}.

Como, por ejemplo, {1, 2, 3} - {3, 4, 6} = {1, 2}.

Se puede demostrar que X - Y = X ∩ Y '.

Establecer la diferencia X - Y se ilustra con el sombreado región en la Figura 14.6 usando un diagrama de Venn.

Figura 14.6. Diagrama de Venn para X - Y

Producto cartesiano. Un par ordinario {p, q} es Un conjunto con dos elementos. En un conjunto, el orden de la elementos es irrelevante, entonces {p, q} = {q, p}.

En un par ordenado (p, q), el orden de ocurrencia Rences de los elementos es relevante. Por lo tanto, (p, q) ≠ (q, p) a menos que p = q. En general (p, q) = (s, t) si y solo si p = s y q = t.

Dados dos conjuntos X e Y, su producto cartesiano X × Y es el conjunto de todos los pares ordenados (p, q) de modo que p ∈ X y q ∈ Y.

En otras palabras, X × Y = {(p, q) | (p ∈ X) ∧ (q ∈ Y)}.

Como por ejemplo, {a, b} × {1, 2} = {(a, 1), (a, 2), (b, 1), (b, 2)}

1.2. Propiedades del conjunto

Algunas de las propiedades y leyes importantes de los conjuntos. se mencionan a continuación.

1. Leyes asociativas:
   X ∪ (Y ∪ Z) = (X ∪ Y) ∪ Z
   X ∩ (Y ∩ Z) = (X ∩ Y) ∩ Z

Figura 14.5. Diagrama de Venn para el conjunto de complementos de X

2. Leyes conmutativas:
   X ∪ Y = Y ∪ X X ∩ Y = Y ∩ X

3. Leyes distributivas:
   X ∪ (Y ∩ Z) = (X ∪ Y) ∩ (X ∪ Z)
   X ∩ (Y ∪ Z) = (X ∩ Y) ∪ (X ∩ Z)

4. Leyes de identidad:
   X ∪ ∅ = X X ∩ U = X

5. Leyes complementarias:
   X ∪ X '= U X ∩ X '= ∅

6. Leyes idempotentes:
   X ∪ X = X X ∩ X = X

7. Leyes vinculadas:
   X ∪ U = U X ∩ ∅ = ∅

8. Leyes de absorción:
   X ∪ (X ∩ Y) = X X ∩ (X ∪ Y) = X

9. Leyes de De Morgan:
   (X ∪ Y) '= X' ∩ Y ' (X ∩ Y) '= X' ∪ Y '

1.3. Relación y función

Una relación es una asociación entre dos conjuntos de información. Por ejemplo, consideremos un conjunto de residentes de una ciudad y sus números de teléfono. El emparejamiento de nombres con el teléfono correspondiente Los números son una relación. Este emparejamiento está *ordenado* para toda la relación En el ejemplo que se considera ered, para cada par, o el nombre viene primero seguido por el número de teléfono o al revés. El conjunto del que se extrae el primer elemento es llamado el *conjunto de dominios* y el otro conjunto se llama El *rango establecido* . El dominio es con lo que comienzas y el rango es con lo que terminas.

Una función es una relación de *buen comportamiento* . Una relación ción R (X, Y) se comporta bien si la función se asigna cada elemento del dominio establece X en un solo elemento men del conjunto de rango Y. Consideremos el conjunto de dominios X como un conjunto de personas y dejar que el conjunto de gama Y sea sus números de teléfono. Asumiendo que una persona pueda tener más de un número de teléfono, siendo la relación considerado no es una función. Sin embargo, si dibujamos una relación entre los nombres de los residentes y sus fecha de nacimiento con el nombre establecido como dominio, luego esto se convierte en una relación de buen comportamiento y, por lo tanto, una función. Esto significa que, si bien todas las funciones son relaciones, no todas las relaciones son funciones. En caso de una función dada una x, uno obtiene uno y exactamente uno y para cada par ordenado (*x* , *y*).

Por ejemplo, consideremos los siguientes dos relaciones.

A: {(3, –9), (5, 8), (7, –6), (3, 9), (6, 3)}.
B: {(5, 8), (7, 8), (3, 8), (6, 8)}.

¿Son estas funciones también?

En el caso de la relación A, el dominio es todo el valores x, es decir, {3, 5, 6, 7}, y el rango es todo el valores y, es decir, {–9, –6, 3, 8, 9}.

La relación A no es una función, ya que hay dos diferentes valores de rango, –9 y 9, para el mismo Valor x de 3.

En el caso de la relación B, el dominio es el mismo que para A, es decir, {3, 5, 6, 7}. Sin embargo, el rango es un elemento único {8}. Esto califica como un ejemplo de una función incluso si todos los valores de x están asignados al mismo valor de y. Aquí, cada valor de x es distinto y, por lo tanto, la función se comporta bien. Relación B puede estar representado por la ecuación y = 8.

La característica de una función puede ser verificada utilizando una prueba de línea vertical, que se indica a continuación:

Dada la gráfica de una relación, si se puede dibujar una línea vertical que cruza el gráfico en más de un lugar, entonces la relación no es una función.

Figura 14.7. Prueba de línea vertical para función

En este ejemplo, ambas líneas L1 y L2 cortan el gráfico para la relación tres veces. Esto significa que para el mismo valor x, hay tres diferentes valores de y para cada caso. Por lo tanto, la relación no es Una función.

Page 261

2. Lógica básica [1 *, c1]

2.1. Lógica proposicional

Una proposición es una afirmación que es cierta o falso, pero no ambos. Consideremos declarativo oraciones para las cuales es significativo asignar cualquiera de los dos valores de estado: *verdadero* o *falso* . Algunos A continuación se dan ejemplos de proposiciones.

1. El sol es una estrella.
2. Los elefantes son mamíferos.
3. 2 + 3 = 5.

Sin embargo, a + 3 = b no es una propuesta, como lo es ni verdadero ni falso. Depende de los valores de las variables *a* y *b* .

La Ley del Medio Excluido: Para cada propuesta sición p, o p es verdadero o p es falso.

La Ley de Contradicción: para cada propuesta ción p, no es el caso de que p sea verdadero y falso.

La lógica proposicional es el área de la lógica que trata con proposiciones. Una tabla de verdad muestra las relaciones entre los valores de verdad de proposiciones

Una variable booleana es aquella cuyo valor es verdadero o falso. Las operaciones de bit de computadora corresponden a operaciones lógicas de variables booleanas.

Los operadores lógicos básicos, incluida la negación (¬ p), conjunción (p ∧ q), disyunción (p ∨ q), exclusivo o (p ⊕ q), y la implicación (p → q) son para ser estudiado Las propuestas compuestas pueden ser formado usando varios operadores lógicos.

Una proposición compuesta que siempre es cierta es una tautología. Una proposición compuesta que siempre es falso es una contradicción. Una propuesta compuesta eso no es una tautología ni una contradicción es una contingencia.

Propuestas compuestas que siempre tienen el el mismo valor de verdad se llama lógicamente equivalente (denotado por ≡). Algunos de los equivalentes comunes las ausencias son:

Leyes de identidad:
p ∧ T ≡ p p ∨ F ≡ p

Leyes de dominación:
p ∨ T ≡ T p ∧ F ≡ F

Leyes idempotentes:
p ∨ p ≡ p p ∧ p ≡ p

Ley de doble negación:
¬ (¬ p) ≡ p

Leyes conmutativas:
p ∨ q ≡ q ∨ pp ∧ q ≡ q ∧ p

Leyes asociativas:
(p ∨ q) ∨ r ≡ p ∨ (q ∨ r)
(p ∧ q) ∧ r ≡ p ∧ (q ∧ r)

Leyes distributivas:
p ∨ (q ∧ r) ≡ (p ∨ q) ∧ (p ∨ r)
p ∧ (q ∨ r) ≡ (p ∧ q) ∨ (p ∧ r)

Las leyes de De Morgan:
¬ (p ∧ q) ≡ ¬ p ∨ ¬ q ¬ (p ∨ q) ≡ ¬ p ∧ ¬ q

2.2. Lógica Predicada

Un predicado es una plantilla de frase verbal que describe una propiedad de objetos o una relación entre objetos representados por las variables. por ejemplo, en la oración, *La flor es roja,* el La plantilla *es roja* es un predicado. Describe el propiedad de una flor. El mismo predicado puede ser usado en otras oraciones también.

Los predicados a menudo reciben un nombre, por ejemplo, "Rojo" o simplemente "R" se puede utilizar para representar la predi- Cate *es rojo* . Asumiendo R como el nombre de la predicate *es rojo* , oraciones que afirman que un objeto es del el color rojo se puede representar como *R (x)* , donde *x* representa un objeto arbitrario. *R (x) se* lee como *x es rojo* .

Los cuantificadores permiten declaraciones sobre todo el col- Lecciones de objetos en lugar de tener que enumerar comió los objetos por su nombre.

El cuantificador universal ∀x afirma que un sentence es verdadero para todos los valores de la variable x.

Por ejemplo, Tigerx Tigre (x) → Mamífero (x) significa que todos los tigres son mamíferos.

El cuantificador existencial ∃x afirma que un sentence es verdadero para al menos un valor de la variable x.

Por ejemplo, Tigerx Tigre (x) → Hombre-comedor (x) significa existe al menos un tigre que es devorador de hombres.

Por lo tanto, mientras que la cuantificación universal utiliza implicación, la cuantificación existencial naturally usa conjunción.

Página 262

Una variable *x* que se introduce en una lógica la expresión de un cuantificador está vinculada a la más cercana cuantificador adjunto.

Se dice que una variable es una variable libre si no es vinculado a un cuantificador.

Del mismo modo, en una programación estructurada en bloques lenguaje, una variable en una expresión lógica se refiere al cuantificador más cercano dentro de cuyo alcance aparece.

Las declaraciones utilizadas en una prueba incluyen axiomas y postulados que son esencialmente el subyacente supuestos sobre estructuras matemáticas, el hipótesis del teorema a probar, y pre Teoremas vivamente probados.

Un teorema es una declaración que se puede mostrar a ser cierto.

Un lema es un teorema simple utilizado en la demostración. de otros teoremas.

Por ejemplo, en ∃x (Cat (x) ∧ ∀x (Black (x))) x en negro (x) está universalmente cuantificado. La expresión implica que los gatos existen y todo es negro.

La lógica proposicional se queda corta en la representación de muchas afirmaciones que se usan en ciencias de la computación, inteligence y matemática. Tampoco se compara equivalencia y algunos otros tipos de relación entre proposiciones.

Por ejemplo, la afirmación *a es mayor que 1* no es una proposición porque no se puede inferir si es verdadero o falso sin conocer el valor de *a* . Por lo tanto, la lógica proposicional no puede tratar con tales oraciones. Sin embargo, tales afirmaciones aparecer con bastante frecuencia en matemáticas y queremos inferir sobre esas afirmaciones. Además, el patrón involucrado en los siguientes dos equivalentes lógicos las ausencias no pueden ser capturadas por proposicional lógica: " *No todos los hombres son fumadores* " y " *Algunos hombres no fumes* "Cada una de estas dos proposiciones se trata independientemente en lógica proposicional. No existe un mecanismo en la lógica proposicional para averiguar si los dos son o no equivalentes a unos y otros. Por lo tanto, en lógica proposicional, cada propuesta equivalente se trata individualmente en lugar de tratar con una fórmula general que cubre todas las equivalencias colectivamente.

Se supone que la lógica predicada es más poderosa. lógica inteligente que aborda estos problemas. En un sentido, lógica de predicados (también conocida como lógica de primer orden o cálculo predicado) es una extensión de la propuesta lógica fraccionaria a fórmulas que involucran términos y predicados

3. Técnicas de prueba

[1 *, c1]

Una prueba es un argumento que establece rigurosamente La verdad de una declaración. Las pruebas pueden ser ellas mismas representado formalmente como estructuras discretas.

Un corolario es una proposición que puede establecerse listado directamente de un teorema que ha sido demostrado.

Una conjetura es una declaración cuyo valor de verdad es desconocido.

Cuando se encuentra la prueba de una conjetura, la conjecTure se convierte en un teorema. Muchas conjeturas se muestran falsos y, por lo tanto, no son teoremas.

3.1. Métodos de probar teoremas

Prueba directa La prueba directa es una técnica para establecer Lish que la implicación p → q es verdadera mostrando que q debe ser verdadero cuando p es verdadero.

Por ejemplo, para mostrar que si n es impar, entonces $n_2 - 1$ es par, supongamos que n es impar, es decir, n = 2k + 1 para algunos entero k:

$n_2 - 1 = (2k + 1)_2 - 1 = 4k_2 + 4k + 1 - 1 = 4k_2 + 4k = 4 (k_2 + k)$.

Como los dos primeros términos del lado derecho (RHS) son números pares independientemente del valor de k, el lado izquierdo (LHS) (es decir, n_2) es un número impar número. Por lo tanto, $n_2 - 1$ es par.

Prueba por contradicción. Una proposición p es verdadera por contradicción si se prueba basado en la verdad de la implicación ¬ p → q donde q es una contradicción.

Por ejemplo, para mostrar que la suma de 2x + 1 y 2y - 1 es par, supongamos que la suma de 2x + 1 y 2y - 1 es impar. En otras palabras, 2 (x + y), que es múltiplo de 2, es impar. Esto es una contradicción. Por lo tanto, la suma de 2x + 1 y 2y - 1 es par.

Una regla de inferencia es un patrón que establece que si un conjunto de premisas son todas verdaderas, entonces puede ser dedujo que una cierta declaración de conclusión es cierto. Las reglas de referencia de la suma, simplificación y conjunción deben estudiarse.

Prueba por inducción. Se realiza la prueba por inducción. en dos fases Primero, la proposición es establecida se consideró cierto para un caso base, generalmente para el

Página 263

entero positivo 1. En la segunda fase, se establece Listed que si la proposición es válida para un arbitrario entero positivo *k,* entonces también debe mantenerse para eltarea. siguiente entero mayor, *k + 1* . En otras palabras, prueba por inducción se basa en la regla de inferencia que nos dice que la verdad de una secuencia infinita de se establecen las proposiciones P (n), ∀n ∈ [1 ... ∞] si P (1) es verdadero, y en segundo lugar, ∀k ∈ [2 ... n] si P (k) → P (k + 1).

Cabe señalar aquí que, para una prueba matemática, inducción matemática, no se supone que P (k) es verdadero para todos los enteros positivos k. Probar un teorem o proposición solo requiere que establezcamos que si se supone que P (k) es cierto para cualquier arbitrario entero positivo k, entonces P (k + 1) también es cierto. los corrección de la inducción matemática como válida la técnica de prueba está más allá de la discusión de la corriente Alquiler de texto. Probemos la siguiente proposición usando inducción.

Proposición: *La suma del primer n positivo impar enteros P (n) es* n_2.

Paso básico: la proposición es verdadera para n = 1 como P (1) = 1_2 = 1. El paso base está completo.

Paso inductivo: la hipótesis de inducción (IH)

n_2 maneras, y si estas tareas no se pueden hacer en el al mismo tiempo, entonces hay $n_1 + n_2$ formas de hacer cualquiera

- Si A y B son conjuntos disjuntos, entonces | A ∪ B | = | A | + | B |.
- En general si A1, A2,…. Un son disjuntos establece, entonces | A1 ∪ A2 ∪… ∪ An | = | A1 | + | A2 | +… + | Un |.

Por ejemplo, si hay 200 atletas haciendo eventos de sprint y 30 atletas que participan en el evento de salto de longitud, entonces, ¿de cuántas maneras para elegir un atleta que sea velocista o un puente largo?

Usando la regla de la suma, la respuesta sería 200 + 30 = 230.

La regla del producto establece que si una tarea t_1 puede ser hecho en n_1 formas y se puede hacer una segunda tarea t_2 de n_2 maneras después de que se haya realizado la primera tarea, luego Hay $n_1 * n_2$ formas de hacer el procedimiento.

- Si A y B son conjuntos disjuntos, entonces | A × B | = | A | * | B |.

es que la proposición es verdadera para n = k, siendo k un entero positivo arbitrario k.

$\therefore 1 + 3 + 5 + \ldots + (2k - 1) = k^2$

Ahora, se debe demostrar que P (k) → P (k + 1).

$P(k + 1) = 1 + 3 + 5 + \ldots + (2k - 1) + (2k + 1)$
$= P(k) + (2k + 1)$
$= k^2 + (2k + 1)$ [usando IH]
$= k^2 + 2k + 1$
$= (k + 1)^2$

Por lo tanto, se muestra que si la proposición es verdadera para n = k, entonces también es cierto para n = k + 1.

El paso base junto con el paso inductivo de la prueba muestra que P (1) es verdadero y el condicional la declaración P (k) → P (k + 1) es verdadera para todos los positivos enteros k. Por lo tanto, la proposición está probada.

4. Conceptos básicos de contar

[1 * c6]

La regla de suma establece que si se puede hacer una tarea t en n_1 formas y una segunda tarea t_2 se puede hacer en

• En general, si A1, A2, ..., An son conjuntos disjuntos, entonces | A1 × A2 ×… × An | = | A1 | * | A2 | * ... * | Un |.

Por ejemplo, si hay 200 atletas haciendo eventos de sprint y 30 atletas que participan en el evento de salto de longitud, entonces, ¿de cuántas maneras allí para elegir dos atletas para que uno sea un velocista y el otro es un puente largo?

Usando la regla del producto, la respuesta sería 200 * 30 = 6000.

El *principio de inclusión-exclusión* establece que si una tarea t_1 se puede hacer de n_1 maneras y un segundo la tarea t_2 se puede hacer de n_2 formas al mismo tiempo con t_1, luego para encontrar el número total de formas en que se pueden hacer dos tareas, restar el número de maneras de hacer ambas tareas desde $n_1 + n_2$.

• Si A y B no son disjuntos, | A ∪ B | = | A | + | B | - | A ∩ B |.

En otras palabras, el principio de inclusión: la exclusión tiene como objetivo garantizar que los objetos en el La intersección de dos conjuntos no se cuenta más de una vez.

La recursión es el término general para la práctica. de definir un objeto en términos de sí mismo. Existen algoritmos recursivos, funciones definidas recursivamente relaciones, relaciones, conjuntos, etc.

Una función recursiva es una función que llama sí mismo. Por ejemplo, definimos f (n) = 3 * f (n - 1) para todos n ∈ N y n ≠ 0 yf (0) = 5.

Un algoritmo es recursivo si resuelve un problema. reduciéndolo a una instancia del mismo problema con una entrada más pequeña

Se dice que un fenómeno es aleatorio si individualmente Los resultados actuales son inciertos, pero el patrón a largo plazo El término de muchos resultados individuales es predecible.

La probabilidad de cualquier resultado para una carrera fenómeno dom es la proporción de veces que el el resultado ocurriría en una serie muy larga de repeticiones

La probabilidad P (A) de cualquier evento A satisface 0 ≤ P (A) ≤ 1. Cualquier probabilidad es un número entre 0 y 1. Si S es el espacio muestral en una probabilidad ity model, the P (S) = 1. Todos los resultados posibles juntos deben tener una probabilidad de 1.

Dos eventos A y B son disjuntos si tienen no hay resultados en común y por eso nunca puede ocurrir juntos. Si A y B son dos eventos disjuntos, P (A o B) = P (A) + P (B). Esto se conoce como el addi- regla para eventos disjuntos.

Si dos eventos no tienen resultados en común, la probabilidad de que uno u otro ocurra es el suma de sus probabilidades individuales.

La permutación es una disposición de objetos en que el orden importa sin repetición. Uno puede elegir r objetos en un orden particular de un total de n objetos usando $_nP_r$ maneras, donde, $_np_r$ = ¡norte! / (n - r) !. Varias notaciones como $_nP_r$ y P (n, r) se usan para representar el número de permutaciones de un conjunto de n objetos tomados r a la vez.

La combinación es una selección de objetos en los que

5. Gráficos y árboles

[1 *, c10, c11]

5.1. Gráficos

Un gráfico G = (V, E) donde V es el conjunto de vértices (nodos) y E es el conjunto de aristas. Los bordes también son referidos como arcos o enlaces.

Figura 14.8. Ejemplo de un gráfico

F es una función que asigna el conjunto de aristas E a un conjunto de pares de elementos ordenados o no ordenados V. Por ejemplo, en la Figura 14.8, G = (V, E) donde V = {A, B, C}, E = {e1, e2, e3}, y F = {(e1, (A, C)), (e2, (C, B)), (e3, (B, A))}.

El gráfico de la figura 14.8 es un gráfico simple que consiste en un conjunto de vértices o nodos y un conjunto de bordes que conectan pares desordenados.

Los bordes en gráficos simples no están dirigidos. Dichos gráficos también se denominan no dirigidos gráficos

Por ejemplo, en la Figura 14.8, (e1, (A, C)) puede ser reemplazado por (e1, (C, A)) como el par entre los vértices A y C no están ordenados. Esto es bueno para los otros dos bordes también.

el orden no importa sin repetición. Esta es diferente de una permutación porque el orden no importa. Si el orden solo se modifica (y no los miembros), entonces no hay una nueva combinación formado. Uno puede elegir r objetos en cualquier orden de un total de n objetos usando $_nC_r$ maneras, donde, $_nC_r = n! / [r! * (n - r)!]$.

En un multigrafo, más de un borde puede conecta los mismos dos vértices. Dos o más conectan ing bordes entre el mismo par de vértices pueden reflejar múltiples asociaciones entre el mismo dos vértices Tales bordes se llaman paralelos o Múltiples aristas.

Por ejemplo, en la Figura 14.9, los bordes e3 y e4 están ambos entre A y B. La figura 14.9 es un multigrafo donde los bordes e3 y e4 son múltiples bordes

Un gráfico dirigido G = (V, E) consiste en un conjunto de vértices V y un conjunto de aristas E que están ordenados pares de elementos de V. Un gráfico dirigido puede con- Tain Loops.

Por ejemplo, en la Figura 14.11, G = (V, E) donde V = {A, B, C}, E = {e1, e2, e3}, y F = {(e1, (A, C)), (e2, (B, C)), (e3, (B, A))}.

Figura 14.9. Ejemplo de un multigrafo

En un *pseudograma* , los bordes conectan un nodo a en sí están permitidos. Tales bordes se llaman bucles.

Figura 14.12. Ejemplo de un gráfico ponderado

En un gráfico ponderado G = (V, E), cada arista tiene un peso asociado con él. El peso de un borde normalmente representa el valor numérico asociado con la relación entre el correspondiente dos vértices

Por ejemplo, en la Figura 14.12, los pesos para los bordes e1, e2 y e3 se consideran 76, 93, y 15 respectivamente. Si los vértices A, B y C representar tres ciudades en un estado, los pesos, para ejemplo, podrían ser las distancias en millas entre Estas ciudades.

Deje G = (V, E) ser un gráfico no dirigido con conjunto de bordes E. Entonces, para un borde e ∈ E donde e = {u, v}, a menudo se utilizan las siguientes terminologías:

- u, v se dice que son *adyacentes* o *vecinos* o *conectados* .
- el borde e *incide* con los vértices u y v.
- edge e *conecta* u y v.
- los vértices u y v son *puntos finales* para el borde e.

Si vértice v ∈ V, el conjunto de vértices en la unidad gráfico recitado G (V, E), luego:

- el *grado* de v, deg (v), es su número de inci- bordes de abolladuras, excepto que cualquier auto-bucle es contado dos veces.

Figura 14.10. Ejemplo de un seudografo

Por ejemplo, en la Figura 14.10, el borde e4 tanto comienza y termina en B. La figura 14.10 es un pseudo- gráfico en el que e4 es un bucle.

Figura 14.11. Ejemplo de un gráfico dirigido

Página 266

- un vértice con grado 0 se llama *aislado vértice* .
- un vértice de grado 1 se llama *colgante vértice* .

Deje G (V, E) ser un gráfico dirigido. Si e (u, v) es un borde de G, entonces las siguientes terminologías son utilizado a menudo:

- u es *adyacente a* v, y v es *adyacente a* u.
- e *viene de* u y *va a* v.
- e *conecta* u a v, o e *va de* u a v.
- el *vértice inicial* de e es u.
- el *vértice terminal* de e es v.

Si el vértice v está en el conjunto de vértices para el gráfico dirigido G (V, E), luego

- *en grado* de v, $\deg^-(v)$, es el número de bordes que van a v, es decir, para el cual v es el ter- vértice minal.
- *grado* de v, $\deg^+(v)$, es el número de bordes que provienen de v, es decir, para los cuales v es el vértice inicial
- *grado* de v, $\deg(v) = \deg^-(v) + \deg^+(v)$, es el suma de in-degree y out-degree.
- un bucle en un vértice contribuye 1 a ambos grado y grado de este vértice.

Cabe señalar que, siguiendo las definiciones arriba, el grado de un nodo no cambia si consideramos que sus bordes están dirigidos o no dirigidos.

En un gráfico no dirigido, una ruta de longitud n desde u to v es una secuencia de n bordes adyacentes de ver- tex u al vértice v.

- Una ruta es un *circuito* si u = v.
- Un camino *atraviesa* los vértices a lo largo de él.
- Una ruta es *simple* si no contiene más borde de una vez.

Un ciclo en n vértices C_n para cualquier $n \geq 3$ es un sim- gráfico completo donde $V = \{v_1, v_2, ..., v_n\}$ y $E = \{\{v_1, v_2\}, \{v_2, v_3\},\ldots, \{v_{n-1}, v_n\}, \{v_n, v_1\}\}$.

Por ejemplo, la figura 14.13 ilustra dos ciclos de longitud 3 y 4.

Figura 14.13. Ejemplo de ciclos C_3 y C_4

Una lista de adyacencia es una tabla con una fila por vértice, enumerando sus vértices adyacentes. La adyacencia listado para un gráfico dirigido mantiene un listado de los nodos terminales para cada uno de los vértices en el grafico.

Vértice	Proximidad Lista
UNA	ANTES DE CRISTO
segundo	A B C
do	A, B

Figura 14.14. Listas de adyacencia para gráficos en las figuras 14.10 y 14.11

Por ejemplo, la figura 14.14 ilustra el adyacente listas de cency para el pseudografo en la figura 14.10 y el gráfico dirigido en la figura 14.11. Como el fuera del grado de vértice C en la figura 14.11 es cero, no hay entrada contra C en la lista de adyacencia.

Diferentes representaciones para un gráfico, como matriz de adyacencia, matriz de incidencia y adyacencia listas de privacidad: deben estudiarse.

5.2. Arboles

Un árbol T (N, E) es una estructura de datos jerárquica de n = | N | nodos con un nodo raíz especialmente designado R mientras que los n - 1 nodos restantes forman subárboles debajo del nodo raíz R. El número de aristas | E | en un árbol siempre sería igual a | N | - 1.

El subárbol en el nodo X es el subgráfico de la árbol que consiste en el nodo X y sus descendientes y todos los bordes inciden en esos descendientes. Como un alternativamente a esta definición recursiva, un árbol puede ser definido como un gráfico conectado no dirigido con No hay circuitos simples.

Página 267

en el nivel 0. Alternativamente, el nivel de un nodo X es el longitud de la ruta única desde la raíz del árbol al nodo X.

Por ejemplo, el nodo raíz A está en el nivel 0 en la figura 14.15. Los nodos B, C y D están en el nivel 1. El los nodos restantes en la Figura 14.15 están todos en el nivel 2.

Figura 14.15. Ejemplo de un árbol

Sin embargo, uno debe recordar que un árbol es estrictamente de naturaleza jerárquica en comparación con un gráfico, que es plano. En el caso de un árbol, un pedido par se construye entre dos nodos como padre y niño. Cada nodo hijo en un árbol está asociado con un solo nodo padre, mientras que esta restricción ción no tiene sentido para un gráfico donde no existe asociación padre-hijo.

Un gráfico no dirigido es un árbol si y solo si hay una ruta simple única entre dos de sus vértices

La figura 14.15 presenta un árbol T (N, E) donde el conjunto de nodos N = {A, B, C, D, E, F, G, H, I, J, K}. El conjunto de bordes E es {(A, B), (A, C), (A, D), (B, E), (B, F), (B, G), (C, H), (C, I), (D, J), (D, K)}.

El padre de un nodo no raíz v es el único nodo u con un borde dirigido de u a v. Cada El nodo en el árbol tiene un nodo padre único, excepto La raíz del árbol.

Por ejemplo, en la Figura 14.15, el nodo raíz A es el nodo padre para los nodos B, C y D. De manera similar, B es el padre de E, F, G, etc. La raíz el nodo A no tiene ningún padre.

Un nodo que tiene hijos se llama interno nodo.

Por ejemplo, en la Figura 14.15, nodo A o nodo B son ejemplos de nodos internos.

El grado de un nodo en un árbol es el mismo que su numero de niños.

Por ejemplo, en la Figura 14.15, el nodo raíz A y su hijo B son ambos de grado 3. Nodos C y D tener grado 2.

La distancia de un nodo desde el nodo raíz en términos de número de saltos se llama su *nivel* . Nodos en un árbol están en diferentes niveles. El nodo raíz es

La altura de un árbol es el máximo del nivel Els de nodos en el árbol.

Por ejemplo, en la figura 14.15, la altura del el árbol es 2.

Un nodo se llama *hoja* si no tiene hijos. los El grado de un nodo hoja es 0.

Por ejemplo, en la Figura 14.15, los nodos E a través K son todos los nodos hoja con grado 0.

Los antepasados o predecesores de un no raíz nodo X son todos los nodos en la ruta desde la raíz hasta nodo X.

Por ejemplo, en la figura 14.15, los nodos A y D formar el conjunto de antepasados para J.

Los sucesores o descendientes de un nodo X son todos los nodos que tienen X como ancestro. Para un árbol con n nodos, todos los n - 1 nodos restantes son sucesores del nodo raíz.

Por ejemplo, en la figura 14.15, el nodo B tiene éxito cesadores en E, F y G.

Si el nodo X es un ancestro del nodo Y, entonces el nodo Y es un sucesor de X.

Dos o más nodos que comparten el mismo padre Los nodos se denominan nodos *hermanos* .

Por ejemplo, en la figura 14.15, los nodos E y G son hermanos Sin embargo, los nodos E y J, aunque del mismo nivel, no son nodos hermanos.

Dos nodos hermanos son del mismo nivel, pero dos nodos en el mismo nivel no son necesariamente hermanos

Un árbol se llama *árbol ordenado* si la relación La posición tive de las ocurrencias de los nodos hijos es significativo.

Por ejemplo, un árbol genealógico es un árbol ordenado si, por regla general, aparece el nombre de un hermano mayor siempre antes (es decir, a la izquierda de) el más joven hermano.

En un árbol desordenado, la posición relativa de ocurrencias entre los hermanos no lleva cualquier significado y puede ser alterado arbitrariamente.

Se forma un árbol binario con cero o más nodos. donde hay un nodo raíz R y todos los restantes los nodos forman un par de subárboles ordenados bajo el nodo raíz

Página 268

En un árbol binario, ningún nodo interno puede tener más de dos niños Sin embargo, uno debe considerar que además de este criterio en cuanto al grado de nodos internos, siempre se ordena un árbol binario. Si las posiciones de los subárboles izquierdo y derecho para cualquier nodo en el árbol se intercambia, luego un nuevo árbol es derivado.

Figura 14.16. Ejemplos de árboles binarios

Por ejemplo, en la figura 14.16, los dos binarios los árboles son diferentes según las posiciones de los sucesor de los hijos de A son diferentes en los dos árboles.

Figura 14.18. Ejemplo de árboles binarios completos

Curiosamente, siguiendo las definiciones anteriores, el árbol de la figura 14.18 (b) es completo pero no árbol binario completo ya que el nodo B solo tiene un hijo en D. Por el contrario, el árbol de la figura 14.17 es un árbol completo. —Pero no completo — árbol binario, como los niños de B ocurren en el árbol mientras que los hijos de C No aparece en el último nivel.

Figura 14.17. Ejemplo de un árbol binario completo

Según [1 *], un árbol binario se llama completo árbol binario si cada nodo interno tiene exactamente dos niños.

Por ejemplo, el árbol binario en la figura 14.17 es un árbol binario completo, ya que los dos internos los nodos A y B son de grado 2.

Un árbol binario completo siguiendo la definición. arriba también se conoce como un *árbol estrictamente binario*.

Por ejemplo, ambos árboles binarios en la figura 14.18 Son árboles binarios completos. El arbol en la figura 14.18 (a) es un binario completo y completo árbol. Un árbol binario completo tiene todos sus niveles, excepto posiblemente el último, lleno hasta el tope. En caso de que el último nivel de un árbol binario completo sea no lleno, los nodos ocurren desde las posiciones más a la izquierda disponible.

Un árbol binario de altura H está equilibrado si todas sus los nodos foliares ocurren en los niveles H o H - 1.

Por ejemplo, los tres árboles binarios en las figuras 14.17 y 14.18 son árboles binarios balanceados.

Hay como máximo 2 hojas $_H$ en un árbol binario de altura H. En otras palabras, si un árbol binario con L las hojas están llenas y equilibradas, entonces su altura es $H = \lceil \log_2 L \rceil$.

Por ejemplo, esta afirmación es verdadera para el dos árboles en las Figuras 14.17 y 14.18 (a) como ambos Los árboles están llenos y equilibrados. Sin embargo, el expreso La sección anterior no coincide con el árbol de la Figura 14.18 (b) ya que no es un árbol binario completo.

