Patrocinadores

Entidades Organizadoras

- AdaSpain.
- Asociación de Enseñantes Universitarios de la Informática (AENUI).
- Asociación de Técnicos Informáticos (ATI).
- Asociación Española para la Inteligencia Artificial (AEPIA).
- Asociación para la Interacción Persona-Ordenador (AIPO).
- Asociación para el Desarrollo de la Informática Educativa (ADIE).
- Ayuntamiento de Zaragoza.
- Capítulo Español de la IEEE Computational Intelligence Society.
- Comité Español de Automática (CEA).
- Conferencia de Decanos y Directores de Informática (CODDI) de las Universidades Españolas.
- Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza.
- European Society for Fuzzy Logia and Technology (EUSFLAT).
- Federación de Asociaciones de Ingenieros en Informática (AI2).
- W3C España (World Wide Web Consortium).
- Programa Nacional de Tecnologías Informáticas - Dirección General de Investigación. Ministerio de Educación y Ciencia.
- Red Española de Metaheurísticas.
- Red Española de Minería de Datos y Aprendizaje.
- Sección Española de la European Association for Computer Graphics (EUROGRAPHICS).
- Sociedad de Arquitectura y Tecnología de Computadores (SARTECO).
- Sociedad de Ingeniería del Software y Tecnologías de Desarrollo del Software (SISTEDES).
- Universidad de Zaragoza.

ISBN: 978-84-9732-595-0

CEDI 2007 XII Jornadas de Ingeniería del Software y Bases de Datos | JISBD'07 |

CEDI 2007

II CONGRESO ESPAÑOL DE INFORMÁTICA

ZARAGOZA SPAIN

AUDITORIO PALACIO DE CONGRESOS
11 AL 14 DE SEPTIEMBRE DE 2007

XII Jornadas de Ingeniería del Software y Bases de Datos

| JISBD'07 |

EDITOR
Xavier Franch

ACTAS DE LAS XII JORNADAS DE INGENIERÍA DEL SOFTWARE Y BASES DE DATOS

EDITOR

Xavier Franch

PATROCINA

InterSystems

COLABORA

THOMSON

ACTAS DE LAS XII JORNADAS DE INGENIERÍA DEL SOFTWARE Y BASES DE DATOS (JISBD'07)

Magallanes, 25; 28015 Madrid, ESPAÑA
Teléfono 91 4463350
Fax: 91 4456218
clientes@paraninfo.es

ISBN: 978-84-9732-595-0
Depósito legal: M-

Maquetación: Los Editores
Coordinación del proyecto: @LIBROTEX
Portada: Estudio Dixi
Impresión y encuademación: FER Fotocomposición, S. A.

IMPRESO EN ESPAÑA-PRINTED IN SPAIN

Comité Ejecutivo

Presidente del Comité de Programa

Xavier Franch (Universitat Politècnica de Catalunya)

Secretario de la Comisión Permanente

Mario Piattini (Universidad de Castilla-La Mancha)

Coordinadora de Tutoriales

Ana M. Moreno (Universidad Politécnica de Madrid)

Coordinador de Talleres

Vicente Pelechano (Universidad Politécnica de Valencia)

Coordinador de Demostraciones

Antonio Vallecillo (Universidad de Málaga)

Coordinador de la Sesión de Divulgación de Trabajos Relevantes ya Publicados

Oscar Díaz (Universidad del País Vasco)

Composición y Maquetación de Actas

Jordi Marco (Universitat Politècnica de Catalunya)

Organización y Relaciones con CEDI 2007

Fran J. Ruiz (Universidad de Zaragoza)
M. Elena Gómez (Universidad de Zaragoza)
Javier Tuya (Universidad de Oviedo)

Comité Organizador

Presidente del CEDI

Alberto Prieto (Universidad de Granada)

Presidente del Comité Ciéntifico

Juan J. Moreno (Universidad Politécnica de Madrid)

Presidente del Comité Organizador CEDI 2007

Victor Viñals (Universidad de Zaragoza)

Coordinador de Actividades Plenarias CEDI 2007

José Duato (Universidad Politécnica de Valencia)

Secretario del CEDI 2007

José A. Castellanos (Universidad de Zaragoza)
José A. Bañares (Universidad de Zaragoza)

