XXI JORNADAS DE INGENIERÍA DEL SOFTWARE Y BASES DE DATOS

Jesús J. García Molina (Ed.)

JISBD

Ediciones Universidad Salamanca

XXI Jornadas de Ingeniería del Software y Bases de Datos

JESÚS J. GARCÍA MOLINA (ED.)

XXI Jornadas de Ingeniería del Software y Bases de Datos

Ediciones Universidad Salamanca

AQUILAFUENTE
219

Motivo de cubierta:
Diseñadora María Alonso Miguel

1.º edición: septiembre, 2016
ISBN: 978-84-9012-627-1 (PDF)

Ediciones Universidad de Salamanca
www.eusal.es
eusal@usal.es
Realizado en España – Made in Spain

Obra sometida a proceso de
evaluación mediante sistema de revisión por pares a ciegas
a tenor de las normas del congreso

Ediciones Universidad de Salamanca es miembro de la UNE
Unión de Editoriales Universitarias Españolas
www.une.es

CEP

A José María Troya

En una comida en Sevilla con José María Troya, Pere Botella e Isidro Ramos y en el marco de unas jornadas de trabajo organizadas por Miguel Toro decidimos iniciar formalmente la presentación de trabajos de investigación en Ingeniería del Software en forma de un congreso anual: las Jornadas de Ingeniería del Software, las que posteriormente, con la fusión de las Jornadas de Bases de Datos, darían origen a las JISBD y posteriormente a SISTEDES.

Algo se hizo bien y no fue fruto de la casualidad. Existían las bases suficientes para emprender nuestro particular viaje a Ithaca.

José María, desde su época de estudiante de la Especialidad de Cálculo Automático de la UCM, y más tarde durante la realización de su tesis doctoral, en la que resolvió un problema difícil definiendo una regla heurística para rebajar su complejidad computacional, se mostró como un trabajador infatigable, con una capacidad de iniciativa propia, una ética profesional a toda prueba y una sonriente amabilidad.

Los frutos de su posicionamiento personal y científico, excelentes y públicos, se acompañaron siempre con una amabilidad y seriedad que fraguaron el clima de amistad que caracteriza a nuestra comunidad. Gracias también por eso, José María.

Nosotros no hemos llegado todavía a Ithaca. Tú te anticipaste también en esto. Nos atrevemos a decir, recordando nuestras experiencias latinoamericanas juntos, que "te fuiste pronto como los elegidos en plena gloria y juventud"

Los versos de Kavafis en los que usa el Viaje como metáfora de la Vida no son capaces de llenar el vacío que nos has dejado, pero captan tu particular singladura:

"Cuando emprendas tu viaje a Ithaca /pide que el camino sea largo, /lleno de aventuras, lleno de experiencias/No temas a los lestrigones ni a los cíclopes / ni al colérico Poseidón, /seres tales jamás hallarás en tu camino, /si tu pensar es elevado, si selecta /es la emoción que toca tu espíritu y tu cuerpo/Ten siempre a Ithaca en tu mente. /Llegar allí es tu destino."

Pero, con celeridad y silencio nos has dejado:

"Mas no apresures nunca el viaje/Mejor que dure muchos años /y atracar, viejo ya, en la isla, enriquecido de cuanto ganaste en el camino /sin esperar que Ithaca te enriquezca"

Tu ausencia estará siempre presente en JISBD, la cuales ayudaste a crear.

Descansa en Paz

Pere Botella, Isidro Ramos y Miguel Toro

Prólogo

Las "Jornadas de Ingeniería del Software y Bases de Datos" (JISBD) constituyen el foro que cada año reúne a la comunidad científica española en las áreas de Ingeniería del Software y Bases de Datos y siempre han atraído el interés de grupos de investigación de Portugal e Iberoamérica en estas dos áreas. JISBD es organizada por la Sociedad de Ingeniería de Software y Tecnologías de Desarrollo de Software (SISTEDES) junto a otras dos conferencias: Jornadas de Programación Declarativa (PROLE) y Jornadas de Ciencia e Ingeniería de Servicios (JCIS). La vigésimo primera edición de JISBD es uno de los quince eventos científicos que integran la IV Conferencia Española de Informática (CEDI) que se celebra en Salamanca. CEDI tiene una periodicidad de tres años y su propósito es mostrar a la sociedad el estado actual de la informática en España.