Un árbol de búsqueda binaria (BST) es un tipo especial de árbol binario en el que cada nodo contiene un distinto valor clave y el valor clave de cada nodo en el el árbol es menor que cada valor clave en su subárbol derecho y mayor que cada valor clave en su subárbol izquierdo.

Un algoritmo transversal es un procedimiento para el sistema visitando temáticamente cada nodo de un árbol binario. Los recorridos de los árboles se pueden definir de forma recursiva.

Si T es un árbol binario con raíz R y el resto los nodos ing forman un par ordenado de izquierda no nula subárbol T_L y subárbol derecho no nulo T_R debajo de R, entonces la función transversal de preorden PreOrder (T) Se define como:

Preordenar (T) = R, Preordenar (T_L), Preordenar (T_R) ... eqn. 1

Página 269

El proceso recursivo de encontrar el preorden el recorrido de los subárboles continúa hasta que los árboles se encuentran nulos. Aquí, las comas tienen sido utilizado como delimitadores en aras de mejoras legibilidad.

El postorder y en orden pueden ser similares definido usando la ecuación. 2 y eqn. 3 respectivamente.

PostOrder (T) = PostOrder (T_L), PostOrder (T_R), R ... ecuación 2

InOrder (T) = InOrder (T_L), R, InOrder (T_R) ... ecuación 3

Figura 14.19. Un árbol de búsqueda binaria

Por ejemplo, el árbol en la figura 14.19 es un binario Árbol de búsqueda (BST). El preorden, postorder y se dan salidas transversales en orden para el BST a continuación en su respectivo orden.

Salida de pedido anticipado: 9, 5, 2, 1, 4, 7, 6, 8, 13, 11,

la aleatoriedad se ha definido en la sección 4 de este KA. Aquí, comencemos con los conceptos detrás de distribución de probabilidad y probabilidad discreta.

Un modelo de probabilidad es una descripción matemática. ción de un fenómeno aleatorio que consiste en dos partes: un espacio muestral S y una forma de asignar probabilidades de eventos. El espacio muestral define el conjunto de todos los resultados posibles, mientras que un evento es un subconjunto de un espacio muestral que representa una posición resultado posible o un conjunto de resultados.

Una variable aleatoria es una función o regla que asigna un número a cada resultado. Básicamente es solo un símbolo que representa el resultado de un experimentar.

Por ejemplo, sea X el número de cabezas cuando el experimento lanza una moneda n veces. Del mismo modo, que S sea la velocidad de un automóvil registrado en un detector de radar.

Los valores para una variable aleatoria podrían ser dis- Creta o continua dependiendo del experimento.

Una variable aleatoria discreta puede contener todas las pos- posibles resultados sin perder ninguno, aunque podría tomar una cantidad infinita de tiempo.

Una variable aleatoria continua se utiliza para medir Seguro un número incontable de valores, incluso si un Se da una cantidad infinita de tiempo.

Por ejemplo, si una variable aleatoria X representa un resultado que es un número real entre 1 y 100, entonces X puede tener un número infinito de valores ues. Uno nunca puede enumerar todos los resultados posibles para X incluso si se permite una cantidad infinita de tiempo. Aquí, X es una variable aleatoria continua. En al contrario, para el mismo intervalo de 1 a 100, otra variable aleatoria Y se puede usar para enumerar todos

10, 15
Salida posterior al pedido: 1, 4, 2, 6, 8, 7, 5, 10, 11, 15, 13, 9
Salida en orden: 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15

Más discusión sobre los árboles y su uso ha incluido en la sección 6, Estructura de datos y Rep. presentación, de las Fundaciones de Computación KA.

6. Probabilidad discreta

[1 *, c7]

La probabilidad es la descripción matemática de aleatoriedad Definición básica de probabilidad y los valores enteros en el rango. Aquí, Y es un dis- Creta variable aleatoria.

Una letra mayúscula, digamos X, representará El *nombre* de la variable aleatoria. Su minúscula La contraparte, x, representará el *valor* del rango Dom variable.

La probabilidad de que la variable aleatoria X igual x es:

P (X = x) o, más simplemente, P (x).

Una función de distribución de probabilidad (densidad) es una tabla, fórmula o gráfico que describe el valor ues de una variable aleatoria y la probabilidad asociada ciado con estos valores.

Probabilidades asociadas al azar discreto Las variables tienen las siguientes propiedades:

yo. $0 \le P(x) \le 1$ para todo x
ii) $\Sigma P(x) = 1$

Una distribución de probabilidad discreta puede ser representada presentado como una variable aleatoria discreta.

X	1	2	3	4 4	5 5	6 6
P (x)	1/6	1/6	1/6	1/6	1/6	1/6

Figura 14.20. Una función de probabilidad discreta para un balanceo Morir

La media μ de un modelo de distribución de probabilidad es la suma de los términos del producto por individuo eventos y su probabilidad de resultado. En otra palabras, para los posibles resultados $x_1, x_2, \ldots, x_n$ en un espacio muestral S si p_k es la probabilidad de viene x_k, la media de esta probabilidad sería $\mu = x_1 p_1 + x_2 p_2 + \ldots + x_n p_n$.

Por ejemplo, la media de la probabilidad den- La distribución de la figura 14.20 sería

$1 * (1/6) + 2 * (1/6) + 3 * (1/6) + 4 * (1/6) + 5 * (1/6) + 6 * (1/6)$
$= 21 * (1/6) = 3.5$

Aquí, el espacio muestral se refiere al conjunto de todos posibles resultados

La varianza s_2 de un modelo de probabilidad discreta es: $s_2 = (x_1 - \mu)_2 p_1 + (x_2 - \mu)_2 p_2 + \ldots + (x_k - \mu)_2 p_k$. La *desviación estándar* s es la raíz cuadrada de diferencia.

Por ejemplo, para la distribución de probabilidad en Figura 14.20, la variación σ_2 sería

$s_2 = [(1 - 3.5)_2 * (1/6) + (2 - 3.5)_2 * (1/6) + (3 - 3.5)_2 * (1/6) + (4 - 3.5)_2 * (1/6) + (5 - 3.5)_2 * (1/6) + (6 - 3.5)_2 * (1/6)]$
$= (6.25 + 2.25 + 0.25 + 0.5 + 2.25 + 6.25) * (1/6)$
$= 17.5 * (1/6)$
$= 2.90$

∴ desviación estándar s =

Estos números de hecho apuntan a derivar el promedio valor de edad de experimentos repetidos. Esto es basado en el fenómeno más importante enón de probabilidad, es decir, el valor promedio de Es probable que los experimentos repetidos estén cerca del valor esperado de un experimento. Además, es más probable que el valor promedio esté más cerca de el valor esperado de cualquier experimento como el El número de experimentos aumenta.

7. Máquinas de estado finito

[1 *, c13]

Un sistema informático puede abstraerse como un mapa ping de estado a estado impulsado por entradas. En otra palabras, un sistema puede considerarse como una transición función $T: S \times I \rightarrow S \times O$, donde S es el conjunto de estados e I, O son las funciones de entrada y salida.

Si el conjunto de estados S es finito (no infinito), el sistema tem se llama *máquina de estado finito* (FSM).

Alternativamente, una máquina de estados finitos (FSM) es un abstracción matemática compuesta de un finito número de estados y transiciones entre esos estados. Si el dominio $S \times I$ es razonablemente pequeño, entonces uno puede especificar T explícitamente usando diagramas similar a un diagrama de flujo para ilustrar la forma en que logi c Flujos para diferentes entradas. Sin embargo, esto es práctico. tical solo para máquinas que tienen un tamaño muy pequeño capacidad de información.

Un FSM tiene una memoria interna finita, una entrada característica que lee símbolos en una secuencia y una a la vez, y una función de salida.

El funcionamiento de un FSM comienza desde el principio. estado, pasa por transiciones dependiendo de la entrada a diferentes estados, y puede terminar en cualquier estado válido. Sin embargo, solo unos pocos de todos los estados marcan un éxito flujo de operación cesante. Estos se llaman *aceptar estados* .

La capacidad de información de un FSM es $C = \log |S|$. Por lo tanto, si representamos una máquina que tiene una capacidad de información de bits C como FSM, luego su gráfico de transición de estado tendrá $|S| = 2$ nodos c.

Una máquina de estados finitos se define formalmente como $M = (S, I, O, f, g, s_0)$.

S es el conjunto de estados;
I es el conjunto de símbolos de entrada;
O es el conjunto de símbolos de salida;

f es la función de transición de estado;

Página 271

g es la función de salida;
y s_0 es el estado inicial.

Dada una entrada x ∈ I en el estado S_k, el FSM hace una transición al estado S_h después del tránsito estatal activa la función f y produce una salida y ∈ O utilizando la función de salida g.

Figura 14.21. Ejemplo de un FSM

Por ejemplo, la figura 14.21 ilustra un FSM con S_0 como estado inicial y S_1 como estado final. Aquí, S = {S_0, S_1, S_2}; I = {0, 1}; O = {2, 3}; f (S_0, 0) = S_2, f (S_0, 1) = S_1, f (S_1, 0) = S_2, f (S_1, 1) = S_2, f (S_2, 0) = S_2, f (S_2, 1) = S_0; g (S_0, 0) = 3, g (S_0, 1) = 2, g (S_1, 0) = 3, g (S_1, 1) = 2, g (S_2, 0) = 2, g (S_2, 1) = 3.

Corriente Estado	Entrada	
	0 0	1
S_0	S_2	S_1
S_1	S_2	S_2
S_2	S_2	S_0

(una)

Corriente Estado	Salida		Estado	
	Entrada		Entrada	
	0 0	1	0 0	1
S_0	3	2	S_2	S_1
S_1	3	2	S_2	S_2
S_2	2	3	S_2	S_0

(segundo)

Figura 14.22. Representación tabular de un FSM

La transición de estado y los valores de salida para diferentes Se pueden representar entradas ent en diferentes estados usando una tabla de estado. La tabla de estado para el FSM en La figura 14.21 se muestra en la figura 14.22. Cada pareja contra un símbolo de entrada representa el nuevo estado y el símbolo de salida.

Por ejemplo, las figuras 14.22 (a) y 14.22 (b) son dos representaciones alternativas de la FSM en la figura ure 14.21.

8. gramáticas

[1 *, c13]

La gramática de un lenguaje natural nos dice si una combinación de palabras hace válida frase. A diferencia de los lenguajes naturales, un lenguaje formal el indicador se especifica mediante un conjunto bien definido de reglas para sintaxis Las oraciones válidas de un lenguaje formal. puede ser descrito por una gramática con la ayuda de estas reglas, denominadas *reglas de producción* .

Un lenguaje formal es un conjunto de longitud finita palabras o cadenas sobre un alfabeto finito, y una gramática especifica las reglas para la formación de Estas palabras o cadenas. Todo el conjunto de palabras que son válidos para una gramática constituye el lenguaje calibrador para la gramática. Por lo tanto, la gramática G es cualquier definición matemática compacta y precisa de un lenguaje L en lugar de solo una lista cruda de todos de las oraciones legales del lenguaje o ejemplos de esas oraciones

Una gramática implica un algoritmo que generar todas las oraciones legales del idioma. Existen diferentes tipos de gramáticas.

Una estructura de frase o gramática tipo 0 G = (V, T, S, P) es una tupla de 4 en la que:

- V es el vocabulario, es decir, conjunto de palabras.
- T ⊆ V es un conjunto de palabras llamadas terminales.
- S ∈ N es una palabra especial llamada inicio símbolo.
- P es el conjunto de reglas de producción para la sustitución ing fragmento de una oración para otro.

Existe otro conjunto N = V - T de palabras llamados no terminales. Los no terminales representan conceptos como *sustantivo.* Se aplican las reglas de producción. en cadenas que contienen no terminales hasta que no haya más los símbolos no terminales están presentes en la cadena. El símbolo de inicio S es un no terminal.

Página 272

El lenguaje generado por una gramática formal. G, denotado por L (G), es el conjunto de todas las cadenas sobre

3. Cada CSG es una gramática de estructura de frase (PSG)

el conjunto de alfabetos V que se pueden generar, inicio-
ing con el símbolo de inicio, aplicando produc-
Reglas hasta que todos los símbolos no terminales sean
reemplazado en la cadena.

Por ejemplo, sea G = ({S, A, a, b}, {a, b}, S, {S
→ aA, S → b, A → aa}). Aquí, el conjunto de termi-
nals son N = {S, A}, donde S es el símbolo de inicio.
Las tres reglas de producción para la gramática son
dado como P1: S → aA; P2: S → b; P3: A → aa.

Aplicando las reglas de producción en todo lo posible
formas, se pueden generar las siguientes palabras
desde el símbolo de inicio.

S → aA	(usando P1 en el símbolo de inicio)
→ aaa	(usando P3)
S → b	(usando P2 en el símbolo de inicio)

Nada más se puede derivar para G. Por lo tanto, el
el lenguaje de la gramática G consta de solo dos
palabras: L (G) = {aaa, b}.

8.1. Reconocimiento de idioma

Las gramáticas formales se pueden clasificar según el
tipos de producciones permitidas. El chom-
jerarquía del cielo (introducido por Noam Chomsky en
1956) describe tal esquema de clasificación.

Figura 14.23. Jerarquía Chomsky de gramáticas

Como se ilustra en la figura 14.23, inferimos lo siguiente:
bajando en diferentes tipos de gramáticas:

1. Cada gramática regular es un contexto libre
   gramática (CFG).
2. Cada CFG es una gramática sensible al contexto
   (CSG).

Gramática sensible al contexto: todos los fragmentos en
los RHS son más largos que los correspondientes
fragmentos en el LHS o vacíos, es decir, si b → a, entonces
| b | <| a | o a = ∅.

Un lenguaje formal es sensible al contexto si un
la gramática sensible al texto lo genera.

Gramática libre de contexto: todos los fragmentos en el
LHS son de longitud 1, es decir, si A → a, entonces | A | = 1
para todo A ∈ N.

El término libre de contexto deriva del hecho de que
A siempre se puede reemplazar por a, independientemente del
contexto en el que ocurre.

Un lenguaje formal no tiene contexto si un contexto
la gramática libre lo genera. Lan- sin contexto
los indicadores son la base teórica para la sintaxis de
La mayoría de los lenguajes de programación.

Gramática regular. Todos los fragmentos en el RHS
son terminales individuales o un par construido por un
terminal y no terminal; es decir, si A → a, entonces
ya sea a ∈ T, o a = cD, o a = Dc para c ∈ T, D ∈ N.

Si a = cD, entonces la gramática se llama derecha
gramática lineal Por otro lado, si a = Dc, entonces
la gramática se llama gramática lineal izquierda. Ambos
las gramáticas lineales derecha e izquierda son regulares
Lar o gramática tipo 3.

El lenguaje L (G) generado por un regular
la gramática G se llama lenguaje regular.

Una expresión regular A es una cadena (o patrón)
formado a partir de las siguientes seis piezas de información
mación: a ∈ S, el conjunto de alfabetos, e, 0 y el
operaciones, OR (+), PRODUCTO (.), CONCATE-
NACIÓN (*). El lenguaje de G, L (G) es igual a
todas esas cadenas que coinciden con G, L (G) = {x ∈ S * | x
coincide con G}.

Para cualquier a ∈ S, L (a) = a; L (e) = {ε}; L (0) = 0.
+ funciona como un o, L (A + B) = L (A) ∪ L (B).
. crea una estructura de producto, L (AB) = L (A).
L (B).
* denota concatenación, L (A *) = {$x_1 x_2 \ldots x_n$ |
$x_i \in L(A)$ y $n \geq 0$}

Por ejemplo, la expresión regular (ab) *
coincide con el conjunto de cadenas: {e, ab, abab, ababab,
abababab, ...}.

Página 273

Por ejemplo, la expresión regular (aa) *
coincide con el conjunto de cadenas en una letra *a* que tiene
longitud uniforme

Por ejemplo, la expresión regular (aaa) * +
(aaaaa) * coincide con el conjunto de cadenas de longitud igual
a un múltiplo de 3 o 5.

9. Precisión numérica, precisión y errores

[2 *, c2]

El objetivo principal del análisis numérico es
desarrollar algoritmos eficientes para computar pre
cise valores numéricos de funciones, soluciones de
ecuaciones algebraicas y diferenciales, optimización
problemas, etc.

Una cuestión de hecho es que todas las computadoras digitales por lo
interpretar o *cortar* (es decir, el representante exacto
inmediatamente debajo, o arriba, si es negativo, el
número).

Los números que se encuentran más allá del rango deben ser representados
sultado por el mayor (o mayor negativo) número
eso puede ser representado. Esto se convierte en un símbolo
por desbordamiento. El desbordamiento ocurre cuando un cálculo
ción produce un valor mayor que el máximo
valor en el rango.

Cuando la velocidad de procesamiento es una botella significativa
cuello, el uso de las representaciones de punto fijo
es una alternativa atractiva y más rápida a la más
engorrosa aritmética de coma flotante más compleja
Monly utilizado en la práctica.

Definamos un par de términos muy importantes:
precisión y *precisión* asociada con el número

solo almacene números finitos. En otras palabras, hay no hay forma de que una computadora pueda representar una número muy grande, ya sea un número entero, racional número, o cualquier número real o todos los números complejos (ver sección 10, teoría de los números). Entonces las matemáticas de aproximación se vuelve muy crítico de manejar todos los números en el rango finito que una computadora puede manejar.

A cada número en una computadora se le asigna una ubicación o palabra, que consiste en un número específico de dígitos binarios o bits. Una palabra de k bits puede almacenar un total de $N = 2^k$ números diferentes.

Por ejemplo, una computadora que usa aritios de 32 bits la métrica puede almacenar un total de $N = 2^{32} \approx 4.3 \times 10^9$ números diferentes, mientras que otro que usa 64 los bits pueden manejar $N' = 2^{64} \approx 1.84 \times 10^{19}$ diferentes números. La pregunta es cómo distribuir estos N números sobre la línea real para una eficiencia máxima Ciencia y precisión en cálculos prácticos.

Una opción evidente es distribuirlos de manera uniforme, que conduce a la aritmética de punto fijo. En este sistema el primer bit de una palabra se usa para representar un signo y los bits restantes se tratan para el valor entero ues. Esto permite la representación de los enteros. de 1 - ½N, es decir, = 1 - 2^{k-1} a 1. Como aproximadamente método de apareamiento, esto no es bueno para los no enteros números.

Otra opción es espaciar los números de cerca juntos, digamos con un espacio uniforme de 2^{-n}, y así distribuir el total de N números uniformemente sobre el intervalo $-2^{n-1} N < x \leq 2^{n-1} N$. Números reales mentirosos entre los huecos están representados por cualquier *redondeo* (es decir, el representante exacto más cercano) Análisis ical.

La precisión es la cercanía con la que un medio El valor demandado o calculado coincide con el valor verdadero. Esa precisión, por otro lado, es la cercanía. son el que dos o más medidos o calculados valores para la misma sustancia física de acuerdo con El uno al otro. En otras palabras, la precisión es el cierre ness con que un número representa un exacto valor.

Sea x un número real y sea x * un aproximado. El *error absoluto* en la aproximación. $x^* \approx x$ se define como $| x^* - x |$. El *error relativo* se define como la relación del error absoluto al dimensión de x, es decir, $| x^* - x | / | x |$, que supone $x \neq 0$; de lo contrario, el error relativo no está definido.

Por ejemplo, 1000000 es una aproximación a 1000001 con un error absoluto de 1 y un relativo error de 10^{-6}, mientras que 10 es una aproximación de 11 con un error absoluto de 1 y un error relativo de 0.1. Típicamente, el error relativo es más intuitivo y El determinante preferido del tamaño del error. La presente convención es que los errores son siempre ≥ 0, y son = 0 si y solo si la aproximación es exacto

Una aproximación x * tiene k deci significativo dígitos mal si su error relativo es $< 5 \times 10^{-k-1}$. Esta significa que los primeros k dígitos de x * después de su El primer dígito distinto de cero es el mismo que el de x.

Los dígitos significativos son los dígitos de un número que se sabe que son correctos En una medida, uno Se incluye un dígito incierto.

Por ejemplo, medición de longitud con una regla de 15.5 mm con ± 0.5 mm máximo error permitido tiene 2 dígitos significativos, mientras que una medida de la misma longitud usando un calibrador y registrado como 15.47 mm con ± 0.01 mm máx. El error permitido de mamá tiene 3 dígitos significativos.

Página 274

10. Teoría de números

[1 *, c4]

La teoría de números es una de las ramas más antiguas. de matemática pura y una de las más grandes. De Por supuesto, se trata de preguntas sobre números, generalmente significa números enteros y fraccionales o numeros racionales. Los diferentes tipos de números incluye entero, número real, número natural, número complejo, número racional, etc.

10.1 Divisibilidad

Comencemos esta sección con una breve descripción de cada uno de los tipos de números anteriores, comenzando con Los números naturales.

Números naturales. Este grupo de números comienza en 1 y continúa: 1, 2, 3, 4, 5, y así sucesivamente. Cero No está en este grupo. No hay negativos o fracnúmeros nacionales en el grupo de números naturales. El símbolo matemático común para el conjunto de todos los números naturales son N.

Números enteros Este grupo tiene todos los natunúmeros ral en él más el número 0.

Desafortunadamente, no todos aceptan lo anterior los decimales no existen, por ejemplo, 15 o cuando los decimales existen, pueden terminar, como en 15.6, o pueden repetir con un patrón, como en 1.666 ..., (que es 5/3).

Numeros irracionales. Estos son números que no se puede expresar como un entero dividido por un entero. Estos números tienen decimales que nunca terminar y nunca repetir con un patrón, por ejemplo, PI o √2.

Numeros reales. Este grupo está formado por todos los números racionales e irracionales. Los números que se encuentran al estudiar álgebra son reales números. El símbolo matemático común para el conjunto de todos los números reales es R.

Números imaginarios Todos estos se basan en el número imaginario i. Este número imaginario es igual a la raíz cuadrada de −1. Cualquier número real múltiplo de i es un número imaginario, por ejemplo, i, $5i$, $3.2i$, $-2.6i$, etc.

Números complejos. Un número complejo es un combinación de un número real y un imaginario número en la forma $a + bi$. La parte real es una, y b se llama la parte imaginaria. Las matemáticas comunes símbolo matemático para el conjunto de todos los números complejos Bers es **C.**

Por ejemplo, $2 + 3i$, $3-5i$, $7.3 + 0i$, y $0 + 5i$.

Considere los dos últimos ejemplos:

$7.3 + 0i$ es lo mismo que el número real 7.3.

Por lo tanto, todos los números reales son números complejos con cero para la parte imaginaria.

definiciones de números naturales y enteros. Ahí parece no haber acuerdo general sobre si para incluir 0 en el conjunto de números naturales.

Muchos matemáticos consideran que, en Europa, la secuencia de números naturales tradicionalmente comenzó con 1 (0 ni siquiera se consideraba un número de los griegos). En el siglo 19, establecer teóricos y otros matemáticos comenzaron la convención de incluir 0 en el conjunto de naturales números.

Enteros Este grupo tiene todos los números enteros. en ella y sus negativos. La matemática común El símbolo cal para el conjunto de todos los enteros es Z, es decir Z = {..., −3, −2, −1, 0, 1, 2, 3, ...}.

Numeros racionales. Estos son cualquier número que puede expresarse como una razón de dos enteros. los símbolo común para el conjunto de todos los números racionales bers es Q.

Los números racionales pueden clasificarse en tres tipos, en función de cómo actúan los decimales. los

Del mismo modo, 0 + 5 *i* es solo el número imaginario 5 *i* . Por lo tanto, todos los números imaginarios son complejos. números con cero para la parte real.

La teoría elemental de números implica divisibilidad entre enteros. Sea a, b ∈ Z con a ≠ 0. La expresión sión a | b, es decir, *a divide b* si ∃c ∈ Z: b = ac, es decir, hay es un número entero c tal que c veces a es igual a b.

Por ejemplo, 3 | −12 es verdadero, pero 3 | 7 es falso.

Si *a* divide *b* , entonces decimos que *a* es un factor de *b* o *a* es un divisor de *b* , y *b* es un múltiplo de *a* .

b es incluso si y solo si 2 | *b* .

Deje *a, d* ∈ Z con d> 1. Entonces *un mod d* denota el Zesto *r* del algoritmo de división con dividendo *una* y divisor *d* , es decir, el resto cuando a se divide por d. Podemos calcular *(un mod d)* por: *a - d* * ⌊ *a / d* ⌋, donde ⌊ *a / d* ⌋ representa el piso del número real.

Deje $Z_+ = \{n \in Z \mid n > 0\}$ y *a, b* ∈ Z, m ∈ Z_+, entonces a es congruente con *b módulo m* , escrito como *a* ≡ *b (mod m)* , si y solo si *m* | *a* − *b* .

Página 275

Alternativamente, *a* es congruente con *b módulo m* si y solo si *(a − b) mod m = 0* .

10.2 Número primo, MCD

Un número entero p> 1 es primo si y solo si no lo es el producto de dos enteros mayores que 1, es decir, p es primo si p> 1 ∧ ∃ ¬ a, b ∈ N: a> 1, b> 1, a * b = p.

Los únicos factores positivos de un primer p son 1 y p mismo. Por ejemplo, los números 2, 13, 29, 61, etc. son números primos. Enteros no primos mayores que 1 se llaman números compuestos. UNA el número compuesto puede estar compuesto por múltiples manejando dos enteros mayores que 1.

Hay muchas aplicaciones interesantes de números primos; entre ellos están el público esquema de criptografía clave, que implica el intercambio de claves públicas que contienen el producto *p* * *q* de dos primos grandes aleatorios *p* y *q* (un privado clave) que una parte determinada debe mantener en secreto.

El máximo común divisor mcd (a, b) de integers a, b es el mayor número entero d que es un divisor tanto de a como de b, es decir,

d = mcd (a, b) para max (d: d | a ∧ d | b)

Por ejemplo, mcd (24, 36) = 12.

Enteros *una* y *b* se llaman primos entre sí o coprime si y solo si su MCD es 1.

Por ejemplo, ni 35 ni 6 son primos, pero son coprimos ya que estos dos números no tienen factores comunes mayores que 1, por lo que su MCD es 1.

Un conjunto de enteros $X = \{i_1, i_2, ...\}$ es relativamente prima si todos los pares posibles i_h, i_k, h ≠ k extraídos de El conjunto X son relativamente primos.

11. Estructuras algebraicas

Esta sección presenta algunas representaciones utilizado en álgebra superior. Una estructura algebraica consiste en uno o dos conjuntos cerrados bajo algunos operaciones y satisfacer varios axiomas, incluyendo ninguno

11.1 Grupo

Un conjunto S cerrado bajo una operación binaria • forma un grupo si la operación binaria satisface el siguiente ing cuatro criterios:

- Asociativo: ∀a, b, c ∈ S, la ecuación (a • b) • c = a • (b • c) se mantiene.
- Identidad: existe un elemento de identidad I ∈ S tal que para todo a ∈ S, I • a = a • I = a.
- Inversa: cada elemento a ∈ S, tiene una inversa a '∈ S con respecto a la operación binaria, es decir, a • a '= I; por ejemplo, el conjunto de enteros Z con respecto a la operación de suma es un grupo. El elemento de identidad del conjunto es 0 para La operación de suma. ∀x ∈ Z, el inverso de x sería –x, que también se incluye en Z.
- Propiedad de cierre: ∀a, b ∈ S, el resultado de la operación a • b ∈ S.
- Un grupo que es conmutativo, es decir, a • b = b • a, es conocido como un grupo conmutativo o abeliano .

El conjunto de números naturales N (con la operano es un grupo, ya que no hay inversa para cualquier x> 0 en el conjunto de números naturales. Por lo tanto, la tercera regla (de inversa) para nuestra operación es violado Sin embargo, el conjunto de números naturales Tiene alguna estructura.

Conjuntos con una operación asociativa (la primera condición anterior) se denominan semigrupos; si ellos también tienen un elemento de identidad (la segunda condición ción), entonces se llaman monoides.

Nuestro conjunto de números naturales bajo suma es entonces un ejemplo de un monoide, una estructura que no es del todo un grupo porque le falta el requisito de que cada elemento tenga un inverso bajo la operacion.

Un monoide es un conjunto S que está cerrado bajo un solo operación binaria asociativa • y tiene una identidad elemento I ∈ S tal que para todo a ∈ S, I • a = a • I = a. Un monoide debe contener al menos un elemento.

Por ejemplo, el conjunto de números naturales N forma un monoide conmutativo bajo adición con elemento de identidad 0. El mismo conjunto de números naturales

Por ejemplo, grupo, monoide, anillo y red. son ejemplos de estructuras algebraicas. Cada uno de estos se definen en esta sección.

monoide también se puede considerar como un semigrou p con un elemento de identidad .

Un *subgrupo* es un grupo *H* contenido dentro de un más grande, *G, de* modo que el elemento de identidad de *G* está contenido en *H* , y siempre que h_1 y h_2 son en *H* , entonces también lo son $h_1 \bullet h_2$ y así, el ele- Ments de *H* , equipado con la operación de grupo en *G* restringido a *H* , de hecho forman un grupo.

Dado cualquier subconjunto *S* de un grupo *G* , el subgrupo generado por *S* consiste en productos de elementos de *S* y sus inversas. Es el subgrupo más pequeño. de *G* que contiene *S* .

Por ejemplo, que *G* sea el grupo abeliano cuyo los elementos son *G* = {0, 2, 4, 6, 1, 3, 5, 7} y cuyo la operación grupal es el módulo de suma 8. Este grupo tiene un par de subgrupos no triviales: *J* = {0, 4} y *H* = {0, 2, 4, 6}, donde *J* es también un subgrupo de *H* .

En la teoría de grupos, un grupo cíclico es un grupo que puede ser generado por un solo elemento, en el sentir que el grupo tiene un elemento *a* (llamado *generador* del grupo) tal que, cuando está escrito multiplicativamente, cada elemento del grupo es un poder de *a* .

Un grupo G es cíclico si G = $\{a^n\}$ para cualquier número entero *n*.

Dado que cualquier grupo generado por un elemento en un grupo es un subgrupo de ese grupo, que muestra que El único subgrupo de un grupo G que contiene *a* es G en sí es suficiente para mostrar que G es cíclico.

Por ejemplo, el grupo *G* = {0, 2, 4, 6, 1, 3, 5, 7}, con respecto a la operación de módulo de adición 8, es cíclico Los subgrupos *J* = {0, 4} y *H* = {0, 2, 4, 6} también son cíclicos.

Bers N también forma un monoide bajo multiplicación con elemento de identidad 1. El conjunto de interacciones positivas gers P forma un monoide conmutativo bajo múltiples plication con el elemento de identidad 1.

Cabe señalar que, a diferencia de los de un grupo , Los elementos de un monoide no necesitan tener inversos. UNA

Página 276

11.2 Anillos

Si tomamos un grupo abeliano y definimos un segundo operación en él, se encuentra una nueva estructura que es diferente de solo un grupo. Si esta segunda operación es asociativa y es distributiva sobre el primero, luego tenemos un anillo.

Un anillo es un triple de la forma (S, +, •), donde (S, +) es un grupo abeliano, (S, •) es un semigrupo y • es distributivo sobre +; es decir, "a, b, c ∈ S, la ecuación $a \bullet (b + c) = (a \bullet b) + (a \bullet c)$ se mantiene. Además, si • es conmutativo, entonces se dice que el anillo es conmutativo mutante Si hay un elemento de identidad para el • operación, entonces se dice que el anillo tiene una identidad.

Por ejemplo, (Z, +, *), es decir, el conjunto de enteros Z, con la operación habitual de suma y multiplicación es un anillo Como (Z, *) es conmutativo, este anillo es un anillo conmutativo o abeliano. El anillo tiene 1 como su elemento de identidad.

Tengamos en cuenta que la segunda operación puede no tener un elemento de identidad, ni necesitamos encontrar un inverso para cada elemento con respecto a esto segunda operación En cuanto a lo que significa distributivo, intuitivamente es lo que hacemos en matemáticas elementales matics al realizar el siguiente cambio: a * (b + c) = (a * b) + (a * c).

Un campo es un anillo para el cual los elementos del conjunto, excluyendo 0, forma un grupo abeliano con el segunda operación

Un ejemplo simple de un campo es el campo de relación números finales (R, +, *) con la suma habitual y operaciones de multiplicación. Los números de el formato *a* / *b* ∈ R, donde *a, b* son enteros y $b \neq 0$. El inverso aditivo de tal fracción es simplemente - *a* / *b* , y el inverso multiplicativo es *b* / *a* siempre que $a \neq 0$.

Página 277

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[1 *] osen 2011 R	07 caid 20 en d K[2 *] ey an en Ch
1. Conjuntos, relaciones, funciones	c2	
2. Lógica básica	c1	
3. Técnicas de prueba	c1	
4. Conteo básico	c6	
5. Gráficos y árboles	c10, c11	
6. Probabilidad discreta	c7	
7. Máquinas de estado finito	c13	
8. gramáticas	c13	
9. Precisión numérica, precisión y errores		c2
10. Teoría de números	c4	
11. Estructuras algebraicas		

Referencias

[1 *] K. Rosen, *Matemática discreta y sus Aplicaciones* , 7ma ed., McGraw-Hill, 2011.