Comité de Programa

Alberto Abelló, Univ. Polit. Catalunya
Silvia Abrahão, Univ. Polit. Valencia
Jesus Aguilar, Univ. Sevilla
José Aldana, Univ. Málaga
Bárbara Álvarez, Univ. Polit. Cartagena
María J. Aramburu, Univ. Jaume I
João Araújo, Univ. Nova de Lisboa
Orlando Belo, Univ. do Minho
Rafael Berlanga, Univ. Jaume I
Pere Botella, Univ. Polit. Catalunya
Nieves Brisaboa, Univ. Coruña
Isabel S. Brito, Inst. Polit. Beja
Coral Calero, Univ. Castilla-La Mancha
Carlos Canal, Univ. Málaga
José M. Cavero, Univ. Rey Juan Carlos
Matilde Celma, Univ. Polit. Valencia
Rafael Corchuelo, Univ. Sevilla
Dolors Costal, Univ. Polit. Catalunya
Yania Crespo, Univ. Valladolid
Oscar Dieste, Univ. Polit. Madrid
Javier Dolado, Univ. País Vasco
João Falcão e Cunha, Univ. Porto
Pablo de la Fuente, Univ. Valladolid
Lidia Fuentes, Univ. Málaga
Mario Gaspar da Silva, Univ. Lisboa
Marcela Genero, Univ. Castilla-La Mancha
Cristina Gómez, Univ. Polit. Catalunya
Jaime Gómez, Univ. Alicante
Alfredo Goñi, Univ. País Vasco
Juan Hernández, Univ. Extremadura
Jon Iturrioz, Univ. País Vasco
Natalia Juristo, Univ. Polit. Madrid
Patricio Letelier, Univ. Polit. Valencia
Antonia Lopes, Univ. Lisboa
Adolfo Lozano, Univ. Extremadura
Esperanza Marcos, Univ. Rey Juan Carlos
Eduardo Mena, Univ. Zaragoza
Ana Moreira, Univ. Nova de Lisboa
Juan J. Moreno, Univ. Polit. Madrid
Juan M. Murillo, Univ. Extremadura
Oscar Pastor, Univ. Polit. Valencia
Antonio Polo, Univ. Extremadura
Carme Quer, Univ. Polit. Catalunya
Celia Ramos, Univ. Algarve
Isidro Ramos, Univ. Polit. Valencia
José Riquelme, Univ. Sevilla
Antonio Rito, Univ. Técnica de Lisboa
Antonio Ruíz, Univ. Sevilla
Francisco Ruíz, Univ. Castilla-La Mancha
José Samos, Univ. Granada
Fernando Sánchez, Univ. Extremadura
Juan Sánchez, Univ. Polit. Valencia
Ernest Teniente, Univ. Polit. Catalunya
Miguel Toro, Univ. Sevilla
Ambrosio Toval, Univ. Murcia
Juan C. Trujillo, Univ. Alicante
Javier Tuya, Univ. Oviedo
Belén Vela, Univ. Rey Juan Carlos
Cristina Vicente, Univ. Polit. Cartagena

Comité Asesor para la Selección de Trabajos de Prestigio

Oscar Díaz (Presidente), Univ. País Vasco
Alan Davis, Univ. of Colorado
Neil A.M. Maiden, City Univ. London
Timos Sellis, Nat. Technical Univ. Athens

Revisores Adicionales

César J. Acuña
Amaia Aguirregoitia
Diego Alonso
David Benavides
Jordi Cabot
Paloma Cáceres
Javier Cámara
Dante Carrizo
Pedro J. Clemente
Jose M. Conejero
Javier Cubo
Norberto Díaz
Amador Durán
Sergio España
Mauricio Espinoza
Ismael Etxeberria
Antonio Fariña
Raul Fernandez
L. Fredlund
Antonielly Garcia
Antonio Cesar Gómez
Ángel Herranz
Sergio Ilarri
Miguel Ángel Laguna
Maria Lencastre
Marta López
Francisco Javier Lucas
María Esperanza Manso
Julio Mariño
José Manuel Marqués
Francisco Martínez
Jorge Martínez
Miguel Ángel Martínez
Fernando Molina
Ana M. Moreno
Elena Navarro
Ismael Navas
Isabel Nepomuceno
Juan A. Nepomuceno
Joaquín Nicolás
Guadalupe Ortiz
Juan Angel Pastor
Joaquin Peña
Jenifer Pérez
Juan Manuel Pérez
Beatriz Pontes
Álvaro Prieto
Antonia M. Reina
Domingo Savio Rodríguez
Roberto Rodríguez
Oscar Romero
Fran J. Ruiz
Angeles Saavedra
Gwen Salaün
Pedro Sánchez
André L. Santos
Diego Seco
Jesús Serrano
Encarna Sosa
Toufik Taibi
Raquel Trillo
José Antonio Troyano
Juan Manuel Vara

Sistema Automático de Revisión

Quercus Software Engineering Group

Jose Javier Berrocal Universidad de Extremadura

Conferencia auspiciada por

Prólogo

Respondiendo a su cita anual, las XII Jornadas de Ingeniería del Software y Bases de Datos (JISBD) se han celebrado en Zaragoza, entre el 11 y el 14 de septiembre de 2007. Las Jornadas representan un punto de encuentro de la comunidad investigadora en ingeniería del software y en bases de datos. En sus inicios se celebraron dos eventos diferenciados, las Jornadas de Ingeniería del Software y las Jornadas sobre Investigación y Docencia en Bases de Datos. Posteriormente, en 1999, ambos eventos se unificaron en uno solo, reflejando la interrelación existente entre estas disciplinas. En esta duodécima edición, las Jornadas han constituido, una vez más, un punto de encuentro en el que profesionales y académicos de España, Portugal y Latinoamérica, de ambos campos, han podido compartir experiencias y resultados entre distintos grupos de investigación, desarrollo e innovación tecnológica.

Actualmente, JISBD es un evento auspiciado por Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software (SISTEDES, http://www.sistedes.org). Entre los fines de dicha organización destacan el de promover la investigación, la innovación y la transferencia de tecnología entre los distintos agentes involucrados en el avance las tecnologías del Software y el de fomentar actividades con otras asociaciones nacionales e internacionales con fines similares, consiguiendo así proporcionar una mayor visibilidad a la investigación de sus asociados.

Al igual que en 2005, las XII Jornadas de Ingeniería del Software y Bases de Datos se han realizado en el marco del II Congreso Español de Informática (CEDI 2007). Esto ha permitido a los participantes de las Jornadas participar en las diversas actividades de CEDI de interés para toda la comunidad de investigación en Informática, tales como conferencias invitadas y mesas redondas. La celebración cada dos años de JISBD en el marco de CEDI encaja con los objetivos citados de dicha organización.

Este volumen recoge los trabajos seleccionados por el Comité de Programa de JISBD'07. Se recibieron un total de 87 contribuciones de 9 países: España, Portugal, Argentina, Brasil, Chile, Colombia, Cuba, México y Venezuela. Cada contribución fue revisada por tres miembros del Comité de Programa. Posteriormente, se abrió una fase de discusión en la que se debatieron en mayor profundidad algunos trabajos y eventualmente se pidieron revisiones adicionales para ellos; asimismo, algunos trabajos se aceptaron condicionalmente, pendientes de verificar que la versión definitiva trataba adecuadamente los comentarios de los revisores; gracias al esfuerzo de los autores, todos estos trabajos fueron finalmente aceptados. Como resultado de todo el proceso, se configuró un programa compuesto por 30 artículos. Adicionalmente, se seleccionaron 5 trabajos más para su presentación como artículos cortos. Además, en esta edición de JISBD se recogió la posibilidad de presentar trabajos ya publicados en foros de prestigio reconocido. Se seleccionaron 4 artículos de esta modalidad. Finalmente, destacamos la celebración de una sesión para la presentación de herramientas, cuya convocatoria tuvo una acogida excelente por parte de la comunidad de JISBD, de manera que en dicha sesión se programaron un total de 19 demostraciones de herramientas.

El día previo a la conferencia, se organizaron un total de 7 talleres y un tutorial. Estos eventos están ganando importancia a cada nueva edición de JISBD y en el caso de los talleres, están creando sus propias comunidades con intereses más específicos. Algunos talleres ya están plenamente consolidados y llegan a acumular hasta un total de 8 ediciones. Cabe destacar que a partir de este año, las actas de los talleres se recogen en una publicación única en formato electrónico, con el soporte de SISTEDES, para potenciar la difusión de los trabajos presentados.

En referencia al programa, mencionar la participación de dos conferenciantes invitados de reconocido prestigio, siguiendo la pauta de ediciones anteriores. La primera conferencia impartida por Stephen Mellor, miembro del Object Management Group, y con un largo historial en la formulación de métodos para el análisis orientado a objetos. La segunda conferencia a cargo del profesor

John Mylopoulos, que posee igualmente una dilatada experiencia en diversos ámbitos de la ingeniería del software. La presencia de estos dos investigadores representó un elemento importante en el programa de las Jornadas.

Quisiera destacar un hecho que no por obvio, deja de ser merecedor de mención. La celebración de un evento de las características de JISBD, con una participación cada vez más numerosa y consolidada, y con unas exigencias de calidad que se van incrementando en cada edición, no podría realizarse sin la dedicación totalmente desinteresada de un gran número de personas. Desde el punto de vista científico, el trabajo en equipo desarrollado por los miembros del Comité Ejecutivo, en cuyo seno se han debatido los temas más candentes en la configuración de la oferta científica del congreso; y por supuesto la ardua y puntual labor de revisión efectuada por los miembros del Comité de Programa y los revisores adicionales. Desde el punto de vista organizativo, destacar la gran dedicación de los miembros del Comité Ejecutivo responsables de las tareas de enlace con CEDI, y la labor del Grupo Quercus de Ingeniería del Software de la Universidad de Extremadura, quienes han estado a cargo de todo el sistema de recepción y revisión de artículos. También deseo agradecer el soporte recibido por las entidades patrocinadoras y colaboradoras, y en especial la labor de respaldo de SISTEDES, tanto por lo que se refiere a apoyo logístico como a tareas de difusión, como ya se ha comentado. Y por último, especialmente, a los autores de los trabajos enviados a JISBD'07, en definitiva son ellos los que hacen posible la celebración del evento.

Finalmente, desear que el volumen que ahora tienes en tus manos, y que refleja el estado del arte en la investigación en Ingeniería del Software y Bases de Datos en la comunidad de habla hispana y portuguesa, sea de utilidad para tu trabajo.

Zaragoza, Septiembre 2007
Xavier Franch (editor)

Índice

CONFERENCIAS INVITADAS

TUTORIAL

TRABAJOS RELEVANTES YA PUBLICADOS

DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS

ALMACENES Y MINERÍA DE DATOS

PRUEBAS DEL SOFTWARE

TECNOLOGÍAS DE BASES DE DATOS

LÍNEAS DE PRODUCTO. ORIENTACIÓN A ASPECTOS

REQUISITOS. METAMODELADO EN MEDICIÓN

MODELIZACIÓN CONCEPTUAL DE DATOS

ARQUITECTURAS SOFTWARE

MODELOS DE CALIDAD

PROCESOS

ARTÍCULOS CORTOS

DEMOSTRACIONES

Técnica de Mejora del Mantenimiento Software Basada en Valor

Daniel Cabrero
Dirección General de Tráfico
Ministerio del Interior
dani_mas_d@yahoo.fr

Javier Garzás
Kybele Consulting S.L.
javierg@gmail.com

Mario Piattini
Grupo de Investigación ALARCOS
Univ. Castilla la Mancha
Mario.Piattini@uclm.es

Resumen

Sabemos que gran parte del coste asociado al mantenimiento del software puede ser optimizado a través de un diseño flexible y estable frente a futuras modificaciones. Cuando los ingenieros del software realizan decisiones de diseño, una gran cantidad de dinero y oportunidades de negocio están en juego.

A pesar de esto, hay una gran falta de sistematización del proceso de decisión. Más aún. No está claro cómo cuantificar dichas decisiones en términos de coste. Aunque son muchas las heurísticas, principios o patrones propuestos para mejorar la viabilidad de los diseños, no existe ninguna aproximación sistemática que defina el valor de mantenimiento aportado por cada una de las decisiones de diseño.

El presente trabajo presenta una nueva línea de investigación que tiene por objetivo optimizar, desde el punto de vista del valor, el mantenimiento implicado en cada decisión de diseño. En esta línea, se presentará una técnica sistemática de estimación del valor aportado y cuantificación en términos de coste de la mantenibilidad de las distintas opciones de diseño.