[2 *] EW Cheney y DR Kincaid, *numéricos Matemáticas e Informática* , 6ª ed., Brooks / Cole, 2007.

EXPRESIONES DE GRATITUD

El autor agradece agradecidamente la contribución Sesión del Prof. Arun Kumar Chatterjee, Ex-Jefe, Departamento de Matemáticas, Universidad de Manipur sity, India, y la profesora Devadatta Sinha, ex-directora, Departamento de Informática e Ingeniero ing, Universidad de Calcuta, India, en preparación este capítulo sobre Fundamentos matemáticos.

CAPITULO 15

FUNDACIONES DE INGENIERIA

SIGLAS

CANALLA	Diseño asistido por ordenador
CMMI	Modelo de Capacidad de Madurez Integración
pdf	Función de densidad de probabilidad
pmf	Función de probabilidad
RCA	Análisis de causa raíz
SDLC	Ciclo de vida del desarrollo de programas

INTRODUCCIÓN

IEEE define la ingeniería como "la aplicación de un enfoque sistemático, disciplinado y cuantificable a estructuras, máquinas, productos, sistemas o efectivamente es un objetivo de todos los ingenieros en todos los motores disciplinas de neering.

DESGLOSE DE TEMAS PARA FUNDACIONES DE INGENIERIA

El desglose de temas para la Ingeniería Fundamentos KA se muestra en la Figura 15.1.

1. Métodos empíricos y experimentales. Tecnicas

[2 *, c1]

Un método de ingeniería para resolver problemas. implica proponer soluciones o modelos de soluciones y luego realizar experimentos o pruebas para estudiar las soluciones o modelos propuestos. Así,

procesos."[1]. Este capítulo describe algunos de los habilidades y técnicas fundamentales de ingeniería que son útiles para un ingeniero de software. El foco trata sobre temas que admiten otros KA mientras que los mi- mizing duplicación de temas cubiertos en otros lugares en este documento

Como la teoría y la práctica del software de ingeniería la orientación madura, cada vez es más evidente que la ingeniería de software es una disciplina de ingeniería plan que se basa en el conocimiento y las habilidades Mon a todas las disciplinas de ingeniería. Este motor el área de conocimiento de fundaciones neering (KA) es preocupado por los fundamentos de ingeniería que aplicar a la ingeniería de software y otros motores disciplinas de neering. Los temas en este KA incluyen métodos empíricos y técnicas experimentales; análisis estadístico; medición; Ingenieria diseño; modelado, creación de prototipos y simulación; estándares; y análisis de causa raíz. Solicitud de este conocimiento, según corresponda, permitirá ingenieros de software para desarrollar y mantener software de manera más eficiente y efectiva. Com- completar su trabajo de ingeniería de manera eficiente y los ingenieros deben entender cómo crear una experiencia mire y luego analice los resultados de la experiencia para evaluar la solución propuesta.

Métodos empíricos y técnicas experimentales. ayudar al ingeniero a describir y comprender habilidad en sus observaciones, para identificar las fuentes de variabilidad y para tomar decisiones.

Tres tipos diferentes de estudios empíricos componen monly utilizado en esfuerzos de ingeniería están diseñados experimentos, estudios observacionales y retro Estudios espectivos. Breves descripciones de la composición. Los métodos utilizados a continuación se dan a continuación.

1.1. Experimento diseñado

Un experimento diseñado o controlado es una inversión. Tigración de una hipótesis comprobable donde uno o más variables independientes son manipuladas para medir su efecto en uno o más dependientes variables Una condición previa para llevar a cabo un El experimento es la existencia de una hipótesis clara. Es importante que un ingeniero entienda cómo formular hipótesis claras.

Figura 15.1. Desglose de temas para los fundamentos de ingeniería KA

Los experimentos diseñados permiten a los ingenieros determinar en términos precisos cómo son las variables relacionado y, específicamente, si una causa-efecto La relación existe entre ellos. Cada combinación nación de valores de las variables independientes es un *tratamiento* . Los experimentos más simples tienen solo dos tratamientos que representan dos niveles de pecado gle variable independiente (por ejemplo, usando una herramienta vs no usando una herramienta). Experimental más complejo los diseños surgen cuando más de dos niveles, más de una variable independiente, o cualquier dependiente Se utilizan variables.

1.2. Estudio observacional

Un estudio observacional o de caso es empírico. consulta que hace observaciones de procesos o fenómenos dentro de un contexto de la vida real. Mientras un experimento ignora deliberadamente el contexto, un estudio observacional o de casos incluye contexto como parte de la observación. Un estudio de caso es más útil completo cuando el foco del estudio es *cómo* y *por qué* preguntas, cuando el comportamiento de los involucrados en el estudio no puede ser manipulado, y cuando las condiciones textuales son relevantes y los límites entre los fenómenos y el contexto no están claros.

1.3. Estudio retrospectivo

Un estudio retrospectivo implica el análisis de his- datos tóricos También se conocen estudios retrospectivos.

2. Análisis estadístico

[2 *, c9s1, c2s1] [3 *, c10s3]

Para cumplir con sus responsabilidades, neers deben entender cómo diferente producto y las características del proceso varían. Ingenieros a menudo encontrar situaciones donde la relación entre diferentes variables necesita ser estudiado. Un punto importante a tener en cuenta es que la mayoría de los los estudios se realizan en base a muestras y entonces los resultados observados necesitan ser entendidos con respecto a la población total. Ingenieros debe, por lo tanto, desarrollar una comprensión adecuada ing de técnicas estadísticas para recolectar datos confiables datos en términos de muestreo y análisis para llegar a resultados que pueden generalizarse. Estas tecnicas se discuten a continuación.

2.1. Unidad de análisis (unidades de muestreo), Población y muestra

Unidad de Análisis. Mientras realiza cualquier empiri- estudio cal, las observaciones deben hacerse en cho- unidades sen llamadas unidades de análisis o muestreo unidades. La unidad de análisis debe ser identificada y debe ser apropiado para el análisis. Para examen- ple, cuando una empresa de productos de software quiere encontrar la usabilidad percibida de un producto de software, el usuario o la función del software puede ser la unidad de análisis

Población. El conjunto de todos los encuestados o ítems

como estudios historicos. Este tipo de estudio utiliza datos. (con respecto a algún fenómeno) que ha sido archivado en el tiempo. Estos datos archivados son entonces analizados Lyzed en un intento de encontrar una relación entre variables, para predecir eventos futuros o para identificar tendencias. La calidad de los resultados del análisis dependerá de la calidad de la información contenida en los datos archivados Los datos históricos pueden ser incompletos. Completo, inconsistentemente medido o incorrecto.

(posibles unidades de muestreo) a estudiar forma el población . Como ejemplo, considere el caso de estudiar la usabilidad percibida de un software producto. En este caso, el conjunto de todos los usuarios posibles forma la población

Al definir la población, se debe tener cuidado ejercido para comprender el estudio y el objetivo no población. Hay casos en que la población estudio estudiado y la población para la cual

Page 281

Los resultados que se generalizan pueden ser diferentes. Por ejemplo, cuando la población de estudio consiste de solo observaciones pasadas y generalizaciones son requerido para el futuro, la población de estudio y la población objetivo puede no ser la misma.

Muestra. Una muestra es un subconjunto de la población. La cuestión más crucial para la selección de una muestra es su representatividad, incluido el tamaño. Las muestras deben tomarse de manera tal que para garantizar que los sorteos sean independientes, y las reglas para dibujar las muestras deben ser previas definido de modo que la probabilidad de seleccionar un par- La unidad de muestreo ticular se conoce de antemano. Esta método de selección de muestras se llama *probabilidad muestreo.*

Variable aleatoria. En terminología estadística, el proceso de hacer observaciones o medir Ment sobre las unidades de muestreo que se estudian es referido como conducir el experimento. por ejemplo, si el experimento es lanzar una moneda 10 veces y luego contar el número de veces que la moneda cae en las cabezas, cada 10 lanzamientos de la moneda es una unidad de muestreo y el número de cabezas para un muestra dada es la observación o el resultado de el experimento. El resultado de un experimento es obtenido en términos de números reales y define el variable aleatoria en estudio. Por lo tanto, el atributo de los ítems que se miden en el resultado de el experimento representa la variable aleatoria siendo estudiado; la observación obtenida de un unidad de muestreo particular es una realización particular de la variable aleatoria. En el ejemplo de la moneda lanzamiento, la variable aleatoria es el número de cabezas observado para cada experimento. En estudios estadísticos ies, se hacen intentos para comprender la población características sobre la base de muestras.

El conjunto de valores posibles de una variable aleatoria. puede ser finito o infinito pero contable (por ejemplo, el conjunto de todos los enteros o el conjunto de todos los números impares). En tal caso, la variable aleatoria se llama un *dis-Creta variable aleatoria.* En otros casos, el azar variable bajo consideración puede tomar valores en una escala continua y se llama un *rango continuo Dom variable.*

Evento. Un subconjunto de posibles valores de un azar La variable se llama evento . Supongamos que X denota alguna variable aleatoria; entonces, por ejemplo, nosotros puede definir diferentes eventos como X ³ x o X < X y así sucesivamente.

Distribución de una variable aleatoria. El rango y el patrón de variación de una variable aleatoria es dada por su distribución. Cuando la distribución de una variable aleatoria es conocida, es posible Calcule la posibilidad de cualquier evento. Algunos distribu- Se encuentra que las acciones ocurren comúnmente y se usan para modelar muchas variables aleatorias que ocurren en práctica en el contexto de la ingeniería. Un poco de las distribuciones más comunes son dada a continuación.

- Distribución binomial: se utiliza para modelar al azar. variables que cuentan el número de éxitos en *n* ensayos realizados independientemente de cada otro, donde cada prueba resulta exitosa o fracaso. Suponemos que el la posibilidad de obtener un éxito sigue siendo stant [2 *, c3s6].
- Distribución de Poisson: se utiliza para modelar el recuento. de ocurrencia de algún evento en el tiempo o espacio [2 *, c3s9].
- Distribución normal: se utiliza para modelar la continuidad. ous variables aleatorias o discretas al azar variables tomando una gran cantidad de valores [2 *, c4s6].

Concepto de parámetros. Una distribución estadística Se caracteriza por algunos parámetros. Para examen- ple, la proporción de éxito en cualquier ensayo dado es el único parámetro que caracteriza a un binomio distribución. Del mismo modo, la distribución de Poisson es caracterizado por una tasa de ocurrencia. Un normal La distribución se caracteriza por dos parámetros: a saber, su desviación media y estándar.

Una vez que se conocen los valores de los parámetros, la distribución de la variable aleatoria es com completamente conocido y la posibilidad (probabilidad) de Cualquier evento puede ser calculado. Las probabilidades para una variable aleatoria discreta se puede calcular a través de la función de masa de probabilidad, llamada el pmf. El PMF se define en puntos discretos. y da el punto de masa, es decir, la probabilidad que la variable aleatoria tomará ese particular valor. Asimismo, para una variedad aleatoria continua capaz, tenemos la función de densidad de probabilidad, llamado el pdf. El pdf es muy parecido a la densidad y necesita integrarse en un rango para obtener la probabilidad de que la variable aleatoria continua mentiras capaces entre ciertos valores. Por lo tanto, si el pdf

Página 282

o pmf es conocido, las posibilidades de la variación aleatoria capaz de tomar cierto conjunto de valores puede ser calculado teóricamente

Concepto de estimación [2 *, c6s2, c7s1, c7s3]. Los valores verdaderos de los parámetros de una distribución. son generalmente desconocidos y necesitan ser estimados de las observaciones de muestra. Las estimaciones son funciones de los valores de muestra y se denominan statistica. Por ejemplo, la media muestral es una estadística y puede usarse para estimar la media de la población. Del mismo modo, la tasa de aparición de defectos es apareado de la muestra (tasa de defectos por línea de código) es una estadística y sirve como la estimación de la tasa poblacional de la tasa de defectos por línea de código. La estadística utilizada para estimar algunas poblaciones El parámetro ción a menudo se denomina *estimador* del parámetro

Un punto muy importante a tener en cuenta es que los resultados de los estimadores mismos son aleatorios. Si nosotros tomar una muestra diferente, es probable que obtengamos una diferente Estimación actual del parámetro de población. En el teoría de la estimación, necesitamos entender dif propiedades diferentes de los estimadores, en particular, cómo tanto las estimaciones pueden variar entre muestras y Cómo elegir entre diferentes formas alternativas para obtener las estimaciones. Por ejemplo, si deseamos para estimar la media de una población, podríamos usar como nuestro estimador una media muestral, una muestra mediana, un modo de muestra o el rango medio de la muestra. Cada uno de estos estimadores tiene diferentes propiedades estadísticas que pueden afectar el estándar error de la estimación

Tipos de estimaciones [2 *, c7s3, c8s1]. Hay dos tipos de estimaciones: a saber, estimaciones puntuales y estimaciones de intervalo. Cuando usamos el valor de una estadística para estimar un parámetro de población, Obtenemos una estimación puntual. Como su nombre indica, la estimación puntual da un valor puntual del parámetro después de ser estimado.

Aunque a menudo se utilizan estimaciones puntuales, Deje espacio para muchas preguntas. Por ejemplo, nosotros no se les dice nada sobre el posible tamaño de error o propiedades estadísticas del punto esticompañero. Por lo tanto, podríamos necesitar complementar un punto estimar con el tamaño de la muestra, así como la variAnce de la estimación. Alternativamente, podríamos usar Una estimación de intervalo. Una estimación de intervalo es un intervalo aleatorio con el límite inferior y superior es del intervalo siendo funciones de la muestra observaciones, así como el tamaño de la muestra. El limse calculan sobre la base de algunos supuestos iones relativas a la distribución muestral de la estimación puntual en la que se basan los límites.

Propiedades de los estimadores. Varios estadísticos las propiedades de los estimadores se utilizan para decidir sobre la idoneidad de un estimador en un determinado situación. Las propiedades más importantes son que un estimador es imparcial, eficiente y consistente con respecto a la población.

Pruebas de hipótesis [2 *, c9s1]. Una hipótesis es una declaración sobre los posibles valores de un parámetro eter. Por ejemplo, supongamos que se afirma que un nuevo método de desarrollo de software reduce el ocurrencia de defectos. En este caso, la hipótesis es decir, la tasa de aparición de defectos tiene reducido. En las pruebas de hipótesis, decidimos sobre la base de las observaciones de muestra, ya sea un proLa hipótesis planteada debe ser aceptada o rechazada. Para probar hipótesis, la nula y la alternativa Se forman hipótesis. La hipótesis nula es la hipótesis de no cambio y se denota como H_0. los La hipótesis alternativa se escribe como H_1. Es importante Tant para notar que la hipótesis alternativa puede ser de un lado o de dos lados. Por ejemplo, si tenemos la hipótesis nula de que la media de la población no es menos de un valor dado, la hipótesis alternativa Esis sería que es menor que ese valor y nosotros tendría una prueba unilateral. Sin embargo, si tenemos la hipótesis nula de que la media de la población es igual a algún valor dado, la hipótesis alternativa Esis sería que no es igual y lo haríamos tener una prueba de dos lados (porque el valor verdadero podría ser menor o mayor que el valor dado).

Para probar alguna hipótesis, primero comparamos pute alguna estadística. Junto con el cómputo de la estadística, una región se define de tal manera que en caso el valor calculado de la estadística cae en esa región, la hipótesis nula es rechazada. Esta región se llama región crítica (también conocida como El intervalo de confianza). En pruebas de hipótesis, necesitamos aceptar o rechazar la hipótesis nula sobre la base de la evidencia obtenida. Nosotros notamos que en general, la hipótesis alternativa es la hipótesis de interés Si el valor calculado de la estadística no cae dentro de la región crítica, entonces no podemos rechazar la hipótesis nula. Esta indica que no hay suficiente evidencia para Creemos que la hipótesis alternativa es cierta.

Page 283

Como la decisión se toma sobre la base de observaciones de muestra, son posibles errores; la tipos de tales errores se resumen en el siguiente Mesa baja.

Naturaleza	Decisión estadística	
	Aceptar H_0	Rechazar H_0
H_0 es		Error tipo I

dado el valor de una variable, la otra puede ser estimado sin error. Una correlación positiva coeficiente indica una relación positiva, que es decir, si una variable aumenta, también lo hace la otra. En Por otro lado, cuando las variables son negativamente correlacionado, un aumento de uno conduce a una disminución del otro.

Es importante recordar esa correlación.

cierto	Okay	(probabilidad = a)
H_0 es falso	Error tipo II (probabilidad = b)	Okay

En la prueba de hipótesis, nuestro objetivo es maximizar el potencia de la prueba (el valor de 1 − b) mientras se asegura ing que la probabilidad de un error tipo I (el valor de a) se mantiene dentro de un valor particular: típicamente 5 por ciento.

Cabe señalar que la construcción de una prueba de La hipótesis incluye la identificación de estadística (s) para estimar los parámetros y definir una crítica región tal que si el valor calculado de la statistic cae en la región crítica, la hipótesis nula esis es rechazado.

2.2. Conceptos de correlación y regresión

[2 *, c11s2, c11s8]

Un objetivo principal de muchas investigaciones estadísticas es establecer relaciones que lo hagan posible es posible predecir una o más variables en términos de otros. Aunque es deseable predecir una cantidad tity exactamente en términos de otra cantidad, es self dom posible y, en muchos casos, tenemos que ser satisfecho con la estimación del promedio o esperado valores.

La relación entre dos variables es studied usando los métodos de correlación y regresión Sion. Ambos conceptos se explican brevemente en Los siguientes párrafos.

Correlación. La fuerza de la relación lineal. la nave entre dos variables se mide usando El coeficiente de correlación . Mientras computa el coeficiente de correlación entre dos variables, nosotros supongamos que estas variables miden dos diferencias ent atributos de la misma entidad. La correlación coeficiente toma un valor entre –1 a +1. los los valores –1 y +1 indican una situación cuando el la asociación entre las variables es perfecta, es decir, No implica causalidad. Por lo tanto, si dos variables están correlacionados, no podemos concluir que uno causa el otro.

Regresión. El análisis de correlación solamente mide el grado de relación entre dos variables El análisis para encontrar la relación la nave entre dos variables se llama *regresión análisis.* La fuerza de la relación entre dos variables se miden usando el coeficiente de determinación. Este es un valor entre 0 y 1. Cuanto más cercano sea el coeficiente a 1, más fuerte será relación entre las variables. Un valor de 1 Indica una relación perfecta.

3. Medida

[4 *, c3s1, c3s2] [5 *, c4s4] [6 *, c7s5] [7 *, p442–447]

Saber qué medir y qué medida El método de uso es crítico en ingeniería esfuerzos. Es importante que todos los involucrados en un proyecto de ingeniería entiendo el medio métodos de aseguramiento y resultados de medición eso será usado.

Las medidas pueden ser físicas, medioambientales. Tal, económico, operacional o algún otro tipo de medida que es significativa para el particular proyecto. Esta sección explora la teoría de la medición. aseguramiento y cómo es fundamental diseñar En g. La medición comienza como una conceptualización. luego pasa de conceptos abstractos a definiciones del método de medición a la aplicación real cación de ese método para obtener una medición resultado. Cada uno de estos pasos debe ser entendido, comunicado y empleado adecuadamente para para generar datos utilizables. En ingeniero tradicional ing, a menudo se usan medidas directas. En software ingeniería, una combinación de directa y las medidas derivadas son necesarias [6 *, p273].

La teoría de la medición establece que la garantía es un intento de describir un subyacente

Sistema empírico real. Métodos de medición definir actividades que asignan un valor o un símbolo a un atributo de una entidad.

Los atributos se deben definir en términos de Las operaciones utilizadas para identificar y medir ellos, es decir, los métodos de medición. En esto enfoque, un método de medición se define para ser una operación especificada con precisión que produce un número ber (llamado el *resultado de* la *medición)* cuando mea- representando un atributo. Se sigue que, para ser útil, El método de medición debe estar bien definido. La arbitrariedad en el método se reflejará en ambigüedad en los resultados de la medición.

En algunos casos, particularmente en lo físico mundo: los atributos que deseamos medir son fácil de entender sin embargo, en un mundo artificial como ingeniería de software, definiendo los atributos puede No sea tan simple. Por ejemplo, los atributos de altura, peso, distancia, etc. son fáciles y uni bien entendido (aunque pueden no ser muy fácil de medir en todas las circunstancias), mientras que atributos como el tamaño o la complejidad del software Esta simple medición conducirá a una sustancial variación. Los ingenieros deben apreciar la necesidad de Definir medidas desde una perspectiva operativa.

3.1. Niveles (escalas) de medida

[4 *, c3s2] [6 *, c7s5]

Una vez que se determinan las definiciones operativas, Las medidas reales deben llevarse a cabo. Cabe señalar que la medición puede ser caren cuatro escalas diferentes: a saber, nominal, ordinal, intervalo y razón. Breves descripciones de cada uno se da a continuación.

Escala nominal: este es el nivel más bajo de medición aseguramiento y representa el más sin restricciones asignación de números. Los números solo sirven como etiquetas, y las palabras o letras también servirían. La escala nominal de medición involucra solo clasificación y las unidades de muestreo observadas se ponen en cualquiera de los mutuamente excluyentes y categorías colectivamente exhaustivas (clases). Algunos ejemplos de escalas nominales son:

requieren definiciones claras.

Definiciones operacionales. La definición de atributo butes, para empezar, a menudo es bastante abstracto. Tal Las definiciones no facilitan las mediciones. por ejemplo, podemos definir un círculo como *una línea que forma un circuito cerrado tal que la distancia entre cualquier punto en esta línea y un punto interior fijo llamado El centro es constante.* Podemos decir además que el distancia fija desde el centro a cualquier punto del bucle cerrado da el radio del círculo. Puede ser señaló que aunque el concepto ha sido definido, no Se han propuesto medios para medir el radio. La definición operativa especifica los pasos exactos. o método utilizado para llevar a cabo una medida específica ment. Esto también se puede llamar la *medición método* ; a veces un *procedimiento de medición* puede ser requerido para ser aún más preciso.

La importancia de las definiciones operacionales Difícilmente se puede exagerar. Tome el caso de la medición aparentemente simple de la altura de individuos. A menos que especifiquemos varios factores como el momento en que se medirá la altura (se sabe que la altura de las personas varía en varios momentos del día), cómo se cuidaría la variabilidad debida al cabello, si la medición será con o sin zapatos, qué tipo de precisión se espera (correcto hasta una pulgada, 1/2 pulgada, centímetro, etc.) - incluso

- Títulos de trabajo en una empresa.
- El ciclo de vida del desarrollo de software (SDLC) modelo (como cascada, iterativo, ágil, etc.) seguido de diferentes proyectos de software

En escala nominal, los nombres de los diferentes gatos egories son solo etiquetas y ninguna relación entre ellos se supone. Las únicas operaciones que pueden ser llevado a cabo en escala nominal es la de contar el número de ocurrencias en las diferentes clases y determinar si dos ocurrencias tienen el mismo valor nominal. Sin embargo, los análisis estadísticos pueden llevarse a cabo para comprender cómo pertenecen las entidades ing a diferentes clases realizan con respecto a alguna otra variable de respuesta.

Escala ordinal: se refiere a la escala de medición donde los diferentes valores obtenidos a través del proceso de medición tiene un orden implícito En g. Los intervalos entre valores no son específicos. fied y no hay cero definido objetivamente elemento. Ejemplos típicos de mediciones en las escalas ordinales son:

- Niveles de habilidad (bajo, medio, alto)
- Integración del modelo de madurez de capacidades (CMMI) niveles de madurez de desarrollo de software organizaciones de opciones

Page 285

- Nivel de adherencia al proceso medido en una escala de 5 puntos de excelente, por encima del promedio, promedio, debajo del promedio y pobre, lo que indica el rango de adherencia total a no adherencia ence en absoluto

La medición en escala ordinal satisface el tránsito propiedad de la positividad en el sentido de que si A> B y B > C, luego A> C. Sin embargo, las operaciones aritméticas no puede llevarse a cabo en variables medidas en escamas ordinales. Por lo tanto, si medimos satisfaction en una escala ordinal de 5 puntos de 5 lo que implica un nivel muy alto de satisfacción y 1 implica un muy alto nivel de insatisfacción, no podemos decir que un puntaje de cuatro es el doble de un puntaje de dos. Por lo tanto, es mejor usar terminología como como excelente, por encima del promedio, promedio, por debajo del promedio, edad y pobre que los números ordinales para evitar el error de tratar una escala ordinal como escala de proporción. Es importante tener en cuenta que ordinal las medidas de escala son comúnmente mal utilizadas y tales el mal uso puede llevar a conclusiones erróneas [6 *, p274]. Un mal uso común de la escala ordinal sures es presentar una desviación media y estándar para el conjunto de datos, los cuales no tienen sentido. Sin embargo, podemos encontrar la mediana, como cálculo de la mediana implica contar solo.

Escalas de intervalo: con la escala de intervalo, nosotros llegar a una forma que sea cuantitativa en el ordi- Ningún sentido de la palabra. Casi todas las condiciones habituales Las medidas estadísticas son aplicables aquí, a menos que requieren conocimiento de un *verdadero* punto cero. El cero el punto en una escala de intervalos es una cuestión de conveniencia ción Las proporciones no tienen sentido, pero la diferencia entre niveles de atributos se pueden calcular y es significativo Algunos ejemplos de escala de intervalo de medida de seguimiento:

medido en escala de intervalo, ya que no es necesario deseoso de definir lo que haría la inteligencia cero media.

Si una variable se mide en escala de intervalo, la mayoría de los análisis estadísticos habituales como media, estándar la desviación, la correlación y la regresión pueden se llevará a cabo en los valores medidos.

Escala de relación: estos son muy comunes. Tered en la ciencia física. Estas escamas de mealos sures se caracterizan por el hecho de que las operaciones existen para determinar las 4 relaciones: igualdad, rango orden, igualdad de intervalos e igualdad de razones. Una vez que esta escala está disponible, su valor numérico los ues pueden transformarse de una unidad a otra simplemente multiplicando por una constante, por ejemplo, conversión de pulgadas a pies o centímetros. Cuando medida se están realizando en escala de proporción, existencia de un cero no arbitrario es obligatorio. Todos estadísticos las medidas son aplicables a la escala de razón; logaritmo el uso es válido solo cuando se usan estas escalas, como en el caso de decibelios. Algunos ejemplos de cociente las medidas son

- el número de declaraciones en un software programa
- temperatura medida en la escala Kelvin (K) o en Fahrenheit (F).

Una escala de medición adicional, la absoluta escala, es una escala de proporción con unicidad de la medida Por supuesto; es decir, una medida para la que no hay transformación es posible (por ejemplo, el número de programas Mers trabajando en un proyecto).

3.2. Medidas directas y derivadas

[6 *, c7s5]

- Medición de temperatura en diferentes escamas, como Celsius y Fahrenheit. Cenarpose T_1 y T_2 son temperaturas medidas en alguna escala. Observamos que el hecho de que T_1 es dos veces T_2 no significa que un objeto es dos veces más caliente que otro. También observamos que Los puntos cero son arbitrarios.
- Fechas del calendario. Mientras que la diferencia entre las fechas para medir el tiempo transcurrido es una medida concepto ingenioso, la relación no tiene sentido.
- Muchas medidas psicológicas aspiran a crear escalas de intervalos. La inteligencia es a menudo

Las medidas pueden ser directas o derivadas (algunas tiempos llamados medidas indirectas). Un ejemplo de una medida directa sería un recuento de cuántos veces que ocurrió un evento, como el número de defectos encontrados en un producto de software. Un derivado medida es una que combina medidas directas en de alguna manera que sea consistente con la medición método. Un ejemplo de una medida derivada sería estar calculando la productividad de un equipo como el cantidad de líneas de código desarrolladas por desarrollador mes. En ambos casos, el método de medición determina cómo realizar la medición.

Page 286

3.3. Fiabilidad y Validez

[4 *, c3s4, c3s5]

Una pregunta básica para cualquier medida método de medición es si la medida propuesta El método de ment realmente mide el concepto con buena calidad. La fiabilidad y la validez son las dos Criterios más importantes para abordar esta pregunta.

La fiabilidad de un método de medición es la medida en que la aplicación de la medida El método de aseguramiento produce mediciones consistentes resultados. Esencialmente, la *fiabilidad se* refiere a la consistencia Tendencia de los valores obtenidos cuando el mismo artículo se mide varias veces Cuando los resultados de acuerdo entre sí, el método de medición Se dice que es confiable. La fiabilidad generalmente depende en la definición operacional. Puede ser cuantificado mediante el uso del índice de variación, que es comp como la relación entre la desviación estándar y la media Cuanto más pequeño es el índice, más confiable los resultados de la medición.

Validez se refiere a si la medida El método realmente mide lo que pretendemos medir Por supuesto. La validez de un método de medición puede ser visto desde tres perspectivas diferentes: a saber, validez de constructo, validez de criterio y validez de contenido.

3.4. Evaluar la confiabilidad

[4 *, c3s5]

Existen varios métodos para evaluar reli capacidad; Estos incluyen el método test-retest, el método de forma alternativa, el método de mitades divididas y el método de consistencia interna. El easi- El mejor de estos es el método test-retest. En la prueba- método de prueba, simplemente aplicamos la medida método para los mismos temas dos veces. La correla- coeficiente de transición entre el primer y segundo conjunto de resultados de medición da la fiabilidad de la método de medida.

4. Diseño de ingeniería

[5 *, c1s2, c1s3, c1s4]

Los costos del ciclo de vida de un producto están muy influenciados por el diseño del producto. Esto es cierto para manu- productos fabricados, así como para productos de software. El diseño de un producto de software está guiado por las características que se incluirán y la calidad del atributo butes a ser provistos. Es importante tener en cuenta que los ingenieros de software usan el término "diseño" dentro de su propio contexto; mientras que hay algunos alidades, también hay muchas diferencias entre diseño de ingeniería como se discute en esta sección y diseño de ingeniería de software como se discute en El Software Design KA. El alcance del ingeniero En general, el diseño se considera mucho más amplio que el del diseño de software. El objetivo principal de esta sección es para identificar los conceptos necesarios para desarrollar una comprensión clara sobre el pro- cess del diseño de ingeniería.

Muchas disciplinas participan en la resolución de problemas. actividades donde hay una única solución correcta ción En ingeniería, la mayoría de los problemas tienen muchos soluciones y el foco está en encontrar una solución factible solución (entre las muchas alternativas) que mejor satisface las necesidades presentadas. El conjunto de pos- soluciones posibles a menudo se ven limitadas por explic- limitaciones impuestas por la ley, como el costo, disponible recursos y el estado de disciplina o dominio conocimiento. En problemas de ingeniería, a veces también hay restricciones implícitas (como el propiedades físicas de materiales o leyes de física ics) que también restringen el conjunto de soluciones factibles para un problema dado

4.1. Diseño de Ingeniería en Ingeniería Educación

La importancia del diseño de ingeniería en ingeniería la alta educación puede verse claramente por la alta expectativas mantenidas por varios organismos de acreditación ies para la educación en ingeniería. Tanto el cana- dian Junta de Acreditación de Ingeniería y el Junta de Acreditación de Ingeniería y Tecnología nología (ABET) tenga en cuenta la importancia de incluir Diseño de ingeniería en programas educativos.

La acreditación canadiense de ingeniería La junta incluye requisitos para la cantidad de experiencia en diseño de ingeniería / cursos que es necesario tanto para estudiantes de ingeniería como para calificaciones para los miembros de la facultad que enseñan tales cursos o supervisar proyectos de diseño. Sus criterios de acreditación establecen:

Página 287

> Diseño: capacidad de diseñar soluciones para problema de ingeniería complejo y abierto lems y para diseñar sistemas, componentes o procesos que satisfacen necesidades específicas con Atención adecuada a la salud y la seguridad. riesgos, estándares aplicables y económicos, con- cuidado ambiental, cultural y social sideraciones [8, p12]

De manera similar, ABET define ingeniería diseñar como

> el proceso de idear un sistema, compuesto nent, o proceso para satisfacer las necesidades deseadas. Eso es un proceso de toma de decisiones (a menudo iterativo) tive), en el que las ciencias básicas, las matemáticas ematics, y las ciencias de la ingeniería son aplicado para convertir recursos de manera óptima a Satisfacer estas necesidades declaradas. [9, p4]

Por lo tanto, está claro que el diseño de ingeniería es un componente vital en la formación y educación para Todos los ingenieros. El resto de esta sección será centrarse en varios aspectos del diseño de ingeniería.

4.2. El diseño como una actividad para resolver problemas [5 *, c1s4, c2s1, c3s3]

Cabe señalar que el diseño de ingeniería es primordial Es una actividad de resolución de problemas. Problemas de diseño son abiertos y más vagamente definidos. Ahí Por lo general, hay varias formas alternativas de resolver el el mismo problema. El diseño generalmente se considera ser un *problema perverso,* un término acuñado por primera vez por Horst Rittel en la década de 1960 cuando los métodos de diseño eran un tema de intenso interés. Rittel buscó una alternativa. tive al modelo lineal, paso a paso del diseño proceso siendo explorado por muchos diseñadores y teóricos del diseño y argumentaron que la mayoría de los problemas los lems dirigidos por los diseñadores son problemas perversos lems Como lo explicó Steve McConnell, un malvado el problema es uno que solo podría definirse claramente resolviéndolo o resolviendo parte de él. Esta paradoja implica, esencialmente, que un problema perverso tiene que ser resuelto una vez para definirlo claramente y luego resuelto nuevamente para crear una solución que funcione. Esto ha sido una idea importante para el diseño de software ers por varias décadas [10 *, c5s1].