1. Motivación

"*Necesitamos mejores maneras de analizar un diseño software y predecir el valor que su implementación ofrecerá a los clientes y desarrolladores*" [1]. Así comenzaba Mary Shaw su presentación inaugural de la conferencia internacional IEEE EQUITY 2007, que ponía especial énfasis en cómo lograr que los sistemas software aporten un valor real a los intereses de los usuarios implicados.

Recientemente, Boehm [2] introdujo una agenda que pretende integrar las consideraciones de "valor" dentro del conjunto de principios y prácticas existentes en la ingeniería del software. Uno de los elementos principales de esa agenda es la "Arquitectura, diseño y desarrollo basado en valor".

Existen pocas propuestas sobre la materia, seguramente como consecuencia del poco tiempo transcurrido desde la identificación de esta nueva línea de investigación. En concreto, cabe señalar que no existe ninguna propuesta encaminada a cuantificar el mantenimiento del diseño de un sistema. Esta carencia es especialmente sensible si tenemos en cuenta que "*el mantenimiento consume la mayor parte de recursos del ciclo de vida de las aplicaciones*" [3, 4].

Según Parnas, "*la Ingeniería del Software trata de construir aplicaciones multiversión*" [5], o lo que es lo mismo, aplicaciones mantenibles. Nuestro enfoque parte de esa base para estudiar el valor de diseño en función de los costes de mantenimiento asociados al mismo. De esta forma, se pretende priorizar las decisiones de diseño de las aplicaciones.

Este artículo se estructura de la siguiente manera. La sección 2 introduce el concepto de valor desde el punto de vista del mantenimiento software. Las secciones 3 y 4 presentan un nuevo enfoque de diseño que identifica y cuantifica el mantenimiento de una decisión de diseño. Finalmente, la sección 5 resume las conclusiones del trabajo y describe el trabajo futuro pendiente.

2. El concepto de valor en mantenimiento

"Valor" es un concepto muy extenso y con muchas posibles interpretaciones. Para acotar el significado de "valor" aplicado al mantenimiento, comenzaremos ilustrando un ejemplo de distintas alternativas de diseño. Posteriormente se identificarán los aspectos que más valor han aportado a la solución.

2.1. Ejemplo de diseños alternativos

A modo de ilustración del problema que se está tratando, se procederá a diseñar una parte del

Actas de las XII Jornadas de Ingeniería del Software y Bases de Datos, JISBD'2007, pp.317-325 ISBN: 978-84-9732-595-0

modelo de una compañía aseguradora. En este sentido, se presentarán tres alternativas distintas con el objetivo de observar la problemática que hay detrás del mantenimiento basado en valor.

Se trata de modelar un vendedor que vende seguros de automóvil. La opción más intuitiva y simple en la que puede pensar el diseñador es modelar los conceptos del modelo de análisis como dos clases. De esta forma tendríamos la clase "*Vendedor*" que conocería a otra clase "*SeguroCoche*". Véase la Figura 1.

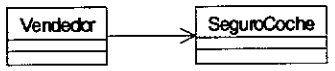

Figura 1. Diseño alternativo 1

Tras contrastarlo con el usuario, el diseñador tiene la constancia de que el modelo satisface los requisitos exigidos. Sin embargo, se podría intentar añadir flexibilidad al diseño. El diseñador podría argumentar que seguramente, en un futuro, se necesite tratar varios tipos de seguros. De esta forma, para mejorar el mantenimiento de la aplicación, decide añadir una clase abstracta "*Seguro*" al diseño, de manera que mediante herencia se puedan añadir más tipos de seguros, como por ejemplo de salud o de vida. Adicionalmente, decide añadir otra superclase, en este caso una interfaz "*IProducto*", para facilitar el mantenimiento en caso de que también se quieran añadir más tipos de productos.

El diseño resultante tras la mejora propuesta es el que se muestra en el segundo diseño alternativo (Figura 2). A primera vista es difícil decir cuál de los dos diseños es mejor. Se intuye que probablemente esto dependa de la flexibilidad que demande el usuario en un futuro.

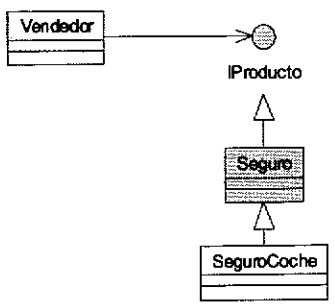

Figura 2. Diseño alternativo 2

En un tono sombreaado se representan las clases que han sido añadidas, y que no están directamente relacionadas con el problema planteado por el usuario. Se trata de *clases de servicio*, que son clases añadidas al diseño con el objeto de crear indirecciones o puntos de flexibilidad. Según Nordberg, estos puntos son "mecanismos más o menos artificiales, que permiten añadir flexibilidad y estabilidad frente a cambios" [6].

Como tercera alternativa, el diseñador podría pensar en añadir aún más estabilidad y flexibilidad a su diseño. Para ello, podría separar en componentes diferentes el diseño.

La separación en componentes añadiría aún más flexibilidad a nuestro sistema, ya que permitiría aislar completamente a los otros componentes del sistema de cambios, a la vez que posibilita añadir nuevos componentes dinámicamente al sistema.

El diseñador podría optar por utilizar un objeto "*ConectorComponente*", aislado a su vez por una interfaz "*IComponente*". Los datos que viajen entre componentes serán encapsulados en objetos de transferencia "*ObjetoTransfer*", tal y como muestra la Figura 3.

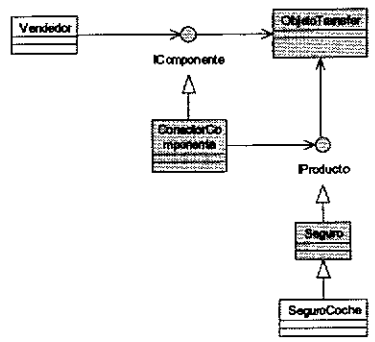

Figura 3. Diseño alternativo 3

El resultado es que se han añadido aún más clases de "servicio" para la misma cantidad de clases de dominio.

2.2. ¿Qué es el diseño basado en valor?

A medida que evolucionaba el diseño, hemos añadido nuevas clases de "servicio", en este caso representadas en azul. Esto nos ha dado sin duda una *mayor flexibilidad y estabilidad* ante cambios que no tenía el primer diseño. Sin embargo, cada vez que añadimos una clase de servicio es más difícil comprender el diseño, esto es, *se ha perdido analizabilidad*. Es además significativo señalar que el coste de implementación inicial aumenta según se complica el diseño.

Por tanto, según lo analizado en el ejemplo anterior, concluimos que añadir puntos de flexibilidad no es gratis. Si un diseño es muy flexible (típicamente implementando muchos patrones de diseño), tendrá una gran cambiabilidad y estabilidad, pero perderá en analizabilidad y además el coste inicial de construcción será mayor. Si no se implementa ningún patrón, la situación será justo la contraria [7]. La Figura 4 muestra el concepto.

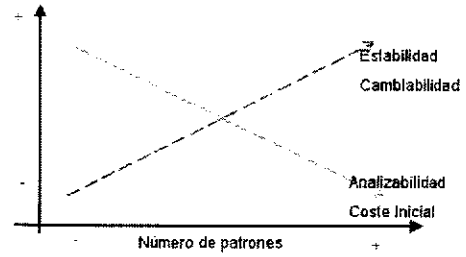

Figura 4. Impacto del número de patrones en diseño [7]

Entonces, ¿Cuándo se debe introducir un patrón de diseño o punto de flexibilidad? Los argumentos más comunes a favor de su inclusión de suelen ser:

- "*Es posible* que este artefacto cambie con frecuencia en el futuro, debería ser flexible frente a cambios"
- "Esta parte del diseño *parece ser* crítica. Por ello, debería ser estable frente a cambios"

Términos como "*es posible*" o "*parece ser*" señalan de por sí una carencia de sistematización y aproximación metodológica al problema. Hasta ahora, el hecho que determinaba si una indirección añadía valor al sistema era la experiencia o la astucia del desarrollador. Para mitigar esta carencia, se presentará una técnica que formaliza y sistematiza el proceso descrito. El objetivo subyacente es que las decisiones de diseño que determinarán la mantenibilidad del software no estén basadas en intuiciones o suposiciones.

3. Identificación del Valor

Una vez definido el concepto de Valor a nivel de diseño, se presentará la aproximación de sistematización del problema, y las técnicas específicas de Ingeniería del Software en que se apoya.

3.1. Aproximación al problema

Para maximizar la mantenibilidad de las soluciones, se propone comenzar por un modelo lo más simple posible e incrementar su complejidad allí donde las características del proyecto lo requieran. El mismo proceso puede ser también usado para guiar refactorizaciones.

La idea es formalizar el problema del proceso descrito en el apartado anterior, donde el diseñador evalúa la mantenibilidad de cada decisión de diseño, eligiendo entre complicar o mantener simple el sistema.

La Figura 5 muestra el proceso mediante un gráfico en el que las "X" representan posibilidades de mejora de diseño. En el eje de las abcisas representaremos la importancia relativa del artefacto sobre el que se ejerce la mejora. En el eje de las ordenadas, se situará la probabilidad de cambio asociada con dicho artefacto.

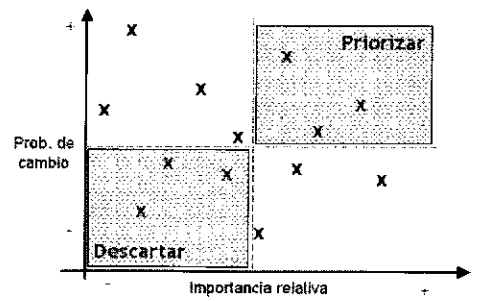

Figura 5. Priorización de mejoras de diseño

De esta manera se trata de *representar gráficamente el razonamiento* anteriormente expuesto, donde a mayor probabilidad de cambio y mayor importancia, mayor será la

mantenibilidad que aporte la mejora. Si dividimos todo el cuadrante de la gráfica en sectores, concluiremos que debemos descartar las mejoras que tengan menor prioridad de cambio y menor importancia para priorizar las mejoras sobre los artefactos más cambiantes e importantes.

Para llevar a cabo lo expuesto, necesitamos:

- Localizar los candidatos de cambio (representados por "X" en la gráfica)
- Estimar la probabilidad de cambio de cada artefacto relacionado con la mejora.
- Estimar la importancia relativa de cada artefacto.

3.2. Localización de las posibilidades de cambio

En primer lugar debemos identificar las posibles mejoras de diseño a aplicar, para luego poder estimar el valor relativo de cada una de ellas. En general, deberemos marcar como posibilidad de mejora aquellos lugares *donde se estime que el mantenimiento podría ser caro.*

Existe ya muchos trabajos sobre la materia, de entre los que podemos destacar principios, heurísticas, patrones o malos olores. En [8] se agrupó este conocimiento bajo el término de "*reglas de diseño*". Se trata de un catálogo de reglas que deben cumplir todas las aplicaciones software para que los costes de mantenimiento permanezcan bajo control. La idea es que allí donde una regla sea violada, hay una posibilidad de mejora de diseño para hacer el sistema más estable y flexible frente a posibles cambios y ampliaciones. La Tabla 1 muestra algunos ejemplos de estas reglas de diseño.