4.3. Pasos involucrados en el diseño de ingeniería [7 *, c4]

La resolución de problemas de ingeniería comienza cuando un se reconoce la necesidad y ninguna solución existente lo hará satisfacer esa necesidad Como parte de esta resolución de problemas, Los objetivos de diseño que debe alcanzar la solución debe ser identificado Además, un conjunto de aceptación Los criterios de seguridad deben definirse y utilizarse para disuadir mina qué tan bien satisfará una solución propuesta la necesidad. Una vez que la necesidad de una solución a un problema ha sido identificado, el proceso de ingeniería el diseño tiene los siguientes pasos genéricos:

a) definir el problema
b) recopilar información pertinente
c) generar múltiples soluciones
d) analizar y seleccionar una solución
e) implementar la solución

Todos los pasos de diseño de ingeniería son iterativos. tive, y el conocimiento adquirido en cualquier paso de la el proceso puede usarse para informar tareas anteriores y desencadenar una iteración en el proceso. Estos pasos son ampliado en las secciones posteriores.

a. Define el problema. En esta etapa, la costumbre Los requisitos de er están reunidos. Informacion especifica También se mencionan las funciones y características del producto. diseñado de cerca. Este paso incluye refinar el planteamiento del problema para identificar el problema real para ser resuelto y establecer los objetivos y criterios de diseño para el éxito.

La definición del problema es una etapa crucial en proceso de diseño de ingeniería. Un punto a tener en cuenta es que esto El paso es engañosamente simple. Por lo tanto, suficiente cuidado debe tomarse para llevar a cabo este paso juiciosamente. Eso Es importante identificar las necesidades y vincular el éxito objetivos con las características requeridas del producto. También es una tarea de ingeniería limitar el alcance de un problema y su solución a través de la negociación entre las partes interesadas.

segundo. Recopilar información pertinente **.** En este punto, el diseñador intenta expandir su conocimiento Estado sobre el problema. Esto es vital, pero a menudo descuidado, etapa. Recopilando información pertinente puede revelar hechos que conducen a una redefinición de la

Page 288

problema, en particular, errores y falsos comienzos puede ser identificado Este paso también puede involucrar descomposición del problema en más pequeño, más subproblemas fácilmente resueltos.

Mientras recopila información pertinente, tenga cuidado refinar el diseño o impulsar la selección de una alternativa Solución de diseño nativo. Uno de los más importantes Tant Actividades en el diseño es documentación de la solución de diseño, así como de las compensaciones para el elecciones hechas en el diseño de la solución. Esta

debe tomarse para identificar cómo puede ser un producto usado y mal usado. También es importante Comprender el valor percibido del producto / servicio ofrecido Incluido en el pertinente la información es una lista de restricciones que deben ser satisfecho por la solución o que puede limitar el conjunto de soluciones factibles.

do. Genera múltiples soluciones. Durante esta etapa, se desarrollan diferentes soluciones al mismo problema abierto Ya se ha dicho que el problema del diseño los lems tienen múltiples soluciones. El objetivo de esto el paso es conceptualizar múltiples soluciones posibles y refinarlas a un nivel de detalle suficiente que se puede hacer una comparación entre ellos.

re. Analizar y seleccionar una solución. Una vez alternativa se han identificado soluciones, deben ser ana- analizado para identificar la solución que mejor se adapte a la Situación de alquiler. El análisis incluye un funcional análisis para evaluar si el diseño propuesto cumpliría los requisitos funcionales. Físico Las soluciones que involucran a usuarios humanos a menudo incluyen análisis de la ergonomía o facilidad de uso de La solución propuesta. Otros aspectos de la solución ión, como la seguridad y responsabilidad del producto, un eco- análisis nomico o de mercado para asegurar un retorno (beneficios) sobre la solución, predicciones de rendimiento y análisis sis para cumplir con características de calidad, oportunidades por entrada incorrecta de datos o mal funcionamiento del hardware y así sucesivamente, pueden estudiarse. Los tipos y cantidad de análisis utilizados en una solución propuesta son dependientes mella en el tipo de problema y las necesidades que el solución debe abordar, así como las restricciones impuesta al diseño.

mi. Implementa la solución. La fase final de la proceso de diseño es implementación. Implemen- tation se refiere al desarrollo y prueba de solución propuesta. A veces un preliminar, se puede desarrollar una solución parcial llamada *prototipo* o abierto inicialmente para probar la solución de diseño propuesta ción bajo ciertas condiciones. Comentarios resultantes de probar un prototipo se puede usar para el trabajo debe llevarse a cabo de manera tal que La solución al problema del diseño puede ser comunicado claramente a los demás.

Las pruebas y la verificación nos llevan de vuelta al criterios de éxito. El ingeniero necesita idear pruebas de tal manera que la capacidad del diseño para cumplir con el Se demuestra el criterio de éxito. Mientras diseño Durante las pruebas, el ingeniero debe pensar detenidamente diferentes modos de falla posibles y luego diseñar pruebas basadas en esos modos de falla. El ingeniero puede optar por llevar a cabo experimentos diseñados para Evaluar la validez del diseño.

5. Modelado, simulación y creación de prototipos.
[5 *, c6] [11 *, c13s3] [12 *, c2s3.1]

El modelado es parte del proceso de abstracción utilizado para representar algunos aspectos de un sistema. Simula- ción utiliza un modelo del sistema y proporciona un medios para realizar experimentos diseñados con ese modelo para comprender mejor el sistema, es comportamiento y relaciones entre subsistemas, así como para analizar aspectos del diseño. Mod- Eling y la simulación son técnicas que pueden ser utilizado para construir teorías o hipótesis sobre el comportamiento del sistema; los ingenieros luego usan esos teorías para hacer predicciones sobre el sistema.

La creación de prototipos es otro proceso de abstracción donde una representación parcial (que captura aspectos de interés) del producto o sistema es construido. Un pro- totype puede ser una versión inicial del sistema pero carece de la funcionalidad completa de la versión final.

5.1. Modelado

Un modelo es siempre una abstracción de algo real. o artefacto imaginado. Los ingenieros usan modelos en muchas formas como parte de su resolución de problemas ocupaciones. Algunos modelos son físicos, como un construcción en miniatura a escala de un puente o edificio. Otros modelos pueden ser no físicos representaciones, como un dibujo CAD de un diente o un modelo matemático para un proceso. Modelos ayudar a los ingenieros a razonar y comprender aspectos de

un problema. También pueden ayudar a los ingenieros a soportar lo que saben y lo que no saben saber sobre el problema en cuestión.

Hay tres tipos de modelos: icónicos, ana- lógica y simbólica. Un modelo icónico es un visu aliado equivalente pero incompleto bidimensional o representación tridimensional, por ejemplo, mapas, globos o modelos de estructuras construidos a escala- Tures como puentes o carreteras. Un icónico El modelo en realidad se parece al artefacto modelado.

En contraste, un modelo analógico es funcionalmente Representación equivalente pero incompleta. Ese es decir, el modelo se comporta como el artefacto físico aunque no se parezca físicamente a él. Los ejemplos de modelos analógicos incluyen una miniatura avión para pruebas de túnel de viento o una computadora simulación de un proceso de fabricación.

Finalmente, un modelo simbólico es un nivel más alto de abstracción, donde el modelo se representa usando

Un problema importante en el desarrollo de un La simulación discreta es la de inicialización. antes de se puede ejecutar una simulación, los valores iniciales de todos Se deben proporcionar las variables de estado. Como el simu- el diseñador de lation puede no saber qué valores iniciales son apropiados para las variables de estado, estos valores Los ues pueden elegirse de manera algo arbitraria. por ejemplo, podría decidirse que una cola debería se inicializará como vacío e inactivo. Tal elección de condición inicial puede tener un significativo pero irreconocible impacto reconocido en el resultado de la simulación.

5.3. Prototipos

Construir un prototipo de un sistema es otro proceso de abstracción En este caso, una versión inicial del sistema se construye, a menudo mientras el sistema tem está siendo diseñado. Esto ayuda a los diseñadores. determinar la viabilidad de su diseño.

símbolos como ecuaciones. El modelo captura los aspectos relevantes del proceso o sistema en forma simbólica Los símbolos se pueden usar para aumentar la comprensión del ingeniero de la final sistema. Un ejemplo es una ecuación como $F = Ma$. Tales modelos matemáticos se pueden usar para describir y predecir propiedades o comportamiento de la sistema final o producto.

5.2. Simulación

Todos los modelos de simulación son una especificación de o ity. Un tema central en la simulación es abstraer y especificar una simplificación apropiada de realidad. Desarrollar esta abstracción es de vital importancia importancia, como especificación errónea de la abstracción la acción invalidaría los resultados de la simulación ejercicio. La simulación se puede usar para una variedad de fines de prueba.

La simulación se clasifica según el tipo de sistema en estudio. Por lo tanto, la simulación puede ser continuo o discreto. En el contexto del software ingeniería, el énfasis estará principalmente en simulación discreta Las simulaciones discretas pueden modelo de programación de eventos o interacción de procesos. Los componentes principales en tal modelo incluyen entidades, actividades y eventos, recursos, el estado del sistema, un reloj de simulación y un azar generador de números La salida es generada por simulación y debe ser analizado.

Hay muchos usos para un prototipo, incluyendo- ing la obtención de requisitos, el diseño y refinamiento de una interfaz de usuario para el sistema, validación de requisitos funcionales, etc. los objetivos y propósitos para construir el prototipo tipo determinará su construcción y el nivel de abstracción utilizada.

El papel de los prototipos es algo diferente. entre sistemas físicos y software. Con sistemas físicos, el prototipo puede en realidad ser la primera versión completamente funcional de un sistema o puede ser un modelo del sistema. En software ingeniería, los prototipos también son abstractos modelo de parte del software pero generalmente no son construido con todo el diseño arquitectónico, mance y otras características de calidad esperadas en el producto terminado En cualquier caso, prototipo la construcción debe tener un propósito claro y ser planeado, monitoreado y controlado, es una tecnología nique para estudiar un problema específico dentro de un limitado contexto [6 *, c2s8].

En conclusión, modelado, simulación y pro los tipados son técnicas poderosas para estudiar el comportamiento de un sistema desde una perspectiva dada. Todo se puede usar para realizar experimentos diseñados para estudiar varios aspectos del sistema. Cómo- alguna vez, estas son abstracciones y, como tales, pueden no modelar todos los atributos de interés.

Page 290

6. Normas

[5 *, c9s3.2] [13 *, c1s2]

Moore afirma que un

> estándar puede ser; (a) un objeto o medida de comparación que define o representa la magnitud de una unidad; (b) una caracterización ción que establece tolerancias permitidas para categorías de artículos; y (c) un grado o nivel de excelencia o logro requerido. Las normas son de naturaleza definitoria, establecidas o bien para una mayor comprensión y interacción o para reconocer observado (o deseado) normas de características exhibidas o comportamiento. [13 *, p8]

Las normas proporcionan requisitos, especificaciones acciones, pautas o características que deben ser observado por ingenieros para que los productos, proceses y materiales tienen niveles aceptables de calidad. Las cualidades que proporcionan diversas normas vide puede ser de seguridad, confiabilidad u otro caracteristicas de producto. Los estándares son considerados crítico para ingenieros e ingenieros se espera que estar familiarizado y usar el estándar apropiado papás en su disciplina.

Cumplimiento o conformidad con un estándar permite una organización dice al público que ellos (o sus productos) cumplen los requisitos establecidos en ese estándar Por lo tanto, las normas dividen a las organizaciones o sus productos en aquellos que se ajustan a el estándar y los que no. Para un estándar organizaciones regionales y reconocidas gubernamentalmente Zations que generan estándares para esa región o país. Por ejemplo, en los Estados Unidos, hay son más de 300 organizaciones que desarrollan estándares papás Estos incluyen organizaciones como el Instituto Americano de Normas Nacionales (ANSI), la sociedad americana para pruebas y materiales (ASTM), la Sociedad de Ingenieros Automotrices (SAE) y Underwriters Laboratories, Inc. (UL), así como el gobierno de los Estados Unidos. Para más detalles sobre estándares utilizados en ingeniería de software, ver Apéndice B sobre normas.

Hay un conjunto de principios de uso común. detrás de los estándares. Los creadores de normas intentan tener consenso en torno a sus decisiones. Ahi esta generalmente una apertura dentro de la comunidad de interés para que una vez que se haya establecido un estándar, haya es una buena posibilidad de que sea ampliamente aceptado. La mayoría de las organizaciones de estándares tienen bien definidas procesos para sus esfuerzos y adherirse a aquellos procesa con cuidado. Los ingenieros deben ser conscientes de los estándares existentes pero también deben actualizar sus comprensión de los estándares como esos estándares cambian con el tiempo.

En muchos esfuerzos de ingeniería, conocer y entender los estándares aplicables es crítico y la ley puede incluso requerir el uso de normas En estos casos, los estándares a menudo representan reenviar los requisitos mínimos que deben cumplir el esfuerzo y, por lo tanto, son un elemento en el contexto tensiones impuestas en cualquier esfuerzo de diseño. El ingenio Neer debe revisar todos los estándares actuales relacionados con un esfuerzo determinado y determinar cuál debe ser

para ser útil, la conformidad con el estándar debe agregar valor, real o percibido, al producto, proceso o esfuerzo.

Además de los objetivos organizacionales, los estándares se utilizan para otros fines, como como proteger al comprador, proteger el negocio, y una mejor definición de los métodos y procedimientos para ser seguido por la práctica. Normas también proporcionar a los usuarios una terminología común y esperanzas de heredar.

Hay muchos reconocidos internacionalmente organizaciones de elaboración de normas, incluida la Unión Internacional de Telecomunicaciones (UIT), la Comisión electrotécnica internacional (IEC), IEEE y la Organización Internacional para la estandarización (ISO). Además, hay reunió. Sus diseños deben incorporar cualquier y todas las restricciones impuestas por el estándar aplicable Dard Estándares importantes para los ingenieros de software. se discuten con más detalle en un apéndice específicamente sobre este tema.

7. Análisis de causa raíz

[4 *, c5, c3s7, c9s8] [5 *, c9s3, c9s4, c9s5] [13 *, c13s3.4.5]

El análisis de causa raíz (RCA) es un proceso diseñado para investigar e identificar por qué y cómo evento indeseable ha sucedido. Causas fundamentales son causas subyacentes El investigador debe intentar identificar causas subyacentes específicas de El evento que ha ocurrido. El objetivo principal

de RCA es para prevenir la recurrencia de los indeseados evento capaz Por lo tanto, cuanto más específica sea la investigación, puede ser sobre por qué ocurrió un evento, cuanto más fácil será para prevenir la recurrencia. Una forma común identificar las causas subyacentes específicas es pedirle a serie de preguntas de *por qué* .

7.1. Técnicas para conducir la causa raíz Análisis

[4 *, c5] [5 *, c3]

Hay muchos enfoques utilizados tanto para la calidad control y análisis de causa raíz. El primer paso en cualquier esfuerzo de análisis de causa raíz es identificar lo real problema. Técnicas como la declaración-reformulación-ment, diagramas de por qué, el método de revisión, estado actual y diagramas de estado deseados, y el El enfoque de ojo fresco se utiliza para identificar y refinar El verdadero problema que debe abordarse.

Una vez que se ha identificado el problema real, entonces el trabajo puede comenzar a determinar la causa de la problema. Ishikawa es conocida por las siete herramientas. por el control de calidad que promovió. Algunos de esas herramientas son útiles para identificar las causas para un problema dado Esas herramientas son hojas de verificación o listas de verificación, diagramas de Pareto, histogramas, ejecutar gráficos, diagramas de dispersión, gráficos de control y espina de pescado o diagramas de causa y efecto. Más recientemente, otros enfoques para mejorar la calidad El análisis de causa y raíz ha surgido. Algunos ejemplos de estos nuevos métodos son la afinidad diagramos, diagramas de relaciones, diagramas de árbol, matriz gráficos, gráficos de análisis de datos matriciales, deci de procesos Tablas de programas de sion y diagramas de flechas. Unos pocos de estas técnicas se describen brevemente a continuación.

Un diagrama de espina de pescado o de causa y efecto es un manera de visualizar los diversos factores que afectan alguna característica La línea principal en el diagrama. representa el problema y las líneas de conexión representar los factores que condujeron o influyeron en problema. Esos factores se dividen en sub-factores y sub-subfactores hasta que la causa raíz pueda ser identificado.

Un enfoque muy simple que es útil en calidad. tigación es el uso de una lista de verificación. Las listas de verificación son Una lista de puntos clave en un proceso con tareas que debe ser completado. A medida que se completa cada tarea, está marcado en la lista. Si ocurre un problema, entonces a veces la lista de verificación puede identificar rápidamente tareas que pueden haberse omitido o solo par-Tialmente completado.

Finalmente, los diagramas de relaciones son un medio para desarmar jugando relaciones complejas. Dan visual Apoyo al pensamiento de causa y efecto. El dia-gramo relaciona lo específico con lo general, revelando Causas clave y efectos clave.

El análisis de causa raíz tiene como objetivo prevenir la recurrencia de eventos indeseables. Reducción de la variación debida a causas comunes requiere utilidad zation de una serie de técnicas. Un importante el punto a tener en cuenta es que estas técnicas deberían ser se usa sin conexión y no necesariamente en respuesta directa a la ocurrencia de algún evento indeseable. Algunas de las técnicas que pueden usarse para reducir la variación debido a causas comunes se dan abajo.

1. Los diagramas de causa y efecto pueden usarse para identificar las causas sub y sub-sub.
2. El análisis de árbol de fallas es una técnica que puede ser solía entender las fuentes de fallas.
3. Los experimentos diseñados pueden ser utilizados para soportar el impacto de diversas causas en el ocurrencia de eventos indeseables (ver Empir-Métodos prácticos y técnicas experimentales. en este KA).
4. Varios tipos de análisis de correlación pueden ser solía entender la relación entre Varias causas y su impacto. Estas tecnicas se pueden usar niques en casos en que Ing experimentos controlados es difícil pero se pueden recopilar datos (ver Análisis estadístico-sis en este KA).

Page 292

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	07 ger 20 norte tu Dr [2 *] ery an tgom en METRO	**06 r 20 tu transmisión exteriores d L [3 *] ll an tu norte**	**02 un 20 K [4 *]**	**03 d 20 Olan V [5 *]**	**09 airley 20 F [6 *]**	**04 ey 20 ock T [7 *]**	**04 ell 20 norte en [10 *] cC METRO**	**07 caid 20 en d K [11 *] ey an en Ch**	**06 ile 2011 erv metro gm S [12 *]**	**oore 20 METRO [13 *]**
1. Empírico Métodos y Experimental Tecnicas	c1									
1.1. Diseñado Experimentar										
1.2. De observación Estudiar										
1.3. Retrospectivo Estudiar										
2. Estadística Análisis	c9s1, c2s1	c10s3								
2.1. Concepto de Unidad de Análisis (Muestreo Unidades), Muestra, y población	c3s6, c3s9, c4s6, c6s2, c7s1, c7s3, c8s1, c9s1									
2.2. Conceptos de Correlación y Regresión	c11s2, c11s8									
3. Medida			c3s1, c3s2	c4s4	c7s5					
3.1. Niveles (Escalas) de Medición			c3s2		c7s5	p442 –447				
3.2. Directo y derivado Medidas										

Page 293

	07 ger 20 norte tu Dr ery an tgom en METRO [2 *]	06 r 20 tu transmisión exterior d L ll an tu norte [3 *]	02 un 20 K [4 *]	03 u 20 Olan V [5 *]	09 airley 20 F [6 *]	04 ey 20 ock T [7 *]	04 ell 20 norte en cC METRO [10 *]	07 caid 20 en d K ey an en Ch [11 *]	ile 2011 erv metro om S [12 *]	06 oore 20 METRO [13 *]
3.3. Confiabilidad y validez			c3s4, c3s5							
3.4. Evaluar Confiabilidad			c3s5							
4. Ingeniería Diseño				c1s2, c1s3, c1s4						
4.1. Diseñar en Ingenieria Educación										
4.2. Diseño como un problema Actividad de resolución				c1s4, c2s1, c3s3			c5s1			
4.3. Pasos Envuelto en Ingenieria Diseño						c4				
5. Modelado, Creación de prototipos, y Simulación				c6				c13s3	c2 s3.1	
5.1. Modelado										
5.2. Simulación										
5.3. Prototipos										
6. Normas				c9 s3.2						c1s2
7. Causa raíz Análisis			c5, c3s7, c9s8	c9s3, c9s4, c9s5						c13 s3.4.5
7.1. Tecnicas para realizar Causa principal Análisis			c5	c3						

LECTURAS ADICIONALES

A. Abran, *Software Metrics y Software Metrología* . [14]

Este libro proporciona muy buena información sobre uso adecuado de los términos medida, medida Método y resultado de la medición. Proporciona fuerte material de soporte para toda la sección en Medición.

WG Vincenti, *lo que los ingenieros saben y cómo Ellos lo saben* . [15]

Este libro ofrece una introducción interesante. ción a fundamentos de ingeniería a través de una serie de estudios de caso que muestran muchos de los fundamentos Conceptos nacionales tal como se utilizan en la ingeniería del mundo real. aplicaciones.

Referencias

[1] *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010.

[2 *] DC Montgomery y GC Runger, *Estadística aplicada y probabilidad para Ingenieros* , 4a ed., Wiley, 2007.

[3 *] L. Null y J. Lobur, *Los fundamentos de Organización y arquitectura de computadoras* , 2ª ed., Jones and Bartlett Publishers, 2006

[4 *] SH Kan, *métricas y modelos en software Ingeniería de calidad* , 2ª ed., Addison-Wesley, 2002.

[5 *] G. Voland, *Ingeniería por diseño* , 2ª ed., Prentice Hall, 2003.

[6 *] RE Fairley, *Gestión y liderazgo Proyectos de software* , computadora Wiley-IEEE Society Press, 2009.

[7 *] S. Tockey, rendimiento *del software: maximización el retorno de su inversión en software* , Addison-Wesley, 2004.

[9] Acreditación de ingeniería ABET Comisión, "Criterios para acreditar Programas de ingeniería, 2012-2013 " ABET, 2011; www.abet.org/uploadedFiles/ Acreditación / Acreditación_Proceso / Acreditación_Documentos / Actual / eac-criterios-2012-2013.pdf .

[10 *] S. McConnell, *Código completo* , 2ª ed., Microsoft Press, 2004.

[11 *] EW Cheney y DR Kincaid, *numéricos Matemáticas e Informática* , 6ª ed., Brooks / Cole, 2007.

[12 *] I. Sommerville, *Ingeniería de Software* , noveno ed., Addison-Wesley, 2011.

[13 *] JW Moore, *La hoja de ruta hacia el software Ingeniería: una guía basada en estándares* , Wiley-IEEE Computer Society Press, 2006.

[14] A. Abran, *Software Metrics and Software Metrología* , Wiley-IEEE Computer Society Prensa, 2010.

[15] WG Vincenti, *lo que los ingenieros saben y cómo lo saben* , John Hopkins

[8] Junta Canadiense de Acreditación de Ingeniería, Ingenieros de Canadá, "Criterios de acreditación y procedimientos ", Consejo Canadiense de Ingenieros Profesionales, 2011; www.engineercanada.ca/files/w_Accredit_Criterios_Procedimientos_2011.pdf .

University Press, 1990.

APÉNDICE A

DESCRIPCIÓN DEL ÁREA DE CONOCIMIENTO PRESUPUESTO

INTRODUCCIÓN

Este documento presenta las especificaciones proenviado a los editores del área de conocimiento (KA Editors) con respecto a las descripciones del área de conocimiento (Descripciones KA) de la edición de la Versión 3 (V3) de la *Guía del organismo de ingeniería de software de conocimiento (Guía SWEBOK)* . Este documento También permitirá a los lectores, revisores y usuarios entender claramente qué especificaciones se usaron al desarrollar esta versión de *SWEBOK Guíe* .

Este documento comienza situando el *SWE-La Guía BOK* como documento fundamental para IEEE Computer Society suite de software de ingeniería productos de neering y más ampliamente dentro del comunidad de ingeniería de software en general. los papel de la línea de base y el control de cambios Luego se describe el tablero. Criterios y requisitos se definen para el desglose de temas, por la razón subyacente a estos desgloses y la breve descripción de los temas, y para referencia Diferencia materiales. Los documentos de entrada importantes s) h también identificado, y su papel dentro del proyecto es explicado. Problemas no relacionados con el contenido, como la presentación también se discuten las pautas de formato y estilo.

LA GUÍA SWEBOK ES UNA DOCUMENTO FUNDACIONAL PARA EL IEEE COMPUTER SOCIETY SUITE OF PRODUCTOS DE INGENIERÍA DE SOFTWARE

grande en particular a través del reconocimiento oficial de la versión 2004 como informe técnico ISO / IEC 19759: 2005. La lista de áreas de conocimiento (KAs) y el desglose de temas dentro de cada KA es descrito y detallado en la introducción de este *SWEBOK Guide* .

En consecuencia, la *guía SWEBOK* es fundamental tional a otras iniciativas dentro de la Comisión IEEE Sociedad de informática:

a) La lista de KAs y el desglose de temas. dentro de cada KA también son adoptados por el soft- certificación de ingeniería de artículos y asociados productos de desarrollo profesional ofrecidos por la IEEE Computer Society (ver www.computer.org/certification).
b) La lista de KAs y el desglose de top- ics también son fundamentales para el software pautas curriculares de ingeniería desarrolladas o avalado por la IEEE Computer Society (www.computer.org/portal/web/education/Currículum)
c) La lista de referencia consolidada (ver Apéndice dix C), que significa la lista de recomendaciones materiales de referencia (al nivel de sección número) que acompaña al desglose de los temas dentro de cada KA también son adoptados por el certificación de ingeniería de software y as- productos ciados de desarrollo profesional ofrecido por la IEEE Computer Society.

BASE Y CONTROL DE CAMBIO

La *guía SWEBOK* es una IEEE Computer Society documento emblemático y estructural para el IEEE Conjunto de ingenieros de software de Computer Society ing productos. La *guía SWEBOK* también es más ampliamente reconocido como documento fundamental dentro de la comunidad de ingeniería de software en

TABLERO

Debido a la naturaleza estructural de *SWEBOK Guía* y su adopción por otros productos, una base la línea se desarrolló al inicio del proyecto compuesto por la lista de KAs, el desglose de

Página 297

temas dentro de cada KA, y el Ref. Consolidado Lista de referencia.

Una Junta de Control de Cambio (CCB) ha estado en lugar para el desarrollo de esta versión para Handle todas las solicitudes de cambio a esta línea base próxima de los editores de KA, surgidos durante la revisión proceso, o de otra manera. Las solicitudes de cambio deben ser aprobado tanto por los editores de la *guía SWEBOK* y por el CCB antes de ser implementado. Esta CCB está compuesto por miembros de las iniciativas. enumerados anteriormente y actuando bajo la autoridad de la Comité de Ingeniería de Software y Sistemas de la IEEE Computer Society Professional Activi-Lazos Junta.

CRITERIOS Y REQUISITOS PARA EL DESGLOSE DE LOS TEMAS DENTRO Un área de conocimiento

a) Los editores de KA tienen instrucciones de adoptar la base Desglose de líneas de temas.
b) Se espera que el desglose de temas sea "Razonable", no "perfecto".
c) El desglose de temas dentro de un KA debe descomponer el subconjunto de Software Engiun cuerpo de conocimiento que es "genreconocido por vía oral ". Ver más abajo para más discusión detallada de este punto.
d) El desglose de temas dentro de un KA debe no presume dominios de aplicación específicos, necesidades comerciales, tamaños de organizaciones, organizativos estructuras nacionales, filosofías de gestión, modelos de ciclo de vida de software, tecnología de softwaregies, o métodos de desarrollo de software.
e) El desglose de temas debe, tanto como sea posible, sea compatible con la variedad ous escuelas de pensamiento dentro del software Ingenieria.
f) El desglose de temas dentro de un KA debe ser compatible con el desglose de softingeniería de artículos generalmente encontrada en la industria probar y en la literatura de ingeniería de software y normas.
g) Se espera que el desglose de temas sea lo más inclusivo posible
h) La *guía SWEBOK* adopta el puesto que a pesar de que los siguientes "temas" son comunes en todas las áreas de conocimiento, también son una parte integral de todo conocimiento Áreas y por lo tanto deben ser incorporadas en el desglose propuesto de temas de cada área de conocimiento. Estos comunes los temas son medición, calidad (en general eral), y seguridad.
i) El desglose de temas debería ser como máximo Dos o tres niveles de profundidad. Aunque no se impone un límite superior o inferior en el número Ber de temas dentro de cada KA, un razonable y se espera un número manejable de temas para ser incluido en cada KA. El énfasis debe También se pondrá en la selección de los temas ellos mismos en lugar de en su organización en una jerarquía apropiada
j) Los nombres de los temas deben ser lo suficientemente significativos ser significativo incluso cuando se cita fuera de la *SWEBOK Guide* .
k) La descripción de un KA incluirá un cuadro (en forma de árbol) que describe el conocimiento Descompostura.

CRITERIOS Y REQUISITOS PARA DESCRIBIR TEMAS

Los temas solo necesitan ser suficientemente descritos para que el lector puede seleccionar el material de referencia apropiado rial de acuerdo a sus necesidades. Descripción del tema No deben ser prescriptivas.

CRITERIOS Y REQUISITOS PARA MATERIAL DE REFERENCIA

a) Los editores de KA tienen instrucciones de utilizar la referencia ences (al nivel del número de sección) asignaciones atendido a su KA por el Consolidated Refer-Lista de referencias como sus referencias recomendadas.
b) Hay tres categorías de referencia. material:

»Referencias recomendadas. El conjunto de Referencias recomendadas (al nivel del número de sección) se conoce colectivamente como la Lista de referencia consolidada.
" Lecturas adicionales.
»Referencias adicionales citadas en el KA Descripción (por ejemplo, la fuente de una cita o material de referencia en apoyo de una justificación detrás de un particular argumento).

Page 298

c) La *Guía SWEBOK* está diseñada por definiser selectivo en su elección de temas y material de referencia asociado. La lista de el material de referencia debe verse claramente como una "selección informada y razonable" en lugar de como una lista definitiva.
d) El material de referencia puede ser capítulos de libros, artículos de revistas arbitradas, conferencias arbitradas papeles de referencia, técnicos o industriales arbitrados informes, o cualquier otro tipo de artículo reconocido hecho. Referencias a otro KA, subárea o El tema también está permitido.
e) El material de referencia debe estar generalmente disponible capaz y no debe ser de naturaleza confidencial.
f) El material de referencia debe estar en inglés.
g) Criterios y requisitos para recomendaciones material de referencia o referencia consolidada Lista de ence:

»Colectivamente la lista de Recomendados Las referencias deben ser

yo. completa: cubriendo todo alcance de la *guía SWEBOK*
ii) suficiente: proporcionar suficiente información para describir "generaliado aceptado "conocimiento
iii) consistente: no proporcionar contraconocimiento ficticio ni conflicto prácticas de ing
iv. creíble: reconocido como proveedor tratamiento experto
v. actual: tratar el tema en una manera que sea acorde con actualmente generalmente aceptado conocimiento
vi. sucinto: lo más corto posible (ambos en número de referencia artículos y en el recuento total de páginas) sin fallar otros objetivos.

»El material de referencia recomendado debe ser identificado para cada tema. Cada recomelemento de referencia reparado puede, por supuesto Cubrir múltiples temas. Excepcionalmente, un el tema puede ser autodescriptivo y no citar un elemento de material de referencia (por ejemplo, un tema que es una definición o un tema para que la descripción en sí sin ningún el material de referencia citado es suficiente para los objetivos de la *Guía SWEBOK*).
»Cada referencia a lo recomendado el material de referencia debe ser tan preciso como sea posible identificando qué específico capítulo o sección es relevante.
»Una matriz de material de referencia (a la nivel de número de sección) versus temas debe ser provisto.
»Se recomienda una cantidad razonable de el material de referencia debe ser identificado por cada KA. Las siguientes pautas debe usarse para determinar cómo mucho es razonable:

yo. Si la referencia recomendada el material fue escrito en un manera de seguir el pro desglose planteado de temas y en un estilo uniforme (por ejemplo, en un nuevo libro basado en la propuesta Descripción de KA), un tarcruzar todos los KAs para el número de páginas serían 750. Sin embargo, este objetivo puede no ser alcanzable al seleccionar la referencia existente material debido a diferencias en estilo y superposición y redundancia entre la referencia seleccionada materiales
ii) En otras palabras, el objetivo para el número de páginas para todo colección de referencias recomendadas Ences de la *Guía SWEBOK* es en el rango de 10,000 a 15,000 páginas
iii) Otra forma de ver esto es que la cantidad de recomendada material de referencia sería razonable si consistiera en el estudiar material sobre este KA para un licencia de ingeniería de software examen que aprobaría un graduado después de completar cuatro años de experiencia laboral.