SI Hay dependencias de clases concretas **ENTONCES** Hacer que estas dependencias sean de abstracciones.
SI una súper clase conoce a alguna de sus subclases **ENTONCES** Eliminar ese conocimiento.
SI una clase colabora con muchas **ENTONCES** Reducir el número de colaboraciones.
SI entre una interfaz y su implementación no hay una abstracción **ENTONCES** Crear una clase abstracta con una implementación por defecto entre la interfaz y la clase que la implementa.
SI Una clase es grande **ENTONCES** Reducir su tamaño repartiendo funcionalidad en otras clases.
SI Una clase rechaza algo de lo que hereda **ENTONCES** Evitarlo, generalmente cambiando la herencia por delegación.

Tabla 1. Ejemplos de reglas de diseño

La idea principal es realizar un diseño basado en el modelo de análisis, utilizando un número mínimo de clases de servicio y abstracciones, y aplicar las reglas de diseño para marcar las clases o artefactos sujetos a una posible mejora, tal y como muestra la Figura 6.

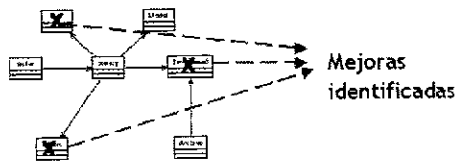

Figura 6. Identificación de mejoras en un diseño

El concepto de guía de diseño mediante heurísticas o métricas para mejorar la mantenibilidad no es nuevo. El problema de las herramientas y aproximaciones existentes es que localizan, en un sistema de tamaño medio, una cantidad inmanejable de métricas difíciles de interpretar. Además no asignan un valor relativo o prioridad a cada una de ellas. Por este motivo, los desarrolladores no pueden saber qué mejoras aportarán más valor a su caso concreto.

El objetivo final de la aplicación de reglas en nuestro modelo es restringir el número de mejoras aplicables a un diseño a una lista manejable. Desde una perspectiva basada en valor, se reducirá la lista de posibles mejoras a las que aporten más valor desde el punto de vista de la mantenibilidad, siguiendo la aproximación propuesta en el punto 3.1.

3.3. Estimación de la probabilidad de cambio

Las técnicas de "*predicción del cambio*" [9] pretenden estimar la probabilidad de cambio futura de cada uno de los artefactos de un diseño software. Muchas propuestas han sido presentadas recientemente al respecto.

El primer conjunto de técnicas que aparecieron fueron las *técnicas basadas en información histórica*, que revisan los cambios que sucedieron en el pasado, y auguran un comportamiento similar en el futuro más cercano. Por ejemplo, [10, 11] utilizan técnicas de "data mining" sobre el histórico para identificar cambios que deben ser realizados conjuntamente para preservar la consistencia del sistema.

Los métodos históricos normalmente dividen la vida del software en versiones. La Tabla 2 muestra un ejemplo de extracción de datos de tres versiones: V1, V2 y V3.

Artefacto	Número de cambios por Versión		
	V1	V2	V3
Art. 1	3	0	1
Art. 2	5	1	3
Art. 3	0	4	2

Tabla 2. Número de cambios por versión

Los cambios esperados para la siguiente versión ($Cambios_{n+1}$) pueden ser calculados en función de los cambios anteriores. Una aproximación simple (aunque existen más alternativas) es utilizar la media, expresada como la suma del número de cambios de las versiones anteriores ($\Sigma_n Cambios_n$) dividido por el número de versiones (n). Véase la ecuación (1).

$$(1) \quad Changes_{n+1} = \Sigma_n Changes_n / n$$

A continuación se muestra la aplicación de la ecuación (1) en el conjunto de datos de la Tabla 2.

Artefacto	Número estimado de cambios
Art. 1	4/3 = 1,25
Art. 2	9/3 = 3
Art. 3	6/3 = 2

Tabla 3. Numero de cambios estimado

Para completar esta aproximación, es posible tener en cuenta que en algunos casos, los cambios recientes tienen más importancia relativa que los cambios antiguos. [12] propone algunas métricas al respecto. Otro factor a tener en cuenta es señalado por [13], dado que "*Cuando el tiempo entre versiones consecutivas es muy corto, se puede observar una sobreestimación de cambios. Lo contrario puede observarse en periodos largos*". Se hace necesario por tanto predecir el número de cambios por unidad de tiempo. El mismo autor propone varios métodos para conseguir este fin, de entre los que destaca el uso de técnicas de aproximación polinómica.

Por otro lado, existe otra gran rama de *técnicas de predicción basadas en el análisis de la estructura estática* de los diseños. Varios investigadores han señalado que las estructuras y propiedades de los diseños orientados a objetos pueden identificar clases propensas al cambio.

Un ejemplo de estas técnicas es el bien conocido antipatrón "God Object", donde un objeto se relaciona con muchos otros objetos. Este objeto tendrá una mayor probabilidad de cambio porque si un objeto referenciado cambia su interfaz, el cambio puede ser propagado al primer objeto. Entre este conjunto de técnicas se puede destacar la técnica de los "ejes de cambio" (Axes of Change) ([9], [13]), las "técnicas de métricas de acoplamiento" (Coupling Measures) ([14], [15], [16]) o las técnicas basadas en tamaño (Size Measures) ([17], [18]).

Existen pues distintos métodos de predicción de cambio basados en históricos o análisis de estructuras. En el ámbito de nuestra propuesta, pretendemos utilizar la estimación basada en histórico, ya que entendemos que siempre será posible modelizar las técnicas de análisis estructural a través de reglas de diseño (ver Tabla 1). Por ejemplo, la métrica de tamaño de clase puede ser modelada como reglas del estilo de "si una clase es mayor de 5Kb, dividirla en distintas clases".

3.4. Importancia relativa de los artefactos

La dificultad de identificar la importancia relativa de cada artefacto de diseño desde el punto de vista de los stakeholders reside en que el usuario no conoce el diseño. La solución que

proponemos es *estimar la importancia de los requisitos* usando opinión de los usuarios, y posteriormente, *trasladar mediante técnicas de trazabilidad dichos datos a los artefactos de diseño* que tengan una valoración más alta.

3.4.1. Asignación de importancia a los requisitos

Para extraer la importancia relativa de cada requisito, existen varias propuestas:

- [19] propone una técnica basada en "Minimal Marketable Features – MMFs", o funcionalidades que tengan sentido desde un punto de vista de marketing. Propone ordenar dichos MMFs para maximizar el valor de los proyectos, y reducir el coste inicial de las inversiones.
- Por su parte, [20] ordena los requisitos basándose en el valor, el riesgo o volatilidad de los requisitos y los costes. El sistema de priorización utilizado para estas tres variables fue extraído de [21].
- [22] incluye consideraciones de negocio, propone un framework que conecte las necesidades de negocio con la priorización de los distintos requisitos según los parámetros: Sales (ventas), Marketing, Competitive (competitividad), Strategic (Estrategia), Customer Retention (retención del usuario).

Cualquiera de las propuestas anteriores tienen como finalidad la estimación de un valor o puntuación normalizado de la importancia de cada requisito. Por tanto, cualquiera de ellos es aplicable. La elección del método dependerá de si es conveniente para la organización que se incluyan parámetros de negocio o simplemente parámetros de valor, riesgo y coste.

3.4.2. Trazabilidad de la importancia desde los requisitos al diseño

Existen relaciones de dependencia entre todos los artefactos desarrollados durante el ciclo de vida, *"La trazabilidad de requisitos es la habilidad de seguir la vida de un requisito"* [23], es decir, la relación entre todos los artefactos inferidos a partir de los requisitos.

Un interesante resumen de técnicas de trazabilidad puede ser consultado en [24]:

- Uniones simples (matrices de artefactos)
- Uniones recuperadas semánticamente
- Uniones ejecutables
- Trazabilidad de requisitos no funcionales utilizando patrones de diseño

Para la trazabilidad de requisitos funcionales, las uniones simples (Simple Links - SL) son el método más fiable, porque no está basado en heurísticas o automatismos, sino en un conjunto de relaciones mantenidas manualmente. Sin embargo, SL suele consumir muchos recursos, lo que la hace inviable para la mayoría de los casos. Esa es la razón por la que existen técnicas (automáticas o semiautomáticas) más complejas.

En el contexto de esta investigación, no se recomienda automatizar el proceso de trazabilidad, porque ya se sabe a priori qué requisitos son más importantes para los stakeholders. Por ello, la cantidad de recursos se reduce drásticamente, y se recomienda la consideración de SL como primera alternativa.

4. Transformación del Valor en Esfuerzo

Cualquier técnica de cuantificación encontrada en la literatura (COCOMO [25], Puntos de Función [26], o cualquier otra) se basa en la apropiada medición de experiencias dependiendo de variables de tamaño o de funcionalidad. Posteriormente se extrapolan los resultados a proyectos de características similares.

En este caso, se propone utilizar la misma aproximación al problema. Para ello, se utilizarán experimentos controlados para estimar el coste que supone arreglar la violación de cada una de las reglas de diseño existentes en el catálogo.

Deberemos estimar tres costes, el coste de mantenimiento de código que viola una regla (Coste A), el coste de reparación de la violación de la regla (Coste B), y finalmente el coste de mantenimiento una vez reparada la violación (Coste C). Todos ellos se expresarán en "horas de trabajo".

Una vez conocidos dichos costes, es posible conocer la *rentabilidad en mantenimiento de una mejora de diseño*, basándonos en la cantidad de cambios que se estima que habrá durante la fase de mantenimiento.

Se propone planear experimentos controlados en grupos de trabajo, donde cada grupo debe modificar diseños. La Tabla 4 muestra un ejemplo de los datos estimados a partir de la experimentación descrita.

	Coste A	Coste B	Coste C
Regla 1	4 horas	4 horas	2 horas
Regla 2	3 horas	6 horas	1 hora
Regla 3	8 horas	12 horas	6 horas

Tabla 4. Estimación de costes asociados a cada regla de diseño

Una vez que hayamos obtenido los costes asociados a cada regla, es posible calcular si arreglar la violación de una regla resultará beneficioso a lo largo de la vida completa del proyecto, es decir, sabremos si el diseño favorece la mantenibilidad de ese software. Para ello, deberemos calcular el número estimado de cambios (n) asociados al artefacto afectado, según se muestra en la ecuación (2).

(2) $n = ProbCambio(Clase) * Numero\_versiones$

A partir de ahí, podemos estimar la rentabilidad del coste actual, y el coste después de la mejora. El cambio será rentable si se cumple la inecuación (3).

(3) $Coste(A) * n > Coste(B) + (Coste(C) * n)$

Donde la primera parte de la inecuación representa el coste actual, y la segunda el coste después de la mejora.

Nótese que dichos costes pueden venir definidos por variables. Por ejemplo, el coste asociado con un cambio por violación de la regla "SI Elementos de Interfaz de Usuario están en Entidades de Dominio, ENTONCES Colocar estos elementos en una entidad aparte" dependerá del número de entidades de dominio del que estemos hablando.

5. Conclusiones y trabajo futuro

Se ha presentado una técnica que pretende guiar el diseño o refactorización de un sistema software basándose en la optimización de los costes de mantenimiento.