»Material de referencia adicional puede ser incluido por el Editor de KA en un "Más Lista de lecturas:

»Estas lecturas adicionales deben estar relacionadas con los temas en el desglose en lugar de por ejemplo, a temas más avanzados.
»La lista debe ser anotada (dentro de 1 párrafo por referencia) en cuanto a por qué esto material de referencia fue incluido en el lista de lecturas adicionales. Lecturas adicionales podría incluir: nuevas versiones de un existeing referencia ya incluida en el referencias recomendadas, alternativas puntos de vista sobre un KA, o un tratamiento semilustrar

¿A QUÉ SIGNIFICA "EN GENERAL ¿CONOCIMIENTO RECONOCIDO?

El cuerpo de conocimiento de ingeniería de software es un término todo incluido que describe la suma de conocimiento dentro de la profesión de software Ingenieria. Sin embargo, la *Guía SWEBOK* busca para identificar y describir ese subconjunto del cuerpo de conocimiento que generalmente se reconoce o, en En otras palabras, el núcleo del conocimiento. Apostar- lo que el conocimiento "generalmente reconocido"

ment de un KA.
»Una pauta general a seguir es 10 o menos lecturas adicionales por KA.
»No hay matriz de la referencia materiales enumerados en lecturas adicionales y El desglose de los temas.

i) Criterios y requisitos con respecto a Referencias nacionales citadas en la Descripción de KA:

»La *guía SWEBOK* no es una investigación documento y sus lectores serán variados ied Por lo tanto, un delicado equilibrio debe mantenerse entre garantizar un alto nivel de legibilidad dentro del documento manteniendo su excelencia técnica lence Material de referencia adicional por lo tanto solo debe ser introducido por el editor de KA si es necesario discusión. Los ejemplos son para identificar el fuente de una cita o para citar una referencia elemento en apoyo de una razón detrás de un argumento particular e importante.

ESTRUCTURA COMÚN

Las descripciones de KA deben usar la siguiente estructura:

- Acrónimos
- Introducción
- Desglose de temas de la KA (incluyendo un figura que describe el desglose)
- Matriz de temas versus material de referencia
- Lista de lecturas adicionales
- referencias

la ventaja es relativa a otros tipos de conocimiento.

La figura A.1 propone un esquema de tres categorías para clasificando el conocimiento.

El Project Management Institute en su *guía al Cuerpo de Conocimientos de Gestión de Proyectos* define el conocimiento "generalmente reconocido" para gestión de proyectos como siendo:

> ese subconjunto de la gestión del proyecto cuerpo de conocimiento generalmente reconocido Como buena práctica. "Generalmente reconocido" significa el conocimiento y las prácticas descritos son aplicables a la mayoría de los proyectos la mayor parte del tiempo, y hay consenso sobre su valor y utilidad. "Bueno práctica "significa que hay un acuerdo general que la aplicación de estas habilidades, herramientas, y las técnicas pueden aumentar las posibilidades de éxito en una amplia gama de proyectos. "Buena práctica" no significa que el el conocimiento descrito siempre debe ser aplicado uniformemente a todos los proyectos; la organizacion y / o equipo de gestion de proyectos es responsable de determinar qué es apropiado para cualquier proyecto dado. [1]

El conocimiento "generalmente aceptado" también podría ser visto como conocimiento para ser incluido en el estudio material de un examen de licencia de ingeniería de software (en los EE. UU.) que un graduado tomaría después completando cuatro años de experiencia laboral. Estas dos definiciones deben verse como complementarias.

También se espera que los editores de KA sean algo mirando hacia adelante en su interpretación por taktteniendo en cuenta no solo lo que es aliado reconocido "hoy y pero lo que esperan será "generalmente reconocido" en un período de 3 a 5 años periodo de tiempo.

Page 300

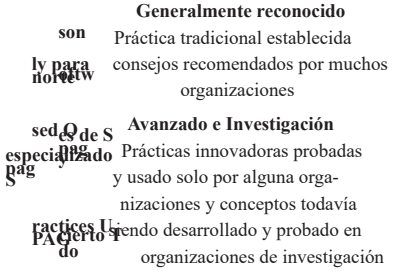

Figura A.1. Categorías de conocimiento

LONGITUD DE KA DESCRIPCIÓN

Las descripciones de KA deben ser aproximadamente de 10 a 20 páginas. utilizando la plantilla de formato para publicaciones de documentos Listed en las actas de conferencias del IEEE Sociedad Informática. Esto incluye texto, referencias, apéndices, tablas, etc. Esto, por supuesto, no incluir los materiales de referencia en sí.

DOCUMENTOS RELACIONADOS IMPORTANTES

y los Comités de Normas de Ingeniería de Sistemas. También ha sido designado como un estándar fundamental por el Software e Ingeniería de Sistemas Stan- Comité dards (S2ESC) del IEEE.

Aunque no pretendemos que la *Guía para el cuerpo de conocimiento de ingeniería de software* sea totalmente conforme a 12207, este estándar sigue siendo un entrada clave a la *Guía SWEBOK* y atención especial se tomará a lo largo de la *guía SWEBOK* con respecto a la compatibilidad de la *Guía* con el 12207 estándar.

3. JW Moore, la hoja de *ruta hacia el software Ingeniería: una guía basada en estándares* , Wiley-IEEE Computer Society Press, 2006. [4 *]

Este libro describe el alcance, roles, usos y tendencias de desarrollo de los soft- más utilizados normas de ingeniería de artículos. Se concentra en actividades importantes de ingeniería de software ity y gestión de proyectos, ingeniero de sistemas ing, confiabilidad y seguridad. El analisis y la reagrupación de las colecciones estándar expone el lector a las relaciones clave entre los estándares.

1. *Graduado en Ingeniería de Software 2009 (GSwE2009): Directrices curriculares para Programas de Posgrado en Software Ingeniería* , 2009; www.gswe2009.org . [2]

Este documento "proporciona pautas y recomendaciones "recomendaciones" para definir los planes de estudio de un programa de maestría profesional en software Ingenieria. Se identifica la *Guía SWEBOK* como una "referencia primaria" en el desarrollo del cuerpo de conocimiento subyacente a estas pautas. Esta documento ha sido respaldado oficialmente por el IEEE Computer Society y patrocinado por el Asociación para Maquinaria de Computación.

2. *IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Norma para sistemas y Ingeniería de software: ciclo de vida del software Procesos* , IEEE, 2008 [3].

Este estándar se considera el estándar clave con respecto a la definición de los procesos del ciclo de vida y ha sido adoptado por los dos principales estandarización cuerpos en ingeniería de software: ISO / IEC JTC1 / SC7 y el software IEEE Computer Society

Aunque la *Guía SWEBOK* no es una aplicación suave estándar de ingeniería de artículos per se, cuidado especial se tomará en todo el documento sobre La compatibilidad de la *Guía* con la actual IEEE e ISO / IEC Sistemas y Software Software Colección de estándares de neering.

4. *Ingeniería de software 2004: plan de estudios Pautas para la licenciatura Programas en Ingeniería de Software* , IEEE Sociedad y asociación de computadoras para Maquinaria de computación, 2004; http: // sitios. computer.org/ccse/SE2004Volume.pdf . [5]

Este documento describe las pautas curriculares para una licenciatura en ingeniería de software En g. La *guía SWEBOK* se identifica como "Una de las fuentes primarias" en el desarrollo de cuerpo de conocimiento que subyace a estas pautas.

5. *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010; www.computer.org/ sevocab . [6]

Page 301

La jerarquía de referencias para la terminología es *Diccionario colegiado de Merriam Webster* (11mo ed.) [7], IEEE / ISO / IEC 24765 [6], y nuevos pro-Definiciones planteadas si es necesario.

6. “Certificación y capacitación para software Profesionales ", IEEE Computer Society, 2013; www.computer.org/certification . [8]

Información sobre la certificación y asociados. desarrollo profesional productos desarrollados y ofrecido por la IEEE Computer Society para profesionales en el campo del ingeniero de software ing se puede encontrar en este sitio web. El *SWEBOK La guía* es fundamental para estos productos.

ESTILO Y DIRECTRICES TÉCNICAS

- Las descripciones de KA deben ajustarse a Plantilla de Word disponible en www.computer. org / portal / web / cscps / formatting .
- Se espera que las descripciones de KA sigan las IEEE Computer Society Style Guide (www. computer.org/portal/web/publications/ guía de estilo)
- Los archivos deben enviarse en Microsoft Word formato.
- Todas las citas de material de referencia deben ser producido usando EndNote Web como se indica en las instrucciones proporcionadas a los editores de KA. A este respecto.

OTRAS DIRECTRICES DETALLADAS

Al hacer referencia a la *Guía del software Cuerpo de conocimiento de ingeniería* , use el título " *Guía SWEBOK.* "

Para simplificar, evite las notas al pie e intente incluir su contenido en el texto principal. la bibliografía. Creemos que este enfoque permite el lector para estar mejor expuesto a la fuente y alcance de un estándar.

El texto que acompaña a las figuras y tablas. debe explicarse por sí mismo o tener suficiente relación texto. Esto aseguraría que el lector sepa qué significan las figuras y las tablas.

Para asegurarse de que alguna información en el *La guía SWEBOK* no se vuelve rápidamente obsoleta Lete y debido a su naturaleza genérica, evite nombrando directamente herramientas y productos. En cambio, intente para nombrar sus funciones.

EDICIÓN

Los editores de la *Guía SWEBOK* , así como los profesores Los editores de copias nacionales editarán las descripciones de KA. La edición incluye la edición de copias (gramática, punc. matrícula y capitalización), edición de estilo (conformidad a la guía de estilo de Computer Society), y edición de contenido (flujo, significado, claridad, directo-ness y organización). La edición final será ser un proceso colaborativo en el que los Editores de la *guía SWEBOK* y el trabajo de los editores de KA juntos para lograr un conciso, bien redactado y Descripción útil de KA.

LIBERACIÓN DE DERECHOS DE AUTOR

Todos los derechos de propiedad intelectual asociados con la *Guía SWEBOK* permanecerá con el IEEE. Los editores de KA deben firmar un formulario de liberación de derechos de autor.

También se entiende que la *Guía SWEBOK* seguirá estando disponible de forma gratuita en el dominio público en al menos un formato, proporcionado por IEEE Computer Society a través de tecnología web ogy o por otros medios.

Para obtener más información, visite www.computer.org/ copyright.htm .

. Utilice referencias explícitas a estándares, en oposición simplemente insertando números que hacen referencia a elementos en

Página 302

Referencias

[1] Project Management Institute, *una guía para el Proyecto Organismo de Gestión del Conocimiento (Guía de PMBOK (R))* , 5ª ed., Proyecto Instituto de Gestión, 2013.

[2] Software y sistemas integrados Proyecto de plan de estudios de ingeniería (iSSEc), *Graduado en Ingeniería de Software 2009 (GSwE2009): Directrices curriculares para programas de posgrado en Ingeniería de Software* , Instituto Stevens de Tecnología, 2009; www.gswe2009.org .

[3] *IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Norma para sistemas y Ingeniería de software: ciclo de vida del software Procesos, IEEE, 2008.*

[4 *] JW Moore, *La hoja de ruta hacia el software Ingeniería: una guía basada en estándares* , Wiley-IEEE Computer Society Press, 2006.

[5] Grupo de trabajo conjunto sobre planes de estudio de informática, IEEE Computer Society and Association para maquinaria informática, *software Ingeniería 2004: Directrices curriculares para programas de pregrado en Ingeniería de Software* , 2004; http: // sitios. computer.org/ccse/SE2004Volume.pdf .

[6] *Sistemas ISO / IEC / IEEE 24765: 2010 y Ingeniería de software: vocabulario* , ISO / IEC / IEEE, 2010.

[7] *Diccionario colegiado de Merriam-Webster* , 11a ed., 2003.

[8] IEEE Computer Society, "Certificación y Capacitación para profesionales del software ", 2013; www.computer.org/certification.

APÉNDICE B

IEEE Y ISO / IEC NORMAS DE APOYO EL CUERPO DE INGENIERÍA DE SOFTWARE DE CONOCIMIENTO (SWEBOK)

Algunos podrían decir que el suministro de software engi-
Los estándares de negociación superan con creces la demanda. Una
rara vez escucha una sesión informativa sobre el tema sin
sufriendo alguna broma aparentemente obligatoria que
Hay demasiados de ellos. Sin embargo, el exis-
la tenencia de estándares requiere una gran cantidad (posiblemente
infinito) espacio comercial de alternativas y reduce
ese espacio a un conjunto más pequeño de opciones, un enorme
ventaja para los usuarios. Sin embargo, todavía puede ser
difícil de elegir entre docenas de alternativas, entonces
orientación complementaria, como este apéndice, puede
Se útil. Una lista resumida de los estándares
mencionado en este apéndice aparece al final.

Para reducir el tedio en la lectura, algunas simplificaciones
En este apéndice se hacen anotaciones y resúmenes:

- ISO / IEC JTC 1 / SC 7 mantiene casi dos
  Cien normas sobre el tema. IEEE
  mantiene unos cincuenta. Las dos organizaciones
  están en el décimo año de un programa sistemático
  para coordinar e integrar sus colecciones.
  En general, este artículo se centrará en el estándar
  los colores reconocidos por ambas organizaciones
  tomando esta condición como evidencia de que
  Se ha obtenido un amplio acuerdo. Otro
  Las normas se mencionarán brevemente.
- Los estándares tienden a tener una larga y taxonómica
  títulos Si hubiera un estándar único para
  construyendo un automóvil, el indicado para su
  Camry probablemente se titularía algo
  como "Vehículo, combustión interna, cuatro-
  rueda, pasajero, sedán ". Además, el estándar moderno
  las organizaciones dards proporcionan sus estándares
  de bases de datos. Como cualquier base de datos, estos
  a veces contienen errores, particularmente para el
  títulos Entonces este artículo a menudo parafraseará el
  título del estándar o simplemente use su número.
  Al obtener un estándar de interés, el lector
  debe confiar en el número, no en el título, dado
  en este articulo. Por razones de coherencia, el
  El artículo utilizará la convención de IEEE para
  capitalización de títulos: sustantivos, pronombres,
  adjetivos, verbos, adverbios, y primero y último
  las palabras tienen una letra mayúscula inicial, a pesar de
  el hecho de que IEEE e ISO / IEC utilizan diferentes
  convenciones
- Debido a que estas normas están siendo continuas
  Aliado revisado para tener en cuenta la nueva tecnología
  gies y patrones de uso, este artículo será
  obsoleto antes de ser publicado. Por lo tanto,
  ocasionalmente discutirá estándares que
  aún no se han publicado, si es probable
  asumir una importancia significativa.
- Se omiten las marcas comerciales explícitas. Basta
  decir que IEEE coloca una marca registrada en todos sus
  designaciones de estándares.

Hay algunas otras convenciones de interés:

- Tanto en IEEE como en ISO / IEC, estándares para
  la ingeniería de *sistemas* es mantenida por el
  mismo comité que los de ingeniería de *software*
  neering. Muchas de las normas se aplican a ambos.
  Entonces, en lugar de hacer distinciones finas, esto
  El artículo tratará con ambos.
- Por otro lado, tanto S2ESC como SC 7
  (vea a continuación las descripciones de estas organizaciones
  organizaciones) son responsables de las normas
  que no califican como "ingeniería".
  Estados Unidos y muchos otros países, los servicios
  de un ingeniero con licencia son necesarios cuando un
  el producto puede afectar la seguridad pública, la salud,

y el bienestar en lugar de afectar simplemente
El bolsillo del cliente. Este apéndice
respetará esa distinción e ignorará las normas
papás que parecen ser meramente económicos en
consecuencia.

- Se supone que la documentación del usuario está desarrollada
  operado de manera similar al software. Por ejemplo,
  7) es el responsable del software y el sistema
  Tems de ingeniería. SC 7, y sus grupos de trabajo,
  se reúne dos veces al año, atrayendo delegaciones que representan
  enviar los organismos nacionales de normalización de partici
  Patear naciones. Cada nación sigue su propio pro-
  cedimientos para determinar posiciones nacionales y
  cada nación tiene la responsabilidad de determinar

Un estándar sobre el diseño del usuario la documentación se describe en el Software Design KA.

• Algunos estándares desarrollados conjuntamente son explícitos etiquetado como desarrollos conjuntos, por ejemplo, ISO / IEC / IEEE 24765. En otros casos, el estándar los padres tienen diferentes designaciones en los dos organizaciones. Ejemplos incluyen

»IEEE Std. 12207: 2008 (también conocido como ISO / IEC 12207: 2008), donde "aka" ("también conocido como ") es la abreviatura de este apéndice anotar la designación en el otro organización;

»IEEE Std. 15939: 2008 Adopción estándar ción de ISO / IEC 15939: 2007, una adopción ción por IEEE de un estándar desarrollado en ISO / IEC;

»IEEE Std. 1220: 2005 (también conocido como ISO / IEC 26702: 2007), una "vía rápida" de ISO / IEC de un estándar desarrollado en IEEE.

En cada uno de estos casos, los estándares son sustancialmente idéntico en los dos organizaciones, que difieren solo en el frente y, ocasionalmente, se agrega información material.

Una lista resumida de todos los estándares mencionados Dards se proporciona al final de este apéndice.

ISO / IEC JTC 1 / SC 7, SOFTWARE Y INGENIERÍA DE SISTEMAS

ISO / IEC JTC 1 / SC 7 es la principal fuente de normas internacionales sobre software y sistemas Ingenieria. Su nombre se forma taxonómicamente. El Comité Técnico Conjunto 1 (JTC 1) es un niño de la Organización Internacional para la Estandarización ción (ISO) y la Electrotécnica Internacional Comisión (IEC); tiene el alcance de "información tecnológica "y subdivide su trabajo entre una serie de subcomités; Subcomité 7 (SC

si se debe adoptar un estándar ISO / IEC como norma nacional

SC 7 crea tres tipos de documentos:

• Normas internacionales: los documentos contienen ing requisitos que deben cumplirse en para reclamar conformidad.

• Especificaciones técnicas (anteriormente llamadas Informes técnicos, tipo 1 y tipo 2): Docu- Publicaciones publicadas de manera preliminar mientras el trabajo continúa.

• Informes técnicos (anteriormente llamados Techni- Informes cal, tipo 3): documentos inherentemente inadecuado para ser estándares, generalmente porque son descriptivos más que prescriptivos.

La clave para recordar es que solo el La primera categoría cuenta como un estándar de consenso. El lector puede reconocer fácilmente a los demás por el sufijo TS o TR antepuesto al número de documento.

SOFTWARE Y SISTEMAS IEEE NORMAS DE INGENIERIA COMITÉ (S2ESC)

IEEE es la organización de tecnología más grande del mundo. profesionales nicos, con cerca de 400,000 miembros en más de 160 países La publicación de los estándares son realizados por los Estándares IEEE Asociación (IEEE-SA), pero los comités que redactar y patrocinar las normas están en los distintos Sociedades IEEE; S2ESC es parte del Comité IEEE Sociedad de la informática. IEEE es un creador de estándares globales porque sus estándares se usan en muchas países ent. A pesar de su miembro internacional Sin embargo, el IEEE-SA envía (aproximadamente el 50% no es de EE. UU.) rutinariamente presenta sus estándares al estadounidense Instituto Nacional de Normas (ANSI) para avalar ment como "Estándares Nacionales Americanos". Algunos Los estándares S2ESC se desarrollan dentro de S2ESC, algunos se desarrollan conjuntamente con SC 7 y otros son adoptados después de ser desarrollados por SC 7.

305 de 1189.

IEEE-SA publica tres tipos de "estándares":

• Estándares, con preponderancia del verbo. "deberá"
• Prácticas recomendadas, con un prepondedor ance del verbo "should"
• Guías, con preponderancia del verbo. "mayo."

Los tres se comparan con el estándar ISO / IEC papás IEEE-SA tiene el concepto de "Prueba- Use "estándar, que es más o menos comparable a Una especificación técnica ISO / IEC. De todos modos, eso no tiene nada comparable a una ISO / IEC Techni- informe cal; uno buscaría en otra parte de IEEE para documentos de este tipo.

LOS ESTANDARES

El resto de este artículo asigna el seleccionado estándares para áreas de conocimiento relevantes (KA) de para cualquiera de las categorías IEEE. En ISO / IEC, es un "Informe técnico" -definido como un documento heredado- totalmente inadecuado para ser un estándar. El IEEE 2004 *La guía SWEBOK* fue adoptada por ISO / IEC con: fuera de cambio. Presumiblemente, ISO / IEC adoptará Ver- Sección 3 de la *Guía SWEBOK* .

ISO / IEC TR 19759: 2005 Ingeniería de software— *Guía del cuerpo de conocimiento de ingeniería de software* (SWEBOK)

Aplica a todos los KAs

ISO / IEC 19759: 2005, una *guía para el software Cuerpo de conocimiento de ingeniería (SWEBOK)* , identifica y describe ese subconjunto del cuerpo de conocimiento que es generalmente aceptado, incluso aunque los ingenieros de software deben tener conocimiento capaz no solo en ingeniería de software, sino también por supuesto, en otras disciplinas relacionadas. SWEBOK es un término todo incluido que describe la suma

la *guía SWEBOK* . Hay una sección para cada KA. Dentro de cada sección, los estándares relevantes están enumerados, los que se aplican principalmente a KA y otros que se aplican principalmente a otros KAs pero que también están relacionados con el curso alquilar uno. Siguiendo cada estándar hay una breve suma María. En la mayoría de los casos, el resumen es una cita o paráfrasis del resumen u otra introducción material del texto de la norma.

La mayoría de los estándares se ajustan fácilmente en un KA. Algunos caben en más de uno; en esos casos, Se proporciona una referencia cruzada. Dos normas se aplican a todos los KAs, por lo que se enumeran en una categoría llamado "General". Todas las normas relacionadas con ingeniería de software asistida por computadora (CASE) herramientas y entornos se enumeran en el Software Ingeniería de Modelos y Métodos KA sección.

GENERAL

Los dos primeros estándares son tan centrales que podría insertarse en todos los KAs. Dos más son descrito en el proceso de ingeniería de software KA, pero se mencionan aquí porque proporcionan un marco útil y porque las descripciones de varios otros estándares se refieren a ellos.

ISO / IEC TR 19759 es la *guía SWEBOK* sí mismo. No es un estándar IEEE porque, al carecer verbos prescriptivos, no cumple los criterios de conocimiento dentro de la profesión de software Ingeniería.

El texto de la *Guía SWEBOK* está disponible gratuitamente. capaz en www.swebok.org/ . La adopción ISO / IEC de la *Guía* está disponible gratuitamente enhttp: // normas. iso.org/ittf/PubliclyAvailableStandards/index. html.

ISO / IEC / IEEE 24765 proporciona un vocabulario compartido ulary para la ingeniería de sistemas y software estándares de SC 7 y S2ESC.

ISO / IEC / IEEE 24765: 2010 Sistemas y software Ingeniería — Vocabulario

Aplica a todos los KAs

ISO / IEC / IEEE 24765: 2010 proporciona un común vocabulario aplicable a todos los sistemas y software trabajo de ingeniería. Estaba preparado para recoger y Apoyar la estandarización de la terminología. YO ASI/ IEC / IEEE 24765: 2010 está destinado a servir como un referencia útil para aquellos en la tecnología de la información campo de la nología y fomentar el uso de sistemas y estándares de ingeniería de software preparados por ISO y organizaciones de enlace IEEE Computer Instituto de Sociedad y Gestión de Proyectos. YO ASI/ IEC / IEEE 24765: 2010 incluye referencias a la estándares de fuente activa para cada definición para que El uso del término puede ser explorado más a fondo.

El vocabulario es descriptivo, en lugar de pre descriptivo; recoge todas las definiciones de todas las normas relevantes, así como algunas otras fuentes, en lugar de elegir entre definiciones pendientes.

El contenido del estándar 24765 es libremente accesible en línea en www.computer.org/sevocab .

Dos estándares, 12207 y 15288, proporcionan un conjunto completo de procesos para todo el ciclo de vida de un sistema o producto de software. Los dos stan- los colores están alineados para uso concurrente en un solo proyecto o en una sola organización. Son mencionado aquí porque a menudo se usan como marco para explicar o localizar el papel de otros estándares en el ciclo de vida.

IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Norma para Ingeniería de Sistemas y Software: Procesos del ciclo de vida del software

Ver Software Engineering Process KA

IEEE Std. 15288-2008 (también conocido como ISO / IEC 15288: 2008) Estándar para los sistemas y *S* OFTWARE Engineering- Procesos del ciclo de vida del sistema

Ver Software Engineering Process KA

REQUISITOS DE SOFTWARE

El estándar principal para software y sistemas.

Define la construcción de un buen requisito, proporciona atributos y características de requerimiento ments, y discute el iterativo y recursivo aplicación de procesos de requisitos a través de fuera del ciclo de vida. ISO / IEC / IEEE 29148: 2011 proporciona orientación adicional en la aplicación de ingeniería y gestión de requisitos procesos para actividades relacionadas con los requisitos en ISO / IEC 12207: 2008 e ISO / IEC 15288: 2008. Elementos de información aplicables a la ingeniería. de requisitos y su contenido están definidos. El contenido de ISO / IEC / IEEE 29148: 2011 puede ser agregado al conjunto existente de requisitos procesos relacionados del ciclo de vida definidos por ISO / IEC 12207: 2008 o ISO / IEC 15288: 2008, o puede ser Utilizado independientemente.

Un estándar multiparte ISO / IEC proporciona principios ples y métodos para "dimensionar" software basado en sus requisitos El tamaño funcional a menudo se usa en el denominador de medidas de calidad ity y productividad en el desarrollo de software. Eso también puede desempeñar un papel en la contratación de servicios acuerdos de nivel.

ISO / IEC 14143 [seis partes] Tecnología de información ogy — Medición de software — Tamaño funcional Medición

ISO / IEC 14143 describe FSM (tamaño funcional medición). Los conceptos de tamaño funcional. medición (FSM) están diseñados para superar el limitaciones de los métodos anteriores de dimensionamiento de software por

la ingeniería de requisitos es una nueva que reemplazó varios estándares IEEE existentes. Pro presenta una visión amplia de la ingeniería de requisitos a lo largo de todo el ciclo de vida.

ISO / IEC / IEEE 29148: 2011 Sistemas y software Ingeniería — Procesos del ciclo de vida— Requisitos Ingeniería

ISO / IEC / IEEE 29148: 2011 contiene disposiciones para los procesos y productos relacionados con la ingeniería neering de requisitos para sistemas y software productos y servicios a lo largo del ciclo de vida.

alejando el enfoque de medir cómo el software se implementa para medir el tamaño en términos de las funciones requeridas por el usuario.

FSM a menudo se conoce como "recuento de puntos de función-ing. "Las cuatro normas enumeradas a continuación son alternativas métodos nativos para el recuento de puntos de función: todos cumple los requisitos de ISO / IEC 14143. El método dominante, en términos de cuota de mercado, es El método IFPUG, descrito en ISO / IEC 20926. Otros métodos son variaciones destinadas a mejorar La validez del recuento en diversas circunstancias. Por ejemplo, ISO / IEC 19761 - COSMIC es

Página 307

especialmente destinado a ser utilizado en software con un componente en tiempo real.

ISO / IEC 19761: 2011 Ingeniería de software — COS-MIC: un método de medición de tamaño funcional

ISO / IEC 20926: 2009 Software y sistemas de ingeniería neering — Medición de software — Función IFPUG Método de medición de tamaño nacional

ISO / IEC 20968: 2002 Ingeniería de software — Mk II Análisis de puntos de función: prácticas de conteo Manual

ISO / IEC 24570: 2005 Ingeniería de software— Método de medición de tamaño funcional NESMA Ver-Sección 2.1 — Definiciones y pautas de conteo para la aplicación del análisis de puntos de función

A veces los requisitos se describen en natu-lenguaje ral, pero a veces se describen en anotaciones formales o semiformales. El objetivo del lenguaje de modelado unificado (UML) es proporcionar arquitectos de sistemas, ingenieros de software y desarrolladores de software con herramientas para análisis diseño e implementación de software basado sistemas, así como para modelar negocios y procesos similares Las dos partes de ISO / IEC 19505 define UML, revisión 2. La ISO más antigua / IEC 19501 es una versión anterior de UML. Ellos se mencionan aquí porque a menudo se usan para requisitos del modelo

ISO / IEC 19501: 2005 Tecnología de la información - Procesamiento distribuido abierto *:* modelado unificado Idioma (UML) Versión 1.4.2

Ver Modelos de ingeniería de software y Métodos KA

ISO / IEC 19505: 2012 [dos partes] Tecnología de la información nología - Modelo unificado de grupo de gestión de objetos-Lenguaje ing (OMG UML)

Ver Modelos de ingeniería de software y Métodos KA

DISEÑO DE SOFTWARE

El diseño de software KA incluye ambos software diseño arquitectónico (para determinar la relación se envía entre los elementos del software y se detalla diseño (para describir los elementos individuales). YO ASI/ IEC / IEEE 42010 se refiere a la descripción de arquitectura para sistemas y software.

ISO / IEC / IEEE 42010: 2011 Sistemas y software Ingeniería - Descripción de la arquitectura

ISO / IEC / IEEE 42010: 2011 aborda la creación ación, análisis y sostenimiento de la arquitectura Tures de sistemas a través del uso de la arquitectura descripciones Un modelo conceptual de arquitectura. Descripción establecida. Los contenidos requeridos de una descripción de la arquitectura se especifican. Archi-puntos de vista tecture, marcos de arquitectura y se introducen lenguajes de descripción de arquitectura para codificar convenciones y prácticas comunes de la descripción de la arquitectura. El contenido requerido de puntos de vista de arquitectura, marco de arquitectura lenguajes de descripción de obras y arquitectura está especificado. Los anexos proporcionan la motivación. y antecedentes de conceptos clave y terminología y ejemplos de aplicación de ISO / IEC / IEEE 42010: 2011.

Al igual que ISO / IEC / IEEE 42010, el siguiente estándar Dard trata el "diseño" del software como una abstracción, independiente de su representación en un documento. En consecuencia, la norma establece disposiciones sobre la descripción del diseño, en lugar del diseño sí mismo.

IEEE Std . 1016-2009 Norma para la información Tecnología - Diseño de sistemas - Diseño de software Descripciones

Esta norma describe diseños de software y establece el contenido de la información y organiza-ción de una descripción de diseño de software (SDD). Un SDD es una representación de un diseño de software para ser utilizado para registrar información de diseño y comp comunicar esa información de diseño al diseño clave

Página 308

partes interesadas Esta norma está destinada para su uso en diseñar situaciones en las que un software explícito La descripción del diseño debe estar preparada. Estas situaciones Las opciones incluyen la construcción tradicional de software actividades (cuando el diseño conduce al código) y revertir situaciones de ingeniería (cuando una descripción del diseño se recupera de una implementación existente). Este estándar se puede aplicar a comerciales, ciencias software específico o militar que se ejecuta en digital ordenadores. La aplicabilidad no está restringida por tamaño, complejidad o criticidad del software. Esta norma se puede aplicar a la descripción. de diseños de alto nivel y detallados. Esta norma Dard no prescribe metodologías específicas para diseño, gestión de configuración o calidad Garantía de seguridad. Esta norma no requiere el uso de cualquier lenguaje de diseño particular, pero establece requisitos de lishes en la selección de diseño idiomas para usar en un SDD. Este estándar puede aplicarse a la preparación de SDD capturados como documentos en papel, bases de datos automatizadas, software herramientas de desarrollo u otros medios.

Por convención, este apéndice trata los documentos de usuario mentación como parte de un sistema de software. Ahí- Por lo tanto, los diversos aspectos de la documentación del usuario su diseño, sus pruebas, etc., se asignan a diferentes KAs. El siguiente estándar trata con el diseño de documentación de usuario.

IEEE Std. 26514-2010 Adopción estándar de ISO / IEC 26514: 2008 Ingeniero de Sistemas y Software ing - Requisitos para diseñadores y desarrolladores de Documentación del usuario

Esta norma proporciona requisitos para diseño y desarrollo de software de usuario documentación como parte de los procesos del ciclo de vida. Esoen define el proceso de documentación desde la vista punto del desarrollador de documentación y también cubre la documentación del producto. Especifica el estructura, contenido y formato para documentos de usuario tation y también proporciona orientación informativa para estilo de documentación del usuario. Es independiente de la herramientas de software que pueden usarse para producir documentos mentación y se aplica tanto a la documentación impresa ción y documentación en pantalla. Mucho de esto estándar también es aplicable a la documentación del usuario para sistemas que incluyen hardware.

CONSTRUCCION DE SOFTWARE

El término "construcción de software" se refiere a la creación detallada de software significativo y funcional a través de una combinación de codificación, verificación, pruebas unitarias, pruebas de integración y depuración.

Hay pocos estándares sobre los detalles de soft- codificación de artículos. Se ha encontrado a través de (principalmente malo) la experiencia de que las convenciones de codificación no son apropiado para la estandarización porque, en la mayoría casos, el beneficio real proviene de la consistencia tendencia a aplicar una convención arbitraria en lugar de que la convención misma. Entonces, aunque la codificación las convenciones son una buena idea, generalmente se deja a la organización o al proyecto para desarrollar tales un estandar.

Sin embargo, el tema de la codificación segura tiene atrajo la atención en los últimos años porque algunos Los modismos de codificación son inseguros ante el ataque. Un informe técnico preparado por ISO / IEC JTC 1 / SC22 (lenguajes de programación) describe vul- nerabilidades en lenguajes de programación y cómo Se pueden evitar.

ISO / IEC TR 24772: 2013 Tecnología de la información - Lenguajes de programación : orientación para evitar Vulnerabilidades en lenguajes de programación a través de Selección de idioma y uso

ISO / IEC TR 24772: 2013 especifica el programa de software se deben evitar las vulnerabilidades de lenguaje gramatical en el desarrollo de sistemas donde asegurado Se requiere comportamiento para seguridad, seguridad software crítico para la empresa y crítico para la empresa. En en general, esta guía es aplicable a la soft- artículos desarrollados, revisados o mantenidos para cualquier solicitud.

Las vulnerabilidades se describen en un manual genérico. ner que es aplicable a una amplia gama de pro- gramática de idiomas. Los anexos relacionan el genérico guía para una selección de programación específica idiomas

Página 309

El informe técnico está disponible gratuitamente en http: // standards.iso.org/ittf/PubliclyAvailableStandards/ index.html.

Aquí se mencionan dos estándares porque la unidad

PRUEBAS DE SOFTWARE

Curiosamente, hay pocos estándares para las pruebas. IEEE Std. 829 es el más completo.

las pruebas a menudo se consideran como una actividad de software construcción. IEEE e ISO / IEC están cooperando en el desarrollo de un estándar conjunto de cuatro partes, 29119, que proporcionará un tratamiento integral ment de prueba y suplante IEEE Std. 1008.

IEEE Std. 1008-1987 Estándar para unidad de software Pruebas

Ver Software Testing KA

Software ISO / IEC / IEEE 29119 [cuatro partes] (Borrador) e Ingeniería de Sistemas - Pruebas de Software

Ver Software Testing KA

El siguiente estándar prevé el desarrollo. de la documentación del usuario durante un desarrollo ágil proceso de elección. Se menciona aquí porque el desarrollo ágil a veces se considera como construcción.

ISO / IEC / IEEE 26515: 2012 Sistemas y software Ingeniería - Desarrollo de documentación de usuario en un Ambiente ágil

Ver Modelos de ingeniería de software y Métodos KA

La codificación no es la única forma de crear un software producto. A menudo, el código (así como los requisitos y diseño) se reutiliza de proyectos o motores anteriores necesario para su reutilización en futuros proyectos. IEEE Std. 1517 se menciona aquí porque proporciona un común marco para extender el sistema y el software procesos del ciclo de vida de IEEE Std. 12207: 2008 a incluir la práctica sistemática de reutilización.

IEEE Std. Norma 1517-2010 para información Tecnología - Sistema y ciclo de vida del software Procesos - Procesos de reutilización

Ver Software Engineering Process KA

IEEE Std. 829-2008 Estándar para software y sistemas Documentación de prueba tem

Los procesos de prueba determinan si el desarrollo Los productos de una actividad dada se ajustan a requisitos de esa actividad y si el sistema tem y / o software satisfacen su uso previsto y Necesidades del usuario. Se especifican las tareas del proceso de prueba para diferentes niveles de integridad. Estas tareas de proceso determinar la amplitud y profundidad de Documentación de prueba. Los elementos de documentación para cada tipo de documentación de prueba puede ser seleccionado. El alcance de las pruebas abarca software-sistemas basados en software, software de computadora, hardware Ware, y sus interfaces. Esta norma aplica a los sistemas basados en software que se están desarrollando, mantenido o reutilizado (heredado, comercial off-el estante, artículos no del desarrollo). El termino "Software" también incluye firmware, microcódigo, y documentación. Los procesos de prueba pueden incluir inspección, análisis, demostración, verificación, y validación de software y software productos del sistema.

IEEE Std. 1008 se centra en pruebas unitarias.

IEEE Std. 1008 - 1987 Estándar para unidad de software Pruebas

El objetivo principal es especificar un estándar enfoque para la prueba de la unidad de software que puede ser utilizado como base para el ingeniero de software de sonido práctica ing. Un segundo objetivo es describir los conceptos y pruebas de ingeniería de software supuestos en los que se basa el enfoque estándar basado. Un tercer objetivo es proporcionar orientación e información de recursos para ayudar con la implementación mentación y uso de las pruebas unitarias estándar enfoque.

Page 310

IEEE e ISO / IEC JTC 1 / SC 7 están cooperando en un proyecto para desarrollar un solo integral estándar que cubre todos los aspectos de las pruebas. Uno puede esperar la publicación del estándar de cuatro partes para 2014. Algunas partes del contenido siguen siendo controvertidas versial Una cuestión taxonómica es si "estática métodos ", como inspección, revisión y estática análisis — debe caer dentro del alcance de “testing "o debe distinguirse como" verificación y validación ". Aunque la resolución de la el problema es probablemente de poca importancia para los usuarios de el estándar, asume una gran importancia para el redactores de normas que deben gestionar un sistema integrado conjunto de estándares de interoperación.

Software ISO / IEC / IEEE 29119 [cuatro partes] (Borrador) e Ingeniería de Sistemas - Pruebas de Software

Especifica procesos para su uso en pruebas y revisión de la documentación del usuario. No es limseguido de la fase de prueba y revisión del ciclo de vida, pero incluye actividades en toda la información gestión y gestión de documentación procesos.

Aquí se mencionan dos estándares porque algunas fuentes consideran la verificación de software y validación para ser incluida taxonómicamente en las pruebas.

IEEE Std. 1012-2012 Norma para sistema y software Verificación y validación de artículos

Ver Software Quality KA

IEEE Std. 1044-2009 Norma para la clasificación de

Anomalías de software

Ver Software Quality KA

El propósito de las pruebas de software ISO / IEC 29119 es definir un estándar internacionalmente acordado para prueba de software que puede ser utilizada por cualquier organización nización al realizar cualquier forma de software pruebas.

La prueba de la documentación del usuario se describe en el siguiente estándar, que proporciona requisitos para prueba y revisión de la documentación del usuario del software como parte de los procesos del ciclo de vida. Define el proceso de documentación desde el punto de vista de la probador de documentación y revisor. Es relevante a los roles involucrados en las pruebas y el desarrollo de software y documentación del usuario, incluido el proyecto gerentes de ect, expertos en usabilidad e información desarrolladores además de probadores y revisores.

IEEE Std. 26513-2010 Adopción estándar de ISO / IEC 26513: Ingeniero de Sistemas y Software 2009 ing - Requisitos para probadores y revisores de Documentación

ISO / IEC 26513 proporciona el requisito mínimo mentores para la prueba y revisión de documentos de usuario mentalización, tanto impresa como en pantalla documentos utilizados en el entorno laboral por el usuarios de software de sistemas. Se aplica a impreso manuales de usuario, ayuda en línea, tutoriales y referencia de usuario Documentación de referencia.

MANTENIMIENTO DEL SOFTWARE

Este estándar, el resultado de armonizar distintos Las normas IEEE e ISO / IEC sobre el tema: describe un único proceso integral para el gestión y ejecución de mantenimiento de software mericón. Se expande en las disposiciones de la soft- proceso de mantenimiento de mercancías proporcionado en ISO / IEC / IEEE 12207.

IEEE Std. 14764-2006 (también conocido como ISO / IEC 14764: 2006) Norma para la ingeniería de software: vida útil del software Procesos de ciclo: mantenimiento

ISO / IEC 14764: 2006 describe en mayor Gestión detallada del proceso de mantenimiento. descrito en ISO / IEC 12207, incluida la modificación ments. También establece definiciones para la variedad Ous tipos de mantenimiento. ISO / IEC 14764: 2006 proporciona orientación que se aplica a la planificación, ejem precaución y control, revisión y evaluación, y cierre del proceso de mantenimiento. El alcance de ISO / IEC 14764: 2006 incluye mantenimiento para múltiples productos de software con el mismo main- recursos de tenencia. "Mantenimiento" en ISO / IEC 14764: 2006 significa mantenimiento de software a menos que se indique lo contrario.

Página 311

ISO / IEC 14764: 2006 proporciona el marco dentro del cual el software genérico y específico los planes de tenencia pueden ser ejecutados, evaluados y adaptado al alcance de mantenimiento y magni tude de productos de software dados. Proporciona el marco, terminología precisa y procesos para permitir la aplicación consistente de tecnología ogy (herramientas, técnicas y métodos) al software mantenimiento.

No aborda el funcionamiento del software. y las funciones operativas, por ejemplo, copia de seguridad, recuperación y administración del sistema, que son normalmente realizado por quienes operan el software.

ISO / IEC 14764: 2006 está escrito principalmente para mantenedores de software y adicionalmente para aquellos responsable de desarrollo y aseguramiento de calidad ance. También puede ser utilizado por compradores y usuarios de sistemas que contienen software, que pueden proporcionar entradas al plan de mantenimiento.

CONFIGURACIÓN DE SOFTWARE ADMINISTRACIÓN

Hay un estándar para la configuración. administración.

IEEE Std. 828-2012 estándar para la configuración Gestión en Ingeniería de Sistemas y Software

en ISO / IEC / IEEE Std. 24765 y la información requisitos del artículo de IEEE Std. 15939.

ISO / IEC JTC 1 / SC 7 aún no ha determinado qué acción debe tomar con respecto a la nueva IEEE Std. 828. Hay cuestiones relacionadas con el grado de compatibilidad con ISO / IEC / IEEE 12207 y otros estándares en la suite SC 7. Eso Cabe señalar, sin embargo, que SC 7 no tiene Un estándar de la competencia.

INGENIERÍA DE SOFTWARE ADMINISTRACIÓN

La mayoría de los lectores interpretarán la frase "software gestión de ingeniería "para referirse a la gestión- ment de un *proyecto* que se refiere al software. Ahí son al menos dos posibles extensiones de este gen- eralización, sin embargo. Algunas actividades de software son gestionado según un acuerdo de nivel de servicio (SLA). Los SLA no cumplen con los criterios para "proyecto ect "según algunas definiciones. Además, tiene llegar a un acuerdo general de que parte de la gestión de software debe ocurrir en la organización en un nivel por encima del proyecto, para que todos los proyectos puedan beneficiarse de una inversión común. Un comúnmente ejemplo citado es la provisión de software pro ceses y herramientas por parte de la organización.

La gestión de proyectos de software puede considerarse como especialización de "gestión de proyectos" - a menudo considerado como una disciplina distinta. El proyecto

Esta norma establece el requisito mínimo
mentos para procesos de gestión de la configuración
(CM) en ingeniería de sistemas y software. los
La aplicación de esta norma se aplica a cualquier forma,
clase o tipo de software o sistema. Esta revisión
del estándar expande la versión anterior a
explicar CM, incluida la identificación y adquisición
elementos de configuración, control de cambios, informes
ing el estado de los elementos de configuración, así como
construcciones de software e ingeniería de lanzamiento. Su pre
el predecesor definió solo el contenido de un software
plan de gestion de configuracion. Esta norma
aborda qué actividades de CM deben realizarse, cuándo
van a suceder en el ciclo de vida, y qué plan
Se requieren recursos y recursos. También describe
las áreas de contenido para un plan CM. El soporte estándar
puertos ISO / IEC / IEEE 12207: 2008 e ISO / IEC /
IEEE 15288: 2008 y se adhiere a la terminología

Guía del proyecto del ect Management Institute *para el proyecto*
Cuerpo Administrativo de Conocimientos (PMBOK ®
Guía) a menudo se considera como la autoridad
fuente de este conocimiento. De vez en cuando,
IEEE adopta la versión más reciente del
Guía PMBOK ® como estándar IEEE.

IEEE Std. Guía 1490-2011 — Adopción del proyecto
ect Management Institute (PMI®) Estándar, A
Guía para el cuerpo de gestión de proyectos de Knowl-
edge (Guía de PMBOK®) — Cuarta edición

La *guía PMBOK*® identifica ese subconjunto de
el cuerpo de gestión de proyectos de conocimiento gen-
Eralmente reconocido como una buena práctica. "Generalmente
reconocido "significa el conocimiento y las prácticas
descritos son aplicables a la mayoría de los proyectos, la mayoría de

Página 312

el tiempo y hay consenso sobre su valor y
utilidad. "Buena práctica" significa que hay general
acuerdo de que la aplicación de estas habilidades, herramientas,
y las técnicas pueden aumentar las posibilidades de éxito
en una amplia gama de proyectos. Buena práctica hace
no significa que el conocimiento descrito siempre debe
ser aplicado uniformemente a todos los proyectos; la organización
El equipo de gestión de proyectos y / o proyectos responde
posible para determinar qué es apropiado para cualquier
proyecto dado La *guía PMBOK*® también proporciona
y promueve un vocabulario común dentro del
profesión de gestión de proyectos para discutir,
redacción y aplicación de la gestión de proyectos
cepts. Tal vocabulario estándar es esencial
elemento de una disciplina profesional. El proyecto
Management Institute (PMI) ve este estándar
como referencia fundamental de gestión de proyectos
por sus programas de desarrollo profesional y
certificaciones

Las revisiones de 2008 de ISO / IEC / IEEE 12207
y 15288 proporcionar proyectos de gestión de proyectos
ceses de software y sistemas y relacionarlos
a procesos a nivel de organización, así como a tecnología
procesos nicos. El 16326 desarrollado conjuntamente
estándar, reemplazando dos estándares más antiguos, se expande
esas disposiciones con orientación para su aplicación.

Sistemas ISO / IEC / IEEE 16326: 2009 y Soft-
Ingeniería de mercancías — Procesos del ciclo de vida — Proyecto
administración

ISO / IEC / IEEE 16326: 2009 proporciona normativa
especificaciones de contenido para la gestión de proyectos
planes que cubren proyectos de software y software
Proyectos de sistemas intensivos. También proporciona información detallada
discusión y asesoramiento sobre la aplicación de un conjunto de proyectos
ect procesos que son comunes tanto para el soft-
ciclo de vida de los equipos y sistemas cubiertos por ISO / IEC
12207: 2008 (IEEE Std. 12207-2008) e ISO / IEC
15288: 2008 (IEEE Std. 15288-2008), respectivamente.
La discusión y los consejos están destinados a ayudar en
la preparación del contenido normativo del proyecto
planes de manejo. ISO / IEC / IEEE 16326: 2009

Particularmente en aplicaciones de alta tecnología.
y proyectos de alta consecuencia, la gestión
El riesgo es un aspecto importante del proyecto general
ect responsabilidades de gestión. Esta norma
trata ese tema.

IEEE Std. 16085-2006 (también conocido como ISO / IEC 16085: 2006)
Norma para Ingeniería de Sistemas y Software:
Procesos del ciclo de vida del software: gestión de riesgos

ISO / IEC 16085: 2006 define un proceso para
gestión de riesgos en el ciclo de vida. Puede ser
agregado al conjunto existente de sistema y software
procesos del ciclo de vida definidos por ISO / IEC 15288 y
ISO / IEC 12207, o se puede usar de forma independiente.
ISO / IEC 16085: 2006 se puede aplicar igualmente a
sistemas y software.
El propósito de la gestión de riesgos es identificar
Identificar posibles problemas técnicos y de gestión.
antes de que ocurran para que se puedan tomar acciones que
reducir o eliminar la probabilidad y / o impacto
de estos problemas en caso de que ocurran. Es una crítica
herramienta cal para determinar continuamente la viabilidad
La flexibilidad de los planes del proyecto, para mejorar la búsqueda.
e identificación de posibles problemas que
puede afectar las actividades del ciclo de vida y la calidad y
rendimiento de productos y para mejorar el
gestión activa de proyectos.

El análisis del riesgo y la mitigación del riesgo depende
crucialmente en la medida. Este internacional
estándar proporciona una elaboración de la medida
proceso de ISO / IEC / IEEE 15288: 2008
e ISO / IEC / IEEE 12207: 2008.

IEEE Std. 15939-2008 Adopción estándar de ISO /
IEC 15939: 2007 Ingeniero de Sistemas y Software
Eng: proceso de medición

ISO / IEC 15939 define un proceso de medición
aplicable al ingeniero de sistemas y software
ing y disciplinas de gestión. El proceso es
descrito a través de un modelo que define la actividad

es el resultado de la armonización de ISO / IEC TR 16326: 1999 y IEEE Std. 1058-1998.

lazos del proceso de medición que se requieren para especificar adecuadamente qué información de medición se requiere mación, cómo las medidas y el análisis Se deben aplicar estos resultados y cómo determinar

Página 313

si los resultados del análisis son válidos La medida El proceso es flexible, adaptable y adaptable a necesidades de diferentes usuarios.

ISO / IEC 15939: 2007 identifica un proceso que admite la definición de un conjunto adecuado de medidas que abordar las necesidades de información específicas. Identifica el actividades y tareas que son necesarias para el éxito identificar, definir, seleccionar, aplicar y mejorar completamente medición dentro de un proyecto u organización general Estructura de medición nacional. También proporciona definiciones de términos de medición comúnmente utilizados dentro de las industrias de sistemas y software.

Los proyectos de software a menudo requieren el desarrollo mento de la documentación del usuario. Gestión de la proyecto, por lo tanto, incluye la gestión de la esfuerzo de documentación.

ISO / IEC / IEEE 26511: 2012 Sistemas y software Ingeniería: requisitos para gerentes de usuario Documentación

ISO / IEC / IEEE 26511: 2012 especifica procedimientos para gestionar la documentación del usuario en todo el ciclo de vida del software. Se aplica a personas u organizaciones que producen conjuntos de documentación, para aquellos que emprenden un único proyecto de documentación, y a la documentación producida internamente, también en cuanto a la documentación contratada para el servicio externo organizaciones. Proporciona una descripción general del software gestión de información y documentación de artículos procesos, y también presenta aspectos de cartera planificación y gestión de contenido que documenta el usuario los gerentes de mentores aplican. Cubre la gestión actividades al iniciar un proyecto, incluida la configuración procedimientos y especificaciones, estableciendo infraestructura y construcción de un equipo. Incluye ejemplos de roles necesarios en una documentación de usuario equipo. Aborda medidas y estimaciones necesario para el control de gestión y el uso de procesos de apoyo como la gestión del cambio, horario y control de costos, gestión de recursos, y gestión de calidad y mejora de procesos ment. Incluye requisitos para documentos clave producido para la gestión de documentación del usuario, incluyendo planes de documentación y documentación planes de manejo. ISO / IEC / IEEE 26511: 2012 es independiente de las herramientas de software que pueden usarse para producir o gestionar documentación, y aplica tanto a la documentación impresa como a la documentación en pantalla Mentación. Gran parte de su orientación es aplicable a documentación de usuario para sistemas que incluyen hard- tanto el software como el software.

A veces, los componentes del software o del sistema son adquirido en lugar de desarrollado.

IEEE Std. 1062-1998 Práctica recomendada para Adquisición de software

Un conjunto de prácticas útiles de calidad que pueden ser seleccionado y aplicado durante uno o más pasos en Se describe un proceso de adquisición de software. Esta la práctica recomendada se puede aplicar al software que se ejecuta en cualquier sistema informático independientemente de el tamaño, complejidad o criticidad del software, pero es más adecuado para su uso en modificados fuera de línea software de plataforma y software completamente desarrollado.

A veces se adquiere la documentación del usuario. independientemente de si el software que describe fue adquirida. Las siguientes ofertas estándar con patema

ISO / IEC / IEEE 26512: 2011 Sistemas y software Ingeniería: requisitos para adquirentes y proveedores alicates de documentación del usuario

ISO / IEC / IEEE 26512: 2011 fue desarrollado para ayudar a los usuarios de ISO / IEC / IEEE 15288: 2008 o ISO / IEC / IEEE 12207: 2008 para adquirir o suministrar software documentación de usuario del software como parte del software procesos del ciclo de vida. Define la documentación. proceso desde el punto de vista del adquirente y el punto de vista del proveedor. ISO / IEC / IEEE 26512: 2011 cubre los requisitos para los elementos de información utilizados en la adquisición de productos de documentación de usuario: el plan de adquisición, la especificación del documento, el estado Mención de trabajo, solicitud de propuestas y propuesta. Proporciona una descripción general de los documentos de usuario del software. procesos de gestión de la información y la mentoría que puede requerir la adquisición y el suministro de software Productos y servicios de documentación de usuario de software. Aborda la preparación de requisitos para

Página 314

documentación de usuario de software. Estos requisitos son fundamentales para la especificación de la documentación del usuario y declaración de trabajo. Incluye requisitos. para salidas de documentos primarios de la adquisición y proceso de suministro: la solicitud de propuesta y la propuesta de productos de documentación del usuario y servicios. También discute el uso de un documento plan de gestión de tation y un plan de documentos como surgen en los procesos de adquisición y suministro. ISO / IEC / IEEE 26512: 2011 es independiente de la herramientas de software que pueden usarse para producir documentos mentación y se aplica a ambos documentos impresos tation y documentación en pantalla. Gran parte de su la orientación es aplicable a la documentación del usuario para sistemas que incluyen hardware y software.

Los siguientes dos estándares se mencionan aquí porque proporcionan información utilizada en la gestión toma de decisiones.

IEEE Std. 1028-2008 Estándar para revisiones de software y auditorías

Ver Software Quality KA

IEEE Std. Norma 1061-1998 para la calidad del software Metodología Metodológica

Ver Software Quality KA

Se menciona el siguiente estándar porque incluye el rol del gerente en el desarrollo del usuario documentación en un proyecto ágil.

ISO / IEC / IEEE 26515: 2012 Sistemas y software Ingeniería: desarrollo de documentación del usuario en un Ambiente ágil

Ver Modelos de ingeniería de software y Métodos KA

PROCESO DE INGENIERÍA DE SOFTWARE

Procesos de ingeniería de software y sistemas. son fundamentales para la estandarización de esos dos disciplinas, no solo porque muchas son inter probado en la mejora de procesos, pero también porque los procesos son efectivos para la descripción de mejores prácticas Por ejemplo, uno podría pro plantear una práctica mejorada para el software requerido análisis de ments. Un tratamiento ingenuo podría relacionarse La descripción de una etapa temprana del ciclo de vida. modelo. Un enfoque superior es describir el practicar en el contexto de un proceso que puede ser aplicado en cualquier etapa del ciclo de vida. El requerimiento es necesario un proceso de análisis, por ejemplo sary para la etapa de desarrollo, para el mantenimiento, y a menudo para la jubilación, por lo que una práctica mejorada descrito en términos del análisis de requisitos El proceso puede aplicarse a cualquiera de esas etapas.

Los dos estándares clave son ISO / IEC / IEEE 12207, *Procesos del ciclo de vida del software* e ISO / IEC / IEEE 15288, *Procesos del ciclo de vida del sistema* . Los dos estándares tienen historias distintas, pero ambos fueron revisados en 2008 para alinear su pro cesses, permitiendo su uso interoperable a través de un amplio espectro de proyectos que van desde un stand componente de software solo para un sistema con neg- contenido de software ligible. Ambos están siendo revisados de nuevo con la intención de contener un idéntico lista de procesos, pero con disposiciones especializadas para las respectivas disciplinas.

IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Norma para Ingeniería de Sistemas y Software: Procesos del ciclo de vida del software

ISO / IEC 12207: 2008 establece un común marco para procesos del ciclo de vida del software, con terminología bien definida a la que se pueda hacer referencia por la industria del software.

ISO / IEC 12207: 2008 se aplica a la adquisición ción de sistemas y productos de software y servicios vicios y al suministro, desarrollo, operación, mantenimiento y eliminación de productos de software y la porción de software de un sistema, ya sea realizado interna o externamente a una organización ción Esos aspectos de la definición del sistema necesarios para proporcionar el contexto para productos de software y Los servicios están incluidos.

ISO / IEC 12207: 2008 también proporciona un proceso que puede emplearse para definir, controlar, y mejorar los procesos del ciclo de vida del software.

Los procesos, actividades y tareas de ISO / IEC 12207: 2008, ya sea solo o junto con ISO / IEC 15288: también se puede aplicar durante el adquisición de un sistema que contiene software.

Página 315

IEEE Std. 15288-2008 (también conocido como ISO / IEC 15288: 2008) Norma para Ingeniería de Sistemas y Software: Procesos del ciclo de vida del sistema

ISO / IEC 15288: 2008 establece un común marco para describir el ciclo de vida del sistema Tems creados por humanos. Define un conjunto de procesos y terminología asociada. Estas los procesos se pueden aplicar a cualquier nivel en el jerarquía de la estructura de un sistema. Conjuntos seleccionados de estos procesos se pueden aplicar en todo el ciclo de vida para gestionar y realizar el etapas del ciclo de vida de un sistema. Este es el alojamiento establecido a través de la participación de todos los interesados elementos de información (productos de información, documentoría) para ser desarrollado y revisado durante ciclos de vida y servicio de sistemas y software Procesos de gestión. Especifica el propósito y contenido de todos los sistemas y software identificados registros de datos y elementos de información del ciclo de vida, como así como registros y elementos de información para información Gestión de servicios tecnológicos. los el contenido del elemento de información se define de acuerdo con tipos de documentos genéricos (descripción, plan, polihelado, procedimiento, informe, solicitud y especificación) y el propósito específico del documento. por simplicidad de referencia, cada elemento de información los describe como si se publicara como un documento separado

fiestas, con el objetivo final de lograr clientes Tomer satisfacción.

ISO / IEC 15288: 2008 también proporciona procesos que apoyan la definición, control y mejora Mención de los procesos del ciclo de vida utilizados dentro de un organización o un proyecto. Organizaciones y los proyectos pueden usar estos procesos del ciclo de vida cuando sistemas de adquisición y suministro.

ISO / IEC 15288: 2008 se refiere a esos sistemas que son artificiales y pueden configurarse con uno o más de los siguientes: hardware, software, datos, humanos, procesos (p. ej., procesos para proinstrucciones tor), instalaciones, materiales y naturaleza rally entidades que se producen. Cuando un elemento del sistema es software, el ciclo de vida del software procesa documentos mencionado en ISO / IEC 12207: 2008 puede usarse para implementar ese elemento del sistema.

ISO / IEC 15288: 2008 e ISO / IEC 12207: 2008 están armonizados para uso concurrente en un solo proyecto o en una sola organización.

Esos dos estándares especifican que procesos puede producir elementos de información pero no escriba su contenido o formato. El siguiente estándar proporciona ayuda con eso.

ISO / IEC / IEEE 15289: 2011 Sistemas y software Ingeniería: contenido de la información del ciclo de vida Productos (Documentación)

ISO / IEC / IEEE 15289: 2011 proporciona requisitos mentos para identificar y planificar el específico documento. Sin embargo, los elementos de información pueden ser inédito pero disponible en un repositorio para refdiferencia, dividida en documentos separados o vol. umes, o combinados con otros elementos de información en un documento ISO / IEC / IEEE 15289: 2011 se basa en los procesos del ciclo de vida especificados en ISO / IEC 12207: 2008 (IEEE Std. 12207 - 2008) e ISO / IEC 15288: 2008 (IEEE Std. 15288-2008), y los procesos de gestión de servicios especificado en ISO / IEC 20000-1: 2005 e ISO / IEC 20000-2: 2005.

Las siguientes dos guías proporcionan información complementaria. información útil para aplicar 12207 y 15288.

IEEE Std. 24748.2-2012 Guía: adopción de ISO / IEC TR 24748-2: 2011 Sistemas y software de ingeniería neering — Gestión del ciclo de vida — Parte 2: Guía para la aplicación de ISO / IEC 15288 (ciclo de vida del sistema Procesos)

ISO / IEC TR 24748-2 es una guía para la aplicación ción de ISO / IEC 15288: 2008. Se dirige al sistema tem, ciclo de vida, proceso, organización, proyecto, y conceptos de adaptación, principalmente a través de referencia a ISO / IEC TR 24748-1 e ISO / IEC 15288: 2008. Luego da orientación sobre la aplicación ISO / IEC 15288: 2008 desde los aspectos de la estrategia egy, planificación, aplicación en organizaciones y Aplicación en proyectos.

IEEE Std. 24748.3-2012 Guía: adopción de ISO / IEC TR 24748-3: 2011 Sistemas y software

Página 316

Ingeniería — Gestión del ciclo de vida— Parte 3: Guía para la aplicación de ISO / IEC 12207 (Soft- procesos de ciclo de vida de Ware)

ISO / IEC TR 24748-3 es una guía para la aplicación ción de ISO / IEC 12207: 2008. Se dirige al sistema tem, ciclo de vida, proceso, organización, proyecto, y conceptos de adaptación, principalmente a través de referencia a ISO / IEC TR 24748-1 e ISO / IEC 12207: 2008. Da orientación sobre la aplicación de ISO / IEC 12207: 2008 desde los aspectos de la estrategia, planificación, aplicación en organizaciones y aplicaciones cación en proyectos.

Las normas 12207 y 15288 proporcionan pro ceses que cubren el ciclo de vida, pero no vide un modelo de ciclo de vida estándar (cascada, increentrega mental, prototipo, etc. Seleccionando un modelo de ciclo de vida apropiado para un proyecto es un preocupación principal de ISO / IEC 24748-1.

IEEE Std. 24748.1-2011 Guía — Adopción de ISO / IEC TR 24748-1: 2010 Sistemas y software de ingeniería neering — Gestión del ciclo de vida — Parte 1: Guía para la gestión del ciclo de vida

guías de aplicación actualizadas para aquellos internacionales Normas. ISO / IEC TR 24748-1 es el resultado de etapa de alineación de la armonización de ISO / IEC 12207 e ISO / IEC 15288.

La siguiente norma amplía las disposiciones de ISO / IEC / IEEE 12207 para hacer frente a sistemática reutilización de software.

IEEE Std. Norma 1517-2010 para información Tecnología: sistema y ciclo de vida del software cesses — Procesos de reutilización

Un marco común para extender el sistema. y procesos del ciclo de vida del software de IEEE Std. 12207: 2008 para incluir la práctica sistemática de reutilización se proporciona. Los procesos, actividades, y tareas a aplicar durante cada ciclo de vida proceso para permitir que un sistema y / o producto sea construido a partir de activos reutilizables se especifican. Los procesos, actividades y tareas para habilitar la identificación, construcción, mantenimiento, y la gestión de los activos suministrados también especificado.

ISO / IEC TR 24748-1 proporciona información sobre conceptos del ciclo de vida y descripciones del poses y resultados del ciclo de vida representativo etapas También ilustra el uso de un ciclo de vida. modelo para sistemas en el contexto de ISO / IEC 15288 y proporciona una ilustración correspondiente del uso de un modelo de ciclo de vida para software en el contexto de ISO / IEC 12207. ISO / IEC TR 24748-1 adicionalmente proporciona una discusión detallada y Consejos para adaptar un modelo de ciclo de vida para su uso en un Proyecto específico y entorno organizacional. Además proporciona orientación sobre el modelo del ciclo de vida uso por dominios, disciplinas y especialidades. YO ASI/ IEC TR 24748-1 ofrece una comparación detallada entre versiones anteriores y actuales de ISO / IEC 12207 e ISO / IEC 15288, así como consejos sobre transición de versiones anteriores a actuales y sobre el uso de sus guías de aplicación. La discusión Las recomendaciones y consejos están destinados a proporcionar un modelo de ence para modelos de ciclo de vida, facilitar el uso de ISO / IEC 15288 e ISO / IEC 12207 actualizados, y proporcionar un marco para el desarrollo de

IEEE Std. 1220 se ha aplicado ampliamente como proceso de ingeniería de sistemas y fue adoptado por ISO / IEC con el número 26702. Desafortunadamente, el estándar no es completamente compatible con ISO / IEC / IEEE 15288 y se está revisando para resuelve ese problema El resultado será publicado. como ISO / IEC / IEEE 24748-4.

IEEE Std. 1220-2005 (también conocido como ISO / IEC 26702: 2007) Norma para la Aplicación y Gestión de la Proceso de Ingeniería de Sistemas

ISO / IEC 26702 define las tareas interdisciplinarias que se requieren a lo largo de la vida de un sistema ciclo para transformar las necesidades del cliente, los requisitos, y restricciones en una solución del sistema. Además de- ción, especifica los requisitos para los sistemas proceso de ingeniería y su aplicación a través de toda la vida del producto. ISO / IEC 26702: 2007 se centra en actividades de ingeniería necesarias para guiar el desarrollo de productos, al tiempo que garantiza

Página 317

que el producto está diseñado adecuadamente para hacerlo asequible para producir, poseer, operar, mantener, y finalmente eliminar sin riesgo indebido salud o medio ambiente.

Desde SC 7 e IEEE han escrito tantos estándares de proceso, uno no se sorprenderá de aprender que su modelo para la descripción del proceso es registrado en un informe técnico.