En el fondo de la propuesta, destaca la utilización de reglas de diseño para acotar la cantidad ilimitada de opciones de diseño válidas, que pueden ser propuestas para un mismo conjunto de requisitos. El hecho de acotar los diseños mediante reglas nos permite poder estimar el valor relativo de la violación de cada una de ellas. Además, este sistema basado en reglas hace posible estimar un cálculo del esfuerzo que implica cada una de las decisiones de diseño, ya que es posible acotar el coste o beneficio de la aplicación de cada una de las reglas.

Con respecto a la validación de costes, queda pendiente realizar experimentos controlados que estimen empíricamente los costes asociados a cada regla.

Finalmente, y como apoyo a todo lo anterior, se propone la creación de herramientas de automatización que apoyen la sistematización de la técnica, y que permitan realizar pruebas en entornos reales.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto GOLD (Ministerio de Educación y Ciencia TIN 2005-00010, España) y por el proyecto FAMOSO-Fabricación y Modernización de Software dirigida por Modelos (Ministerio de Industria, Turismo y Comercio FIT-340000-2005-161).

Referencias

[1] M. Shaw, A. Arora, S. Butler, V. Poladian, and C. Scaffidi, "First Steps toward a Unified Theory for Early Prediction of Software Value," Proc. IEEE Equity, Amsterdam, 2007.

[2] B. Boehm, "Value-Based Software Engineering: Overview and Agenda," in *Value-Based Software Engineering* Springer, 2005, pp. 3-14.

[3] K. H. Bennet and V. T. Rajlich, "Software Maintenance and Evolution: a Roadmap,"

Proc. ICSE (Track on The Future of Software Engineering), Limerick, Ireland, 2000.

[4] T. M. Pigoski, *Practical Software Maintenance. Best Practices for Managing your Investments*. NY. USA: John Wiley & Sons, 1996.

[5] D. L. Parnas, *Software fundamentals: collected papers by David L. Parnas*. Boston, MA, USA Addison-Wesley Longman Publishing Co., Inc, 2001.

[6] M. E. Nordberg, "Aspect-Oriented Indirection – Beyond OO Design Patterns," Proc. OOPSLA 2001, Workshop Beyond Design: Patterns (mis)used, Tampa Bay, Florida, EEUU, 2001.

[7] J. Garzás and M. Piattini, "Analyzability and Changeability in Design Patterns," Proc. SugarLoafPLoP. The Second Latin American Conference on Pattern Languages of Programming, Itaipava, Río de Janeiro, Brasil, 2002.

[8] J. Garzás and M. Piattini, "An ontology for micro-architectural design knowledge," *IEEE Software*, vol. 22, pp. 28-33, 2005.

[9] N. Tsantalis, A. Chantzigeorgiou, and G. Stephanides, "Predicting the Probability of Change in Object-Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 31, pp. 601-614, 2005.

[10] A. Ying, G. Murphy, R. Ng, and M. Chu-Carroll, "Predicting Source Code Changes by Mining Change History," *IEEE Transactions on Software Engineering*, vol. 30, pp. 574-586, 2004.

[11] T. Zimmermann, A. Zeller, P. Weigerber, and S. Diehl, "Mining Version Histories to Guide Software Changes," *IEEE Transactions on Software Engineering*, vol. 31, pp. 429-445, 2004.

[12] T. Girba, S. Ducasse, and M. Lanza, "Yesterday's Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes," Proc. 20th IEEE International Conference on Software Maintenance Washington, DC, USA 2004

[13] A. R. Sharafat and L. Tahvildari, "A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems," Proc. International Conference in Software Maintenance and Reengineering, Amsterdam, 2007.

[14] L. C. Briand, K. E. Emam, D. Surmann, I. Wieczorek, and K. D. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," Proc. International Conference on Software Engineering Los Angeles, California, United States 1999.

[15] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software," *IEEE Transactions on Software Engineering*, vol. 24, pp. 629-639, 1998.

[16] K. Chen and V. Rajlich, "RIPPLES: tool for change in legacy software," Proc. IEEE International Conference on Software Maintenance, Florence, Italy, 2001.

[17] E. Arisholm, L. C. Briand, and A. Føyen, "Dynamic Coupling Measurement for Object-Oriented Software," *IEEE Transactions on Software Engineering*, vol. 30, pp. 491-506, 2004.

[18] F. G. Wilkie and B. A. Kitchenham, "Coupling Measures and Change Ripples in C++ Application Software," *Systems and Software*, vol. 52, pp. 157-164, 2000.

[19] J. Cleland-Huang and M. Denne, "Financially informed requirements prioritization," Proc. International Conference on Software Engineering St. Louis, MO, USA 2005.

[20] M. Heindl and S. Biffl, "A Case Study on Value-based Requirements Tracing," Proc. 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering Lisbon, Portugal 2005.

[21] A. Ngo-The and G. Ruhe, "Requirements Negotiation under Incompleteness and Uncertainty," *Software Engineering and Knowledge Engineering*, 2003.

[22] J. Azar, R. K. Smith, and D. Cordes, "Value-Oriented Prioritization: A Framework for Providing Visibility and Decision Support in the Requirements Engineering Process," Department of computer science. University of Alabama, Alabama 2007.

[23] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," Proc. 1st International Conference on Requirements Engineering, Colorado Springs, CO, USA, 1994.

[24] J. Cleland-Huang, G. Zemont, and W. Lukasik, "A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability," Proc. Requirements Engineering Conference, 12th IEEE International (RE'04), 2004.

[25] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II* Prentice Hall PTR, 2000.

[26] G. Antoniol, C. Lokan, G. Caldiera, and R. Fiutem, " A Function Point-Like Measure for Object-Oriented Software," *Empirical Software Engineering*, vol. 4, pp. 263 - 287, 1999.