IEEE Std. Guía 24774-2012: adopción de ISO / IEC TR 24474: 2010 Ingeniero de Sistemas y Software ing — Gestión del ciclo de vida — Directrices para Descripción del cess

Un creciente número de internacionales, nacionales, y los estándares de la industria describen el modo de proceso els. Estos modelos están desarrollados para una gama de propósitos que incluyen la implementación del proceso y evaluación. Los términos y descripciones utilizados en dichos modelos varían en formato, contenido y nivel de prescripción. ISO / IEC TR 24774: 2010 pres- pautas para los elementos utilizados más frecuentemente en la descripción de un proceso: el título, pose, resultados, actividades, tareas e información ít. Mientras que el propósito principal de ISO / IEC TR 24774: 2010 es fomentar la consistencia en el estándar Dard modelos de referencia de proceso, las pautas que proporciona puede aplicarse a cualquier modelo de proceso desarrollado para cualquier propósito.

Una entidad muy pequeña (VSE) es una empresa, un organización, un departamento o un proyecto que tiene Hasta 25 personas. La serie ISO / IEC 29110 “pro- archivos "estándares grandes, como ISO / IEC 12207 para software e ISO / IEC 15288 para sistemas, en los más pequeños para VSEs. ISO 29110 es aplicable a VSE que no desarrollan sistemas críticos o críticos software de cal. Los perfiles proporcionan una hoja de ruta que permite

Un VSE podría obtener un certificado ISO / IEC 29110 ficación El conjunto de informes técnicos está disponible. sin costo en el sitio web de ISO. Muchos ISO 29110 los documentos están disponibles en inglés, español, por- portugués, japonés y francés.

ISO / IEC TR 29110-5-1-2: 2011 Ingeniero de software- ing — Perfiles de ciclo de vida para entidades muy pequeñas (VSEs) —Parte 5-1-2: Gestión e ingeniería Guía: Perfil genérico Grupo: Perfil básico

ISO / IEC TR 29110-5-1-2: 2011 es aplicable a entidades muy pequeñas (VSEs). Un VSE se define como una empresa, organización, departamento o proyecto ect tener hasta 25 personas. Un conjunto de normas y guías ha sido desarrollado de acuerdo con un conjunto de Características y necesidades de las VSEs. Los guías son basado en subconjuntos de estándares apropiados ments, conocidos como perfiles VSE. El propósito de un perfil VSE es definir un subconjunto de ISO / IEC normas internacionales relevantes para las VSE ' contexto.

ISO / IEC TR 29110-5-1-2: 2011 proporciona el guía de gestión e ingeniería de lo básico Perfil de VSE aplicable a VSE que no Desarrollar software crítico. El perfil genérico grupo no implica ninguna aplicación específica dominio.

El siguiente estándar puede verse como una alternativa tive a 12207 para proyectos individuales. El 1074 estándar explica cómo definir procesos para uso en un proyecto dado. 12207 y 15288 los estándares, sin embargo, se centran en definir procesos para adopción organizacional y uso repetido en muchos proyectos El 1074 actual es la actualización de un estándar que fue un predecesor de 12207.

IEEE Std. 1074-2006 Norma para desarrollar un

una puesta en marcha para crecer paso a paso utilizando la ISO 29110 guías de gestión e ingeniería.

Conjunto de normas ISO / IEC 29110 y técnicas los informes están dirigidos por un público como VSE, clientes o auditores. ISO / IEC 29110 no es destinado a impedir el uso de vida diferente ciclos enfoques como cascada, iterativo, incremental, evolutivo o ágil.

Proceso de ciclo de vida del proyecto de software

Este estándar proporciona un proceso para crear un proceso del ciclo de vida del proyecto de software (SPLCP). Es dirigido principalmente al arquitecto de procesos para una proyecto de software dado.

Página 318

Todos los estándares descritos hasta ahora en esta sección proporcionar una base para *definir* procesos. Algunos los usuarios están interesados en *evaluar* y mejorar sus procesos después de la implementación. El 15504 la serie proporciona evaluación de procesos; es cursiendo revisado y renumerado 330xx.

ISO / IEC 15504 [diez partes] Tecnología de información ogy — Evaluación de procesos

ISO / IEC 15504-2: 2003 define los requisitos para realizar la evaluación del proceso como base para uso en la mejora y capacidad de procesos determinación.

La evaluación del proceso se basa en dos dimensiones. modelo regional que contiene una dimensión de proceso y Una dimensión de capacidad. La dimensión del proceso es proporcionado por un modelo de referencia de proceso externo (como 12207 o 15288), que define un conjunto de procesos caracterizados por declaraciones de proceso Propósito y resultados del proceso. La capacidad la dimensión consiste en un marco de medición que comprende seis niveles de capacidad de proceso y su atributos de proceso asociados.

El resultado de la evaluación consiste en un conjunto de procalificaciones de atributos de cess para cada proceso evaluado, denominado el perfil del proceso, y también puede incluir El nivel de capacidad alcanzado por ese proceso.

ISO / IEC 15504-2: 2003 identifica la medida marco de referencia para la capacidad del proceso y la requisitos para

- realizar una evaluación;
- modelos de referencia de procesos;
- modelos de evaluación de procesos;
- verificar la conformidad de la evaluación del proceso.

Los requisitos para la evaluación del proceso. definido en ISO / IEC 15504-2: 2003 forma una estructura para que

- facilita la autoevaluación;
- proporciona una base para su uso en la mejora de procesos determinación y capacidad;
- toma en cuenta el contexto en el cual se implementa el proceso evaluado;
- produce una calificación de proceso;
- aborda la capacidad del proceso para lograr su propósito;
- es aplicable en todos los dominios de aplicación y tamaños de organización; y
- puede proporcionar un punto de referencia objetivo entre organizaciones

El conjunto mínimo de requisitos definidos en ISO / IEC 15504-2: 2003 asegura esa evaluación los resultados son objetivos, imparciales, consistentes, repetidos capaz y representativo de los procesos evaluados. Los resultados de las evaluaciones del proceso conforme pueden ser comparado cuando los alcances de las evaluaciones se consideran similares; para orientación sobre esto importa, consulte ISO / IEC 15504-4.

Varias otras normas se mencionan aquí porque están escritos como elaboraciones de la procesos de 12207 o 15288. Se asignan a otros KAs porque cada uno trata temas descrito en esos otros KAs.

IEEE Std. 828-2012 estándar para la configuración Gestión en Ingeniería de Sistemas y Software

Ver Software Configuration Management KA

IEEE Std. 14764-2006 (también conocido como ISO / IEC 14764: 2006) Norma para la ingeniería de software: vida útil del software Procesos de ciclo: mantenimiento

Ver Mantenimiento de software KA

ISO / IEC 15026-4: 2012 Sistemas y software de ingeniería neering — Garantía de sistemas y software— Parte 4: Aseguramiento en el ciclo de vida

Ver Software Quality KA

IEEE Std. 15939-2008 Adopción estándar de ISO / IEC 15939: 2007 Ingeniero de Sistemas y Software ing: proceso de medición

Ver Software Engineering Management KA

ISO / IEC 15940: 2006 Tecnología de la información— Servicios de Medio Ambiente de Ingeniería de Software

Ver Modelos de ingeniería de software y Métodos KA

IEEE Std. 16085-2006 (también conocido como ISO / IEC 16085: 2006) Norma para Ingeniería de Sistemas y Software: Procesos del ciclo de vida del software: gestión de riesgos

Ver Software Engineering Management KA

Página 319

Sistemas ISO / IEC / IEEE 16326: 2009 y Soft-Ingeniería de mercancías — Procesos del ciclo de vida — Proyecto administración

Ver Software Engineering Management KA

ISO / IEC / IEEE 29148: 2011 Sistemas y software Ingeniería — Procesos del ciclo de vida— Requisitos Ingenieria

Ver los requisitos de software KA

Algunos usuarios desean estándares de proceso utilizables para operaciones de TI o gestión de servicios de TI. La serie ISO / IEC 20000 describe el servicio de TI administración. Los procesos son menos rigurosos. definido que los de los motores mencionados normas de intercambio, pero puede ser preferible para la situación acciones donde los riesgos de fracaso involucran dinero o la satisfacción del cliente en lugar de la salud pública, seguridad y bienestar. La serie ISO / IEC 20000 ahora se extienden a muchas partes. El fundamento de la serie, ISO / IEC 20000-1, se describe brevemente abajo.

ISO / IEC 20000-1: 2011 Tecnología de la información— Gestión de servicios — Parte 1: Gestión de servicios Requisitos del sistema

ISO / IEC 20000-1: 2011 es una gestión de servicios sistema (SMS) estándar. Especifica los requisitos. para que el proveedor de servicios planifique, establezca e implemente ment, operar, monitorear, revisar, mantener y mejorar un SMS Los requisitos incluyen el diseño, transición, entrega y mejora de servicios para cumplir con los requisitos de servicio acordados.

IEEE ha adoptado las dos primeras partes de la ISO / IEC 20000 series.

MODELOS DE INGENIERÍA DE SOFTWARE

Y métodos

Algunos enfoques para el uso de ingeniería de software métodos que atraviesan grandes partes de la vida ciclo, en lugar de centrarse en procesos específicos. "Programador jefe" fue un examen tradicional ple. "Desarrollo ágil" (en realidad un ejemplo de entrega incremental tradicional) es una corriente ejemplo. Ni S2ESC ni SC 7 tienen un estándar para un desarrollo ágil, pero hay un estándar para desarrollar documentación de usuario de forma ágil proyecto.

ISO / IEC / IEEE 26515: 2012 Sistemas y software Ingeniería: desarrollo de documentación del usuario en un Ambiente ágil

ISO / IEC / IEEE 26515: 2012 especifica la forma en en qué documentación de usuario se puede desarrollar Proyectos de desarrollo ágil. Está destinado para su uso en todas las organizaciones que usan desarrollo ágil o están considerando implementar su proyecto efectos usando estas técnicas. Se aplica a las personas uniones organizaciones que producen conjuntos de documentos tation, para aquellos que realicen un solo documento proyecto de documentación y documentación producida internamente, así como a la documentación contratada a organizaciones de servicio externas. ISO / IEC / IEEE 26515: 2012 aborda la relación entre el proceso de documentación del usuario y el ciclo de vida Proceso de documentación en desarrollo ágil. Eso describe cómo el desarrollador de información o proyecto ect manager puede planificar y administrar la documentación del usuario Desarrollo de la mentalidad en un entorno ágil. No tiene la intención de alentar ni disuadir envejecer el uso de cualquier desarrollo ágil particular herramientas o métodos.

Muchas metodologías se basan en semiformales. descripciones del software a construir. Estos van desde simples anotaciones descriptivas los modelos que pueden ser manipulados y probados y, en algunos casos, puede generar código. Dos rela- Técnicas totalmente antiguas comienzan la lista; el primero tiene ha sido ampliamente aplicado para procesos de modelado y flujos de trabajo

IEEE Std. 1320.1-1998 Estándar para Mod funcional Eling Language: sintaxis y semántica para IDEF0

El modelado de funciones IDEF0 está diseñado para representar envió las decisiones, acciones y actividades de un organización o sistema existente o potencial. Los gráficos IDEF0 y los textos que lo acompañan son pre sembrado de manera organizada y sistemática para ganar

comprensión, análisis de soporte, proporcionar lógica para cambios potenciales, especificar requisitos y asistencia diseño a nivel de sistema portuario y actividades de integración corbatas. IDEF0 puede usarse para modelar una amplia variedad de sistemas, compuestos de personas, máquinas, materiales, computadoras e información de todas las variedades, y estructurado por las relaciones entre ellos, tanto automatizado como no automatizado. Para nuevos sistemas tems, IDEF0 puede usarse primero para definir requisitos ments y especificar las funciones que se llevarán a cabo

Los siguientes dos estándares proporcionan dos versiones de El lenguaje UML.

ISO / IEC 19501: 2005 Tecnología de la información— Procesamiento distribuido abierto: modelado unificado Idioma (UML) Versión 1.4.2

ISO / IEC 19501 describe el modelo unificado ing Language (UML), un lenguaje gráfico para

fuera por el futuro sistema. Como base de esto arquitectura, IDEF0 puede usarse para diseñar Una implementación que cumpla con estos requisitos y realiza estas funciones. Para sistemas existentes Tems, IDEF0 se puede utilizar para analizar las funciones que el sistema realiza y para registrar los medios por el cual estos se hacen.

IEEE Std. 1320.2-1998 Norma para Conceptual Lenguaje de modelado: sintaxis y semántica para IDEF1X97 (IDEFobject)

IDEF1X 97 consta de dos modelos conceptuales idiomas El lenguaje de estilo clave admite datos / modelado de información y compatibilidad descendente ible con el estándar del gobierno de los Estados Unidos de 1993 FIPS PUB 184. El lenguaje de estilo de identidad es basado en el modelo de objetos con reglas declarativas y restricciones. El estilo de identidad IDEF1X 97 incluye construcciones para los componentes distintos pero relacionados de abstracción de objetos: interfaz, solicitudes y realización; utiliza gráficos para establecer la interfaz; y define una regla declarativa, directamente ejecutable y lenguaje de restricción para solicitudes y realización iones IDEF1X 97 soporta modelado conceptual implementación por bases de datos relacionales, extendida bases de datos relacionales, bases de datos de objetos y objetos lenguajes de programación. IDEF1X 97 es formalmente definido en términos de lógica de primer orden. Un procedimiento se proporciona mediante cualquier modelo válido IDEF1X 97 puede transformarse en una teoría equivalente en lógica de primer orden. Ese procedimiento luego se aplica a Un metamodelo de IDEF1X 97 para definir el conjunto válido de los modelos IDEF1X 97.

En los últimos años, la notación UML se ha convertido popular para modelar sistemas intensivos en software. visualizar, especificar, construir y documentar mencionando los artefactos de un sistema intensivo en software tem. El UML ofrece una forma estándar de escribir un planos del sistema, incluidas las cosas conceptuales tales como procesos de negocio y funciones del sistema así como cosas concretas como la programación declaraciones de lenguaje, esquemas de bases de datos y reus- Componentes de software capaces.

ISO / IEC 19505: 2012 [dos partes] Tecnología de la información nología — Modelo unificado de grupo de gestión de objetos- Lenguaje ing (OMG UML)

ISO / IEC 19505 define el modelado unificado Lenguaje (UML), revisión 2. El objetivo de UML es proporcionar arquitectos de sistemas, software ingenieros y desarrolladores de software con herramientas para análisis, diseño e implementación de software sistemas basados así como para modelar negocios y procesos similares.

Dos estándares más construidos sobre la base de UML para proporcionar capacidades de modelado adicionales:

ISO / IEC 19506: 2012 Tecnología de la información— Grupo de gestión de objetos dirigido por la arquitectura Modernización (ADM): descubrimiento de conocimiento Metamodelo (KDM)

ISO / IEC 19506: 2012 define un metamodelo para la representación desentido por los activos de software existentes, sus asociados iones y entornos operativos, denominados El metamodelo de descubrimiento de conocimiento (KDM). Esta es el primero de la serie de especificaciones relacionadas con Software Assurance (SwA) y arquitectura basada en actividades de modernización (ADM). KDM facilita

Página 321

proyectos que involucran sistemas de software existentes asegurando la interoperabilidad y el intercambio de datos entre herramientas proporcionadas por diferentes proveedores.

ISO / IEC 19507: 2012 Tecnología de la información— Grupo de gestión de objetos Restricción de objetos Lancalibre (OCL)

ISO / IEC 19507: 2012 define el objeto Con- Straint Language (OCL), versión 2.3.1. OCL version 2.3.1 es la versión de OCL que está alineada con UML 2.3 y MOF 2.0.

Algunas organizaciones invierten en ingeniería de software entornos de neering (VER) para ayudar en el construcción de software. Un SEE, per se, no es Un reemplazo para los procesos de sonido. Sin embargo, un SEE adecuado debe apoyar los procesos que han sido elegidos por la organización.

ISO / IEC 15940: 2006 Tecnología de la información— Servicios de Medio Ambiente de Ingeniería de Software

Dentro de la ingeniería de sistemas y software, comp herramientas de ingeniería de software asistidas por computadora (CASE) representan una parte importante de la tecnología de apoyo nologías utilizadas para desarrollar y mantener información sistemas tecnológicos. Su selección debe ser llevado a cabo con cuidadosa consideración tanto de la Requisitos técnicos y de gestión.

ISO / IEC 14102: 2008 define tanto un conjunto de procesos y un conjunto estructurado de herramientas de herramientas CASE Acterísticas para su uso en la evaluación técnica y La mejor selección de una herramienta CASE. Sigue El modelo de evaluación del producto de software definido en ISO / IEC 14598-5: 1998.

ISO / IEC 14102: 2008 adopta el modelo general de las características de calidad del producto de software y subcaracterísticas definidas en ISO / IEC 9126- 1: 2001 y los extiende cuando el software el producto es una herramienta CASE; Proporciona productos de char- características exclusivas de las herramientas CASE.

El siguiente documento proporciona orientación sobre cómo adoptar herramientas CASE, una vez seleccionadas.

ISO / IEC 15940: 2006 define la ingeniería de software servicios ambientales (VER) conceptualmente en una referencia modelo de ence que se puede adaptar a cualquier SEE para dar emparejar una o más actividades de ingeniería de software. Describe los servicios que apoyan el proceso defini como en ISO / IEC 12207 para que el conjunto de VER los servicios son compatibles con ISO / IEC 12207. ISO / IEC 15940: 2006 puede usarse como referencia general o definir un proceso de software automatizado.

La selección de herramientas para una ingeniería de software El medio ambiente es en sí mismo una tarea difícil. Dos normas Brindar ayuda. ISO / IEC 14102: 2008 define tanto un conjunto de procesos como un conjunto estructurado de herramienta de ingeniería de software asistida por computadora (CASE) características para su uso en la evaluación técnica y la selección final de una herramienta CASE.

IEEE Std. 14102-2010 Adopción estándar de ISO / IEC 14102: 2008 Tecnología de la información — Guía-línea para la evaluación y selección de herramientas CASE

IEEE Std. Guía 14471-2010: adopción de ISO / IEC TR 14471: 2007 Tecnología de la información: software Ingeniería: pautas para la adopción de CASE Herramientas

El propósito de ISO / IEC TR 14471: 2007 es proporcionar una práctica recomendada para la adopción de CASE ción Proporciona orientación para establecer pro- ceses y actividades que se deben solicitar La adopción exitosa de la tecnología CASE. El uso de ISO / IEC TR 14471: 2007 ayudará para maximizar el rendimiento y minimizar el riesgo de Invertir en tecnología CASE. Sin embargo, ISO / IEC TR 14471: 2007 no establece cumplimiento Criterios de cumplimiento. Se puede usar mejor junto con ISO / IEC 14102 para la evaluación y selección de herramientas CASE. Eso ni dicta ni aboga por un desarrollo particular estándares de procesos, procesos de software, métodos de diseño ods, metodologías, técnicas, programación idiomas o paradigmas del ciclo de vida.

Página 322

Dentro de un entorno de ingeniería de software, Es importante que las diversas herramientas interoperen. Los siguientes estándares proporcionan un esquema para interconexión.

IEEE Std. 1175.1-2002 Guía para CASE Tool Inter-conexiones: clasificación y descripción

IEEE Std. 1175.2-2006 Práctica recomendada para CASO Interconexión de herramientas: caracterización de Interconexiones

IEEE Std. 1175.3-2004 Estándar para la herramienta CASE Interconexiones: modelo de referencia para especificar Comportamiento de software

IEEE Std. 1175.4-2008 Estándar para la herramienta CASE Interconexiones: modelo de referencia para especificar Comportamiento del sistema

El propósito de esta familia de estándares es especificar ify un conjunto común de conceptos de modelado basado en los que se encuentran en las herramientas comerciales CASE para Describir el comportamiento operativo de un software. sistema. Estas normas establecen un uniforme, modelo integrado de conceptos de software relacionados con Funcionalidad de software. También proporcionan un mensaje de texto sintaxis tual para expresar las propiedades comunes (atributos y relaciones) de esos conceptos como Se han utilizado para modelar el comportamiento del software.

CALIDAD DE SOFTWARE

Un punto de vista de la calidad del software comienza con ISO 9001, *Requisitos de gestión de calidad* , lidiar con la política de calidad en toda una organización

para la aplicación de ISO 9001: 2000 a la computadora Software

ISO / IEC 90003 proporciona orientación para la organización iones en la aplicación de ISO 9001: 2000 a la adquisición, suministro, desarrollo, operación y mantenimiento de software y relacionados Servicios de apoyo. ISO / IEC 90003: 2004 no agregar o cambiar los requisitos de ISO 9001: 2000.

Las pautas provistas en ISO / IEC 90003: 2004 no están destinados a ser utilizados como evaluación criterios de ment en el sistema de gestión de calidad registro / certificación.

La aplicación de ISO / IEC 90003: 2004 es apropiado para el software que es

- parte de un contrato comercial con otro organización,
- un producto disponible para un sector de mercado,
- utilizado para apoyar los procesos de un organización,
- incrustado en un producto de hardware, o
- relacionado con servicios de software.

Algunas organizaciones pueden estar involucradas en todos las actividades anteriores; otros pueden especializarse en Independientemente de cualquiera sea la situación, la organización el sistema de gestión de calidad de la nación debe cubrir todos los aspectos (software relacionado y no software) relacionado) de la empresa.

ISO / IEC 90003: 2004 identifica los problemas que debe abordarse y es independiente de la tecnología, modelos de ciclo de vida, desarrollo procesos mentales, secuencia de actividades, y estructura organizativa utilizada por una organización ción Orientación adicional y ref. Frecuente referencias al software ISO / IEC JTC 1 / SC 7

nización La terminología de ese estándar puede no estar familiarizado con los profesionales del software, y los auditores de gestión de calidad pueden no estar familiarizados con jerga de software. El siguiente estándar describe la relación entre ISO 9001 y ISO / IEC 12207. Desafortunadamente, la versión actual sion se refiere a ediciones obsoletas de ambos; un reemplazo ment está en progreso:

IEEE Std. 90003-2008 Guía — Adopción de ISO / IEC 90003: 2004 Ingeniería de software: pautas

Se proporcionan normas de ingeniería para ayudar en la aplicación de ISO 9001: 2000: en particular la ISO / IEC 12207, ISO / IEC TR 9126, ISO / IEC 14598, ISO / IEC 15939 e ISO / IEC TR 15504.

El enfoque ISO 9001 plantea una organización proceso de gestión de calidad a nivel de nación emparejado con planificación de garantía de calidad a nivel de proyecto para lograr los objetivos organizacionales. IEEE 730 describe la planificación de calidad a nivel de proyecto. Es

Página 323

actualmente alineado con una edición obsoleta de 12207, pero se está preparando una revisión.

IEEE Std. 730-2002 Estándar para la calidad del software Planes de aseguramiento

El estándar especifica el formato y el contenido de planes de aseguramiento de la calidad del software.

Comienza otro punto de vista de la calidad del software con enumerar las características deseadas de un producto de software y medidas de selección u otro evaluaciones para determinar si el nivel deseado de características se ha logrado. La llamada SQuaRE (requisitos de calidad del producto de software y evaluación) serie de normas SC 7 cubre Este enfoque en gran detalle.

Ingeniero de software ISO / IEC 25000 a 25099 ing: Requisitos de calidad del producto de software y Evaluación (SQuaRE)

Se seleccionan algunos de los estándares SQuaRE a continuación para especial atención. El primero es el guía general de la serie.

ISO / IEC 25000: 2005 Ingeniería de software — Soft- Requisitos de calidad del producto y evaluación (SQuaRE) —Guía a SQuaRE

ISO / IEC 25000: 2005 proporciona orientación para uso de la nueva serie de normas internacionales Requisitos de calidad del producto de software denominado y evaluación (SQuaRE). El propósito de esto guía es proporcionar una visión general de SQuaRE contenidos, modelos de referencia comunes y definición iones, así como la relación entre los documentos ments, permitiendo a los usuarios de esta guía una buena infsistemas.in posición de esas normas internacionales. Esta El documento contiene una explicación de la transición proceso entre el antiguo ISO / IEC 9126 y la serie 14598 y SQuaRE, y también presenta información sobre cómo usar ISO / IEC 9126 y 14598 series en su forma anterior.

SQuaRE proporciona

- Términos y definiciones,
- modelos de referencia,
- guías
- normas para la especificación de requisitos, planificación y gestión, medición, y fines de evaluación.

El próximo estándar SQuaRE proporciona un taxón. Algunas características de calidad del software que pueden ser útil en la selección de características relevantes para un proyecto específico:

ISO / IEC 25010: 2011 Sistemas y software de ingeniería neering (sistemas y requisitos de calidad de software) mentos y evaluación (SQuaRE) —Sistema y software- Modelos de calidad de artículos

ISO / IEC 25010: 2011 define lo siguiente:

1. Un modelo de calidad en uso compuesto por cinco características (algunas de las cuales son más subdividido en subcaracterísticas) que relacionarse con el resultado de la interacción cuando un El producto se utiliza en un contexto particular de uso. Este modelo de sistema es aplicable a la empresa sistema humano-computadora completo, que incluye tanto sistemas informáticos en uso como software productos en uso.
2. Un modelo de calidad del producto compuesto por ocho características (que se subdividen aún más en subcaracterísticas) que se relacionan con estática propiedades del software y propiedad dinámica Lazos del sistema informático. El modelo es aplicable a ambos sistemas informáticos y productos de software.

Las características definidas por ambos modelos. son relevantes para todos los productos de software y empresas sistemas informáticos Las características y subchar- las características proporcionan una terminología consistente para sistema de especificación, medición y evaluación y calidad del producto de software. También proporcionan un conjunto de características de calidad contra las cuales los requisitos de calidad establecidos se pueden comparar para lo completo.

Página 324

Aunque el alcance de la calidad del producto el modelo está destinado a ser software y computadora sistemas, muchas de las características también están relacionadas evasivo a sistemas y servicios más amplios.

ISO / IEC 25012 contiene un modelo para datos de calidad ity que es complementario a este modelo.

El alcance de los modelos excluye puramente funciones propiedades nacionales, pero incluye funcional idoneidad.

El ámbito de aplicación de los modelos de calidad. incluye especificaciones de soporte y evaluación de software y sistemas informáticos intensivos en software Tems desde diferentes perspectivas por aquellos que son asociado con su adquisición, requisitos, desarrollo, uso, evaluación, apoyo, mantenimiento finanzas, aseguramiento y control de calidad y auditoría. Los modelos pueden, por ejemplo, ser utilizados por operadores, compradores, aseguramiento y control de calidad personal y evaluadores independientes, particularmente los responsables de especificar y evaluar Calidad del producto de software. Actividades durante la producción uct desarrollo que puede beneficiarse del uso de los modelos de calidad incluyen

- identificación de los requisitos de software y sistema;
- validando la exhaustividad de un definición de requisitos;
- identificación de software y diseño de sistemas objetivos;
- identificación de software y pruebas del sistema objetivos;
- identificar criterios de control de calidad como parte de seguro de calidad;
- Identificar los criterios de aceptación de un software. computadora con uso intensivo de producto y / o software sistema;
- establecer medidas de caracterización de calidad- tics en apoyo de estas actividades.

Algunos documentos de la serie SQuaRE tratan específicamente con la característica de usabilidad. los Common Industry Format (CIF) para el informe de usabilidad ing comenzó en el Instituto Nacional de Normas de EE. UU. y Tecnología (NIST) y se trasladó a ISO / IEC JTC 1 / SC 7 para fines de estandarización.

Ingeniero de software ISO / IEC 25060 a 25064 ing: Requisitos de calidad del producto de software y Evaluación (SQuaRE): formato de industria común (CIF) para usabilidad

Una familia de estándares internacionales, llamada Formatos comunes de la industria (CIF), documentos la especificación y evaluación de la usabilidad de sistemas interactivos. Proporciona un exceso general vista del marco CIF y sus contenidos, defini- iones y la relación de los elementos marco ments. Los usuarios previstos del marco son identificado, así como las situaciones en las que Se puede aplicar el marco. Los supuestos y Las restricciones del marco también se enumeran.

El contenido del marco incluye lo siguiente:

- terminología consistente y clasificación de especificación, evaluación e informes;
- una definición del tipo y alcance de los formatos y la estructura de alto nivel que se utilizará para documentando la información requerida y la Resultados de la evaluación.

La familia de estándares CIF es aplicable a productos de software y hardware utilizados para pre tareas definidas Los elementos de información están destinados para ser utilizado como parte de la documentación a nivel de sistema resultante de procesos de desarrollo como aquellos en ISO 9241-210 e ISO / IEC JTC 1 / SC 7 estándares de proceso.

La familia CIF se enfoca en documentar esos elementos necesarios para el diseño y desarrollo de sistemas utilizables, en lugar de prescribir un específico proceso. Está destinado a ser utilizado en conjunto con las normas internacionales existentes, incluidas ing ISO 9241, ISO 20282, ISO / IEC 9126 y la serie SQuaRE (ISO / IEC 25000 a ISO / IEC 25099).

La familia de estándares CIF no prescribe cualquier tipo de método, ciclo de vida o proceso.

No todos están de acuerdo con la taxonomía de características de calidad en ISO / IEC 25010. Eso el estándar tiene un factor de calidad llamado "confiabilidad" que tiene subfactores de madurez, disponibilidad, falla tolerancia y recuperabilidad. IEC TC 65, que tiene la responsabilidad de las normas sobre "confiabilidad ity ", define ese término como una empresa no cuantitativa Posibilidad de fiabilidad, mantenibilidad y mantenimiento. Apoyo financiero. Otros usan el término "confiabilidad"

Page 325

para denotar una medida definida por un matemático ecuación. El desacuerdo sobre el uso de estos palabras significa que las normas sobre el tema son inherentemente no alineado. Algunas se mencionarán a continuación, pero las palabras como las mencionadas anteriormente pueden significar diferentes cosas en diferentes estándares.

Un enfoque para lograr la calidad del software es realizar un extenso programa de verificación y validación. IEEE Std. 1012 es probablemente el estándar más ampliamente aplicado del mundo en este sub- jet. Recientemente se publicó una revisión.

IEEE Std. 982.1-2005 Norma para Diccionario de Medidas de los aspectos de confiabilidad del software

Un diccionario estándar de medidas del soft-
Aspectos de fiabilidad de la evaluación para
prediciendo la confiabilidad, mantenibilidad y
disponibilidad de cualquier sistema de software; en particular,
Se aplica a los sistemas de software de misión crítica.

IEEE Std. 1633-2008 Práctica recomendada para Confiabilidad de software

Los métodos para evaluar y predecir la confianza
capacidad del software, basada en un enfoque de ciclo de vida
a la ingeniería de confiabilidad de software, se prescriben en
Esta práctica recomendada. Proporciona información
necesario para la aplicación de la fiabilidad del software
(SR) medición de un proyecto, sienta las bases
para construir métodos consistentes, y establece
El principio básico para recopilar los datos necesarios para
Evaluar y predecir la fiabilidad del software. los
la práctica recomendada prescribe cómo puede cualquier usuario
participar en evaluaciones y predicciones de SR.

IEEE tiene un estándar general para software
calidad del producto que tiene un alcance similar al
Serie ISO / IEC 250xx descrita anteriormente. Sus
la terminología difiere de la serie ISO / IEC, pero
Es sustancialmente más compacto.

IEEE Std. Norma 1061-1998 para la calidad del software Metodología Metodológica

Una metodología para establecer requisitos de calidad
mentos e identificación, implementación, análisis,
y validar el proceso y el software del producto
Se definen métricas de calidad. La metodología
abarca todo el ciclo de vida del software.

IEEE Std. 1012-2012 Norma para sistema y software Verificación y validación de artículos

Los procesos de verificación y validación (V&V) son
usado para determinar si el producto de desarrollo
Los efectos de una actividad dada se ajustan a los requisitos
mentos de esa actividad y si el producto
satisface su uso previsto y las necesidades del usuario. V&V life
los requisitos del proceso del ciclo se especifican para diferentes
Niveles de integridad ent. El alcance de los procesos de V&V
abarca sistemas, software y hardware, y
Incluye sus interfaces. Esta norma se aplica a
sistemas, software y hardware en desarrollo,
mantenido o reutilizado [heredado, comercial fuera de la red-
estante (COTS), artículos no relacionados con el desarrollo]. El termino
el software también incluye firmware y microcódigo,
y cada uno de los términos sistema, software y hardware
La vajilla incluye documentación. Procesos V&V
incluir el análisis, evaluación, revisión, inspección
ción, evaluación y prueba de productos.

Hay otros estándares que apoyan la veri
Procesos de notificación y validación. Uno describe
técnicas para realizar revisiones y auditorías
durante un proyecto de software.

IEEE Std. 1028-2008 Estándar para revisiones de software y auditorías

Cinco tipos de revisiones y auditorías de software,
junto con los procedimientos necesarios para la ejecución
ción de cada tipo, se definen en esta norma.
Esta norma solo se refiere a las revisiones
y auditorías; procedimientos para determinar lo necesario
La ciudad de una revisión o auditoría no está definida, y el
disposición de los resultados de la revisión o auditoría
no está especificado Los tipos incluidos son gestión
revisiones, revisiones técnicas, inspecciones, caminatas
pasajes y auditorías.

Página 326

En muchos casos, una base de datos de software anoma-
mentiras se utiliza para apoyar la verificación y validación
ocupaciones. El siguiente estándar sugiere cómo
Las anomalías deben clasificarse.

IEEE Std. 1044-2009 Norma para la clasificación de Anomalías de software

Esta norma proporciona un enfoque uniforme a la
clasificación de anomalías de software, independientemente
de cuando se originan o cuando se encuentran
atado dentro del proyecto, producto o vida del sistema
ciclo. Los datos de clasificación se pueden usar para una variedad
Tipo de propósitos, incluido el análisis causal de defectos
sis, gestión de proyectos y proceso de software
mejora (por ejemplo, para reducir la probabilidad de
inserción de defectos y / o aumentar la probabilidad de
detección temprana de defectos).

uso de cada parte y el uso combinado de múltiples
partes. Cobertura de aseguramiento para un ser de servicio
operado y administrado de manera continua no es
cubierto en ISO / IEC 15026.

La segunda parte de la norma describe el
estructura de un "caso de aseguramiento", que se pretende
como argumento estructurado de que la propiedad crítica
ha sido conseguido. Es una generalización de varios
construcciones específicas de dominio como "casos de seguridad".

IEEE Std. 15026.2-2011 Adopción estándar de ISO / IEC 15026-2: 2011 Ingeniero de Sistemas y Software ing — Garantía de sistemas y software — Parte 2: Caso de aseguramiento

ISO / IEC 15026-2: 2011 es adoptado por este estándar
Dard ISO / IEC 15026-2: 2011 especifica mínimo

En algunos sistemas, una propiedad particular del el software es tan importante que requiere especial tratamiento más allá del proporcionado por un convenio Programa nacional de verificación y validación. los El término emergente para este tipo de tratamiento es "systems y garantía de software ". Los ejemplos incluyen seguridad, privacidad, alta seguridad y ultra confiabilidad. El estándar 15026 está en desarrollo para tratar con tales situaciones La primera parte de las cuatro partes estándar proporciona terminología y conceptos utilizados en las partes restantes. Primero fue escrito antes las otras partes y ahora se está revisando para comp Completo acuerdo con los demás.

IEEE Std. 15026.1-2011 Adopción estándar de prueba de uso ción de sistemas ISO / IEC TR 15026-1: 2010 y Soft- Ingeniería de Ware: Sistemas y Software Assurance — Parte 1: Conceptos y vocabulario

Este estándar de uso de prueba adopta ISO / IEC TR 15026-1: 2010, que define términos y establece presenta un conjunto extenso y organizado de conceptos y sus relaciones para software y sistemas aseguramiento, estableciendo así una base para compartir comprensión de los conceptos y principios centrales Rastreo a ISO / IEC 15026 a través de su comunidad de usuarios corbatas. Proporciona información a los usuarios del subpartes posteriores de ISO / IEC 15026, incluido el requisitos para la estructura y el contenido de un caso de garantía para mejorar la consistencia y comparabilidad de casos de aseguramiento y facilitación comunicaciones de partes interesadas, ingeniería decisiones y otros usos de los casos de aseguramiento. Un El caso de garantía incluye un reclamo de nivel superior para un propiedad de un sistema o producto (o conjunto de reclamos), argumentación sistemática con respecto a esta afirmación, y la evidencia y suposiciones explícitas que Detrás de esta argumentación. Discutiendo a través de múltiples niveles de reclamos subordinados, esta estructura argumentación asegurada conecta el reclamo de nivel superior a la evidencia y suposiciones. Garantía los casos generalmente se desarrollan para respaldar reclamos en áreas como seguridad, confiabilidad, mantenimiento habilidad, factores humanos, operabilidad y seguridad, aunque estos casos de garantía a menudo se llaman por nombres más específicos, por ejemplo, caja de seguridad o reli capacidad y capacidad de mantenimiento (R&M). ISO / IEC 15026-2: 2011 no impone requisitos en La calidad del contenido de un caso de aseguramiento y no requiere el uso de un término particular nología o representación gráfica. Del mismo modo, se no establece requisitos sobre los medios físicos implementación de los datos, que no requiere ments para redundancia o colocación.

En muchos sistemas, algunas porciones son críticas para lograr la propiedad deseada mientras que otros son solo

Página 327

incidental. Por ejemplo, el sistema de control de vuelo de un avión es crítico para la seguridad, pero el microondas horno no lo es. Convencionalmente, las diversas porciones se les asignan "niveles de criticidad" para indicar su firma importancia para el logro general de la propiedad. La tercera parte de ISO / IEC 15026 describe cómo eso está hecho. Esta parte será revisada para un mejor ajuste con El resto de la norma 15026.

ISO / IEC 15026-3: 2011 Sistemas y software de ingeniería Neering (Sistemas y Software Assurance), Parte 3: Niveles de integridad del sistema

ISO / IEC 15026-3: 2011 especifica el concepto de niveles de integridad con el nivel de integridad correspondiente requisitos que deben cumplirse para poder para mostrar el logro del nivel de integridad. Eso coloca requisitos y recomienda métodos ods para definir y usar niveles de integridad y sus requisitos de nivel de integridad, incluido el asignación de niveles de integridad a sistemas, software productos de cerámica, sus elementos y productos externos recomendados dependencias finales.

ISO / IEC 15026-3: 2011 es aplicable a los sistemas Tems y software y está destinado a ser utilizado por:

- Definidores de niveles de integridad como la industria. y organizaciones profesionales, estándares organizaciones y agencias gubernamentales;
- usuarios de niveles de integridad tales como desarrolladores y mantenedores, proveedores y adquirentes, usuarios y evaluadores de sistemas o software, y para el apoyo administrativo y técnico

TR 15026-1 proporciona información adicional y referencias para ayudar a los usuarios de ISO / IEC 15026-3: 2011.

ISO / IEC 15026-3: 2011 no requiere el uso de los casos de garantía descritos por ISO / IEC 15026-2 pero describe cómo los niveles de integridad y Los casos de garantía pueden trabajar juntos, especialmente en la definición de especificaciones para niveles de integridad o mediante el uso de niveles de integridad dentro de una parte de un caso de aseguramiento.

La parte final de 15026 proporciona adicional guía para ejecutar los procesos del ciclo de vida de 12207 y 15288 cuando un sistema o software es requerido para lograr una propiedad importante.

ISO / IEC 15026-4: 2012 Sistemas y software de ingeniería neering — Garantía de sistemas y software— Parte 4: Aseguramiento en el ciclo de vida

Esta parte de ISO / IEC 15026 brinda orientación y recomendaciones para llevar a cabo programas seleccionados ceses, actividades y tareas para sistemas y software Productos que requieren garantías de propiedades seleccionado para una atención especial, llamada crítica adecuada corbatas. Esta parte de ISO / IEC 15026 especifica un prop- Lista independiente de procesos, actividades y tareas para lograr el reclamo y mostrar el logro Mención de la reclamación. Esta parte de ISO / IEC 15026 establece los procesos, actividades, tareas, orientación, y recomendaciones en el contexto de un definido modelo de ciclo de vida y conjunto de procesos de ciclo de vida para gestión del ciclo de vida del sistema y / o software.

puerto de sistemas y / o productos de software.

Un uso importante de los niveles de integridad es suplir alicates y adquirentes en acuerdos; por ejemplo, para ayudar a garantizar la seguridad, la economía o la seguridad características de un sistema o producto entregado.

ISO / IEC 15026-3: 2011 no prescribe un conjunto específico de niveles de integridad o su integridad requisitos de nivel. Además, no previene describa la forma en que el uso del nivel de integridad es integrado con el sistema general o el software de ingeniería neering procesos del ciclo de vida.

ISO / IEC 15026-3: 2011 se puede usar solo o con otras partes de ISO / IEC 15026. Se puede usar con una variedad de riesgos técnicos y especializados Análisis y enfoques de desarrollo. ISO / IEC

El siguiente estándar trata con una propiedad: seguridad, que a menudo se identifica como crítica. Era desarrollado originalmente en cooperación con los EE. UU. industria de energía nuclear.

IEEE Std. Norma 1228-1994 para la seguridad del software Planes

Los requisitos mínimos aceptables para el Se establece el contenido de un plan de seguridad de software. Esta norma se aplica al plan de seguridad del software. utilizado para el desarrollo, adquisiciones, mantenimiento nance y retiro de software crítico para la seguridad.

Página 328

Esta norma requiere que el plan esté preparado dentro del contexto del sistema de seguridad gramo. Solo los aspectos de seguridad del software son incluido. Este estándar no contiene especiales disposiciones requeridas para el software utilizado en la distribución Sistemas usados o en procesadores paralelos.

Los tratamientos clásicos sugieren que la "verificación" trata con métodos de evaluación estática y que La "prueba" trata con el método de evaluación dinámica ods. Tratamientos recientes, incluido el borrador ISO / IEC 29119, sin embargo, están borrando esta distinción, así que los estándares de prueba se mencionan aquí.

IEEE Std. 829-2008 Estándar para software y sistemas Documentación de prueba tem

Ver Software Testing KA

IEEE Std. 1008-1987 Estándar para unidad de software Pruebas

Ver Software Testing KA

IEEE Std. 26513-2010 Adopción estándar de ISO / IEC 26513: Ingeniero de Sistemas y Software 2009 ing: Requisitos para probadores y revisores de Documentación

Ver Software Testing KA

Software ISO / IEC / IEEE 29119 [cuatro partes] (Borrador) e ingeniería de sistemas: pruebas de software

Ver Software Testing KA

INGENIERÍA DE SOFTWARE PRACTICA PROFESIONAL

IEEE es un proveedor de productos relacionados con el certificación de profesionales del software Ingenieria. El primero ya ha sido descrito, la *guía para el cuerpo de ingeniería de software de Conocimiento* . La *guía SWEBOK* ha sido adoptada por ISO / IEC como un esbozo del conocimiento que proingenieros de software profesionales deberían tener.

ISO / IEC TR 19759: 2005 Ingeniero de software- ing — Guía del cuerpo de ingeniería de software de Conocimiento (SWEBOK)

Ver general

Un estándar SC 7 proporciona un marco para comparaciones entre certificaciones de software profesionales de ingenieria. Esa norma establece que las áreas consideradas en la certificación deben ser mapeado a la *Guía SWEBOK* .

ISO / IEC 24773: 2008 Ingeniería de software — Certificación de profesionales de ingeniería de software

ISO / IEC 24773: 2008 establece un marco para comparación de esquemas para certificar software profesionales de ingenieria. Un esquema de certificación es un conjunto de requisitos de certificación para software profesionales de ingenieria. ISO / IEC 24773: 2008 especifica los elementos para los que se requiere un esquema contiene e indica para qué se debe definir cada elemento.

ISO / IEC 24773: 2008 facilitará la portabilidad bilidad de la ingeniería profesional de software certificaciones entre diferentes países u organizaciones NIZACIONES En la actualidad, diferentes países y las organizaciones han adoptado diferentes enfoques sobre el tema, que se implementan por medios de reglamentos y estatutos. La intención de ISO / IEC 24773: 2008 estará abierto a estas personas ual enfoques proporcionando un marco para expresándolos en un esquema común que puede conducir a la comprensión.

SC 7 está actualmente redactando una guía que ayudará complemento 24773.

ECONOMÍA DE INGENIERÍA DE SOFTWARE

No hay estándares asignados a este KA.

FUNDAMENTOS INFORMÁTICOS

No hay estándares asignados a este KA.

FUNDAMENTOS MATEMÁTICOS

No hay estándares asignados a este KA.

Page 329

FUNDACIONES DE INGENIERIA

No hay estándares asignados a este KA.

MANTENTE ACTUAL

Este artículo fue obsoleto en el momento en que fue redactado. Algunos lectores necesitarán saber cómo para obtener designaciones actuales y descripciones de normas Esta sección describe algunos útiles recursos

DONDE ENCONTRAR NORMAS

La lista de normas publicadas para ISO / IEC JTC 1 / SC 7 se puede encontrar en www.iso.org/iso/iso_ catalog / catalogue_tc / catalogue_tc_browse. htm? commid = 45086.

Debido a que la URL podría cambiar, los lectores podrían tiene que navegar a la lista. Comience a laswww.iso.org/ iso / store.htm , luego haga clic en "examinar estándares catálogo ", luego" navegar por TC ", luego" JTC 1 " luego "SC 7"

Encontrar la lista actual de estándares para S2ESC Es un poco más difícil. Comience en http: // normas. ieee.org/. En el cuadro de búsqueda debajo de "Buscar Stan- ", escriba" S2ESC ". Esto debería producir un lista de estándares publicados para los cuales S2ESC es responsable.

Tenga en cuenta que las bases de datos de búsqueda son compilaciones Al igual que cualquier base de datos, puede contener errores que conducen a una búsqueda incompleta resultados.

DONDE OBTENER LOS ESTÁNDARES

Algunos lectores querrán obtener estándares descrito en este artículo. Lo primero que saber es que algunas normas internacionales son disponible gratis para uso individual. La corriente lista de estándares ISO / IEC disponibles bajo estos los términos se encuentran en http://standards.iso.org/ittf/ PubliclyAvailableStandards / index.html .

Uno de los estándares disponibles públicamente es el Adopción ISO / IEC de la *Guía SWEBOK* , ISO / IEC 19759.

Las definiciones contenidas en ISO / IEC / IEEE 24765, *Vocabulario de sistemas y software* , son disponible gratuitamente en www.computer.org/sevocab .

Sin embargo, la gran mayoría de los estándares no son gratis. Las normas ISO / IEC generalmente se compran de la organización de normas nacionales de la país en el que se vive. Por ejemplo, en el EE. UU., Se pueden comprar estándares internacionales del American National Standards Institute en http://webstore.ansi.org/. Alternativamente, stan- los dards se pueden comprar directamente de ISO / IEC en www.iso.org/iso/store.htm . Se debería notar que cada nación individual es libre de establecer su propio precios, por lo que puede ser útil verificar ambas fuentes.

Los estándares IEEE pueden estar disponibles para usted gratis si su empleador o biblioteca tiene una suscripción a IEEE Xplore: http://ieeexplore.ieee.org/. Algunos Las suscripciones a Xplore proporcionan acceso solo a los resúmenes de estándares; el texto completo puede entonces ser comprado a través de Xplore. Alternativamente, estándares se puede comprar a través de la tienda de estándares IEEE en www.techstreet.com/ieeegate.html . Debería ser señaló que IEEE-SA a veces incluye estándares en grupos disponibles con un descuento sustancial.

Finalmente, el lector debe tener en cuenta que las normas que IEEE ha adoptado de las normas ISO / IEC que ISO / IEC ha "acelerado" de IEEE, y estándares que se desarrollaron o revisaron conjuntamente están disponibles de ambas fuentes. Para todos los estándares descrito en este artículo, la versión IEEE y el La versión ISO / IEC es sustancialmente idéntica. los las respectivas versiones pueden tener diferentes frontales y la materia posterior pero los cuerpos son idénticos.

DONDE VER LA GUÍA SWEBOK

La *guía SWEBOK* se publica bajo un IEEE derechos de autor. La versión actual de *SWEBOK La guía* está disponible gratuitamente para el público en www. swebok.org/. La adopción ISO / IEC de la *La guía SWEBOK* , ISO / IEC TR 19759, es una de Los estándares disponibles libremente.

Page 330

LISTA RESUMIDA DE LAS NORMAS

Número y título (enumerados en orden de número)	KA más relevante

IEEE Std. Norma 730-2002 para planes de garantía de calidad de software	Calidad SW
IEEE Std. Norma 828-2012 para la gestión de la configuración en Ingeniería de Sistemas y Software	Configuración de SW administración
IEEE Std. 829-2008 Estándar para prueba de software y sistema Documentación	Prueba de SW
IEEE Std. 982.1-2005 Norma para el Diccionario de Medidas de la Aspectos de confiabilidad del software	Calidad SW
IEEE Std. 1008-1987 Estándar para pruebas de unidad de software	Prueba de SW
IEEE Std. Norma 1012-2012 para la verificación del sistema y el software y Validación	Calidad SW
IEEE Std. 1016-2009 Norma para la tecnología de la información: sistemas Diseño: descripciones de diseño de software	Diseño SW
IEEE Std. Norma 1028-2008 para revisiones y auditorías de software	Calidad SW
IEEE Std. 1044-2009 Norma para la clasificación de software Anomalías	Calidad SW
IEEE Std. Norma 1061-1998 para métricas de calidad de software Metodología	Calidad SW
IEEE Std. 1062-1998 Práctica recomendada para la adquisición de software	Ingeniería de SW administración
IEEE Std. 1074-2006 Estándar para desarrollar una vida de proyecto de software Proceso del ciclo	Ingeniería de SW Proceso
IEEE Std. 1175.1-2002 Guía para interconexiones de herramientas CASE— Clasificación y descripción	Ingeniería de SW Modelos y métodos
IEEE Std. 1175.2-2006 Práctica recomendada para la herramienta CASE Interconexión: caracterización de interconexiones	Ingeniería de SW Modelos y métodos
IEEE Std. 1175.3-2004 Estándar para interconexiones de herramientas CASE— Modelo de referencia para especificar el comportamiento del software	Ingeniería de SW Modelos y métodos
IEEE Std. 1175.4-2008 Estándar para interconexiones de herramientas CASE— Modelo de referencia para especificar el comportamiento del sistema	Ingeniería de SW Modelos y métodos
IEEE Std. 1220-2005 (también conocido como ISO / IEC 26702: 2007) Estándar para Aplicación y gestión del proceso de ingeniería de sistemas	Ingeniería de SW Proceso
IEEE Std. Norma 1228-1994 para planes de seguridad de software	Calidad SW
IEEE Std. 1320.1-1998 Norma para el lenguaje de modelado funcional: Sintaxis y Semántica para IDEF0	Ingeniería de SW Modelos y métodos
IEEE Std. 1320.2-1998 Norma para el lenguaje de modelado conceptual: Sintaxis y semántica para IDEF1X97 (IDEFobject)	Ingeniería de SW Modelos y métodos
IEEE Std. Guía 1490-2011: adopción de la gestión del proyecto Norma del Instituto (PMI®), una guía para el organismo de gestión de proyectos de conocimiento (Guía PMBOK®) — Cuarta edición	Ingeniería de SW administración
IEEE Std. 1517-2010 Norma para tecnología de la información: sistema y procesos del ciclo de vida del software: procesos de reutilización	Ingeniería de SW Proceso

Página 331

Número y título (enumerados en orden de número)	KA más relevante
IEEE Std. 1633-2008 Práctica recomendada para la confiabilidad del software	Calidad SW
IEEE Std. 12207-2008 (también conocido como ISO / IEC 12207: 2008) Estándar para Ingeniería de sistemas y software: procesos del ciclo de vida del software	Ingeniería de SW Proceso
IEEE Std. 14102-2010 Adopción estándar de ISO / IEC 14102: 2008 Tecnología de la información: guía para la evaluación y selección de Herramientas CASE	Ingeniería de SW Modelos y métodos
ISO / CEI 14143 [seis partes] Tecnología de la información: software Medición: medición del tamaño funcional	Requisitos de SW
IEEE Std. Guía 14471-2010: adopción de ISO / IEC TR 14471: 2007 Tecnología de la información — Ingeniería del software — Directrices para Adopción de herramientas CASE	Ingeniería de SW Modelos y métodos
IEEE Std. 14764-2006 (también conocido como ISO / IEC 14764: 2006) Estándar para	Mantenimiento SW

Ingeniería de software — Procesos del ciclo de vida del software — Mantenimiento	
IEEE Std. 15026.1-2011 Adopción estándar de prueba de uso de ISO / IEC TR 15026-1: 2010 Ingeniería de sistemas y software: sistemas y Software Assurance — Parte 1: Conceptos y vocabulario	Calidad SW
IEEE Std. 15026.2-2011 Adopción estándar de ISO / IEC 15026-2: Ingeniería de sistemas y software 2011: sistemas y software Aseguramiento — Parte 2: Caso de aseguramiento	Calidad SW
ISO / IEC 15026-3 Ingeniería de sistemas y software: sistemas y Software Assurance — Parte 3: Niveles de integridad del sistema	Calidad SW
ISO / IEC 15026-4: 2012 Ingeniería de sistemas y software: sistemas y Software Assurance — Parte 4: Aseguramiento en el ciclo de vida	Calidad SW
IEEE Std. 15288-2008 (también conocido como ISO / IEC 15288: 2008) Estándar para Ingeniería de sistemas y software: procesos del ciclo de vida del sistema	Ingeniería de SW Proceso
ISO / IEC / IEEE 15289: 2011 Ingeniería de sistemas y software: Contenido de los productos de información del ciclo de vida (documentación)	Ingeniería de SW Proceso
ISO / IEC 15504 [diez partes] Tecnología de la información: proceso Evaluación	Ingeniería de SW Proceso
IEEE Std. 15939-2008 Adopción estándar de ISO / IEC 15939: 2007 Ingeniería de sistemas y software: proceso de medición	Ingeniería de SW administración
ISO / IEC 15940: 2006 Tecnología de la información — Ingeniería de software Servicios ambientales	Ingeniería de SW Modelos y métodos
IEEE Std. 16085-2006 (también conocido como ISO / IEC 16085: 2006) Estándar para Ingeniería de sistemas y software — Procesos del ciclo de vida del software— Gestión de riesgos	Ingeniería de SW administración
ISO / IEC / IEEE 16326: 2009 Ingeniería de sistemas y software — Vida Procesos de ciclo: gestión de proyectos	Ingeniería de SW administración
ISO / IEC 19501: 2005 Tecnología de la información: distribuida abierta Procesamiento: lenguaje de modelado unificado (UML) versión 1.4.2	Ingeniería de SW Modelos y métodos

Página 332

Número y título (enumerados en orden de número)	KA más relevante
ISO / IEC 19505: 2012 [dos partes] Tecnología de la información — Objeto Lenguaje de modelado unificado de grupo de administración (OMG UML)	Ingeniería de SW Modelos y métodos
ISO / IEC 19506: 2012 Tecnología de la información — Gestión de objetos Modernización dirigida por la arquitectura grupal (ADM): conocimiento Descubrimiento metamodelo (KDM)	Ingeniería de SW Modelos y métodos
ISO / IEC 19507: 2012 Tecnología de la información — Gestión de objetos Lenguaje de restricción de objetos grupales (OCL)	Ingeniería de SW Modelos y métodos
ISO / IEC TR 19759: 2005 Ingeniería de software: guía del software Cuerpo de conocimiento de ingeniería (SWEBOK)	[General]
ISO / IEC 19761: 2011 Ingeniería de software: COSMIC: un funcional Método de medida del tamaño	Requisitos de SW
ISO / IEC 20000-1: 2011 Tecnología de la información — Servicio Administración: parte 1: requisitos del sistema de administración de servicios	Ingeniería de SW Proceso
ISO / IEC 20926: 2009 Ingeniería de software y sistemas: software Medición: método de medición de tamaño funcional IFPUG	Requisitos de SW
ISO / IEC 20968: 2002 Ingeniería de software: punto de función Mk II Análisis: Manual de prácticas de conteo	Requisitos de SW
ISO / IEC 24570: 2005 Ingeniería de software: NESMA funcional Método de medición del tamaño Versión 2.1: definiciones y recuento Pautas para la aplicación del análisis de puntos de función	Requisitos de SW
IEEE Std. 24748.1-2011 Guía — Adopción de ISO / IEC TR 24748-1: 2010 Ingeniería de sistemas y software. Gestión del ciclo de vida. Parte 1: Guía para la gestión del ciclo de vida	Ingeniería de SW Proceso
IEEE Std. 24748.2-2012 Guía — Adopción de ISO / IEC TR 24748-2: 2011 Ingeniería de sistemas y software — Gestión del ciclo de vida — Parte 2: Guía para la aplicación de ISO / IEC 15288 (Ciclo de vida del sistema Procesos)	Ingeniería de SW Proceso

IEEE Std. 24748-3: Guía 2012 — Adopción de ISO / IEC TR 24748-3: 2011 Ingeniería de sistemas y software — Gestión del ciclo de vida — Parte 3: Guía para la aplicación de ISO / IEC 12207 (Ciclo de vida del software Procesos)	Ingeniería de SW Proceso
ISO / IEC / IEEE 24765: 2010 Sistemas y software Ingeniería — Vocabulario	[General]
ISO / IEC TR 24772: 2013 Tecnología de la información: programación Idiomas: orientación para evitar vulnerabilidades en la programación Idiomas a través de la selección y uso de idiomas	Construcción SW
ISO / IEC 24773: 2008 Ingeniería de software: certificación de software Profesionales de la ingeniería	Ingeniería de SW Practica profesional
IEEE Std. 24774: Guía 2012 — Adopción de ISO / IEC TR 24474: 2010 Ingeniería de sistemas y software — Gestión del ciclo de vida— Pautas para la descripción del proceso	Ingeniería de SW Proceso
ISO / IEC 25000: 2005 Ingeniería de software: calidad del producto de software Requisitos y evaluación (SQuaRE): guía para SQuaRE	Calidad SW

Página 333

Número y título (enumerados en orden de número)	**KA más relevante**
ISO / IEC 25000 a 25099 Ingeniería de software: software Requisitos y evaluación de la calidad del producto (SQuaRE)	Calidad SW
ISO / IEC 25010: 2011 Ingeniería de sistemas y software: sistemas y Requisitos y evaluación de la calidad del software (SQuaRE): sistema y modelos de calidad de software	Calidad SW
ISO / IEC 25060 a 25064 Ingeniería de software: software Requisitos y evaluación de la calidad del producto (SQuaRE): común Formato industrial (CIF) para usabilidad	Calidad SW
ISO / IEC / IEEE 26511: 2012 Ingeniería de sistemas y software: Requisitos para gerentes de documentación de usuario	Ingeniería de SW administración
ISO / IEC / IEEE 26512: 2011 Ingeniería de sistemas y software: Requisitos para adquirentes y proveedores de documentación del usuario	Ingeniería de SW administración
IEEE Std. 26513-2010 Adopción estándar de ISO / IEC 26513: 2009 Ingeniería de sistemas y software: requisitos para probadores y Revisores de documentación	Prueba de SW
IEEE Std. 26514-2010 Adopción estándar de ISO / IEC 26514: 2008 Ingeniería de sistemas y software: requisitos para diseñadores y Desarrolladores de documentación de usuario	Diseño SW
ISO / IEC / IEEE 26515: 2012 Ingeniería de sistemas y software: Desarrollo de documentación del usuario en un entorno ágil	Ingeniería de SW Modelos y métodos
ISO / IEC 29110 [varias partes] Ingeniería de software — Ciclo de vida Perfiles para entidades muy pequeñas (VSE)	Ingeniería de SW Proceso
ISO / IEC / IEEE 29119 [cuatro partes] (borrador) Software y sistemas Ingeniería: pruebas de software	Prueba de SW
ISO / IEC / IEEE 29148: 2011 Ingeniería de sistemas y software — Vida Procesos de ciclo: ingeniería de requisitos	Requisitos de SW
ISO / IEC / IEEE 42010: 2011 Ingeniería de sistemas y software: Descripción de la arquitectura	Diseño SW
IEEE Std. Guía 90003: 2008: adopción de ISO / IEC 90003: 2004 Ingeniería de software: pautas para la aplicación de ISO 9001: 2000 a software de computadora	Calidad SW

Página 334

APÉNDICE C

LISTA DE REFERENCIA CONSOLIDADA

La Lista de referencia consolidada identifica todos materiales de referencia recomendados (al nivel de número de sección) que acompañan el desglose de temas dentro de cada área de conocimiento (KA). Esta La lista de referencia consolidada es adoptada por el certificación de ingeniería de software y asociados productos de desarrollo profesional que ofrece el IEEE Computer Society. Los editores de KA utilizaron la referencia referencias asignadas a su KA por el Consolidado Lista de referencias como sus referencias recomendadas.

En conjunto, esta lista de referencia consolidada es

- Completo: cubre todo el alcance de la *SWEBOK Guide* .
- Suficiente: proporcionar suficiente información para describir el conocimiento "generalmente aceptado".
- Consistente: no proporciona contradicciones conocimiento ni prácticas conflictivas.
- Creíble: reconocido como proveedor experto tratamiento.
- Actual: tratar el tema de una manera que es acorde con la actualidad en general conocimiento aceptado
- Breve: lo más corto posible (ambos en número Ber de artículos de referencia y en la página total contar) sin fallar otros objetivos.

[1 *] JH Allen et al., *Seguridad de software Ingeniería: una guía para el proyecto Gerentes* , Addison-Wesley, 2008.

[2 *] M. Bishop, *Seguridad informática: Arte y Science* , Addison-Wesley, 2002.

[3 *] B. Boehm y R. Turner, *Equilibrio de la agilidad y disciplina: una guía para perplejos* , Addison-Wesley, 2003.

[4 *] F. Bott et al., *Problemas profesionales en Ingeniería de Software* , 3ª ed., Taylor & Francis, 2000.

[5 *] JG Brookshear, *Ciencias de la computación: un Descripción general* , décima edición, Addison-Wesley, 2008.

[6 *] D. Budgen, *Diseño de software* , 2ª ed., Addison-Wesley, 2003.

[7 *] EW Cheney y DR Kincaid, *numéricos Matemáticas e Informática* , 6ª ed., Brooks / Cole, 2007.

[8 *] P. Clements et al., *Software de documentación Arquitecturas: Vistas y más allá* , 2ª ed., Pearson Education, 2010.

[9 *] RE Fairley, *Gestión y liderazgo Proyectos de software* , computadora Wiley-IEEE Society Press, 2009.

[10 *] D. Galin, *Garantía de calidad del software: De la teoría a la implementación* , Pearson Education Limited, 2004.

[11 *] E. Gamma et al., *Patrones de diseño: Elementos de objetos orientables reutilizables Software* , 1ª ed., Addison-Wesley Profesional, 1994.

[12 *] P. Grubb y AA Takang, *Software Mantenimiento: Conceptos y práctica* , 2do. ed., World Scientific Publishing, 2003.

[13 *] AMJ Hass, *Gestión de configuración Principios y prácticas* , 1ª ed., Addison-Wesley, 2003.

Page 335

[14 *] E. Horowitz et al., *Algoritmos informáticos* , 2a ed., Silicon Press, 2007.

[15 *] IEEE CS / ACM Joint Task Force sobre Ética de Ingeniería de Software y Prácticas profesionales, "Software Código de Ética de Ingeniería y Práctica profesional (Versión 5.2), "1999; www.acm.org/serving/se/code.htm .

[16 *] *IEEE Std. 828-2012, estándar para Gestión de configuración en sistemas y Ingeniería de Software* , IEEE, 2012.

[17 *] *IEEE Std. 1028-2008, Revisiones de software y Auditorías* , IEEE, 2008.

[18 *] *ISO / IEC 14764 IEEE Std. 14764-2006, Ingeniería de software: ciclo de vida del software Procesos — Mantenimiento* , IEEE, 2006.

[19 *] SH Kan, *métricas y modelos en software Ingeniería de calidad* , 2ª ed., Addison-Wesley, 2002.

[20 *] S. McConnell, *Código completo* , 2ª ed., Microsoft Press, 2004.

[21 *] J. McGarry et al., *Software práctico Medición: información objetiva para tomadores de decisiones* , Addison-Wesley Profesional, 2001.

[22 *] SJ Mellor y MJ Balcer, *ejecutable UML: una base para el modelo Arquitectura* , 1ª ed., Addison-Wesley, 2002.

[23 *] DC Montgomery y GC Runger, *Estadística aplicada y probabilidad para Ingenieros* , 4a ed., Wiley, 2007.

[24 *] JW Moore, *La hoja de ruta hacia el software Ingeniería: una guía basada en estándares* , 1er ed., Wiley-IEEE Computer Society Press, 2006

[25 *] S. Naik y P. Tripathy, *Pruebas de software y garantía de calidad: teoría y Práctica* , Wiley-Spektrum, 2008.

[26 *] J. Nielsen, Ingeniería de usabilidad, 1ª ed., Morgan Kaufmann, 1993.

[27 *] L. Null y J. Lobur, *Los fundamentos de Organización y arquitectura de computadoras* , 2ª ed., Jones and Bartlett Publishers, 2006

[28 *] M. Page-Jones, *Fundamentos del objeto-Diseño orientado en UML* , 1ª ed., Addison-Wesley, 1999.

[29 *] K. Rosen, Matemática discreta y sus Aplicaciones, 7ma ed., McGraw-Hill, 2011.

[30 *] A. Silberschatz, PB Galvin y G. Gagne, *Conceptos del sistema operativo* , octavo ed., Wiley, 2008.

[31 *] HM Sneed, "Ofreciendo Software Mantenimiento como servicio offshore ", *Proc. IEEE Int'l Conf. Mantenimiento del software* (ICSM 08), IEEE, 2008, págs. 1-5.

[32 *] I. Sommerville, *Ingeniería de software* , noveno ed., Addison-Wesley, 2011.

[33 *] S. Tockey, rendimiento *del software: Maximizando el retorno de su software Inversión* , 1ª ed., Addison-Wesley, 2004.

[34 *] G. Voland, *Ingeniería por diseño* , 2º ed., Prentice Hall, 2003.

[35 *] KE Wiegers, *Requisitos de software* , 2do ed., Microsoft Press, 2003.

[36 *] JM Wing, "Introducción de un especificador a Métodos formales, " *Computer* , vol. 23, no. 9, 1990, págs. 8, 10–23.