En la edición actual, JISBD ha continuado con la organización basada en áreas temáticas o "tracks" puesta en marcha en la edición previa. Los tipos de contribución han sido los considerados en ediciones anteriores: artículos completos, artículos cortos, artículos relevantes y demos. Como novedad se realizó un llamamiento a nuevos tracks dentro de la primera solicitud de contribuciones lanzada a principios de noviembre de 2015. Dado que no se recibió ninguna solicitud no fue necesario aplicar el mecanismo previsto para dar cabida a nuevos tracks. Por tanto, JISBD'2016 incluye los mismos tracks que en la edición anterior: *Arquitecturas del Software y Variabilidad*, *Calidad y Pruebas*, *Desarrollo de Software Dirigido por Modelos*, *Gestión de Datos*, *Ingeniería del Software Guiada por Búsqueda*, *Ingeniería Web y Sistemas Pervasivos* y *Procesos Software y Metodologías*; y de nuevo se ha incluido el track *Abierto* para dar cabida a los trabajos que no encajan en ninguno de los tracks anteriores.

Otra novedad ha tenido que ver con la modalidad de "artículos relevantes" ya publicados en revistas con índices de impacto o conferencias internacionales prestigiosas en las áreas asociadas a un determinado track. Con el fin de facilitar el proceso de selección y asegurar la calidad de estas contribuciones, se ha establecido que un "trabajo relevante" debe haber sido publicado en una revista en el cuartil Q1 de JCR o en una de las dos conferencias que ha seleccionado cada track.

Cabe destacar un incremento significativo en el número de contribuciones recibidas con respecto a las tres ediciones anteriores. Mientras en 2015 se recibieron 70 contribuciones, 54 en 2014 y 64 en 2013, en esta edición se han recibido 94 contribuciones (34 completos, 29 cortos, 25 relevantes y 6 demos). El número de trabajos aceptados ha sido 79 (30 completos, 21 cortos, 24 relevantes y 5 demos). Estos números parecen avalar la nueva organización en torno a áreas temáticas y que las JISBD pueden jugar un importante papel para dinamizar las diferentes comunidades relacionadas con la ingeniería del software y las bases de datos en nuestro país.

La conferencia invitada será impartida por Andrei Voronkok, prestigioso investigador de la Universidad de Manchester que ha recibido el premio Herbrand por sus contribuciones al razonamiento automático y que es creador de EasyChair, una de las herramientas de gestión de conferencias más extendidas en el mundo. El Dr. Voronkok analizará los desafíos a los que se ha debido hacer frente en la construcción de EasyChair desde el punto de vista del diseño de software y la gestión de los datos, así como de los retos para el futuro. Además, las conferencias invitadas de JCIS (Tommi Mikkonnen, Institute of Pervasive Computing, Tampere, Finland) y de PROLE (Arnaud Gotlieb, Simula Research Laboratory, Norway), que se celebran en paralelo en esta ocasión, son también parte del programa de JISBD'2016.

Dada la estructura actual de JISBD basada en track, toda la labor de organización es realizada por un equipo formado por el Presidente del Comité de Programa y los coordinadores de cada track (el listado aparece a continuación de esta presentación). Expreso mi agradecimiento a cada uno de los coordinadores de tracks por el esfuerzo que han realizado y por su buena disposición, ha sido un placer coordinar este equipo. El trabajo que he debido realizar ha requerido un contacto permanente con SISTEDES y con el Comité Organizador de CEDI'2016. Por un lado, debo agradecer a Fernando de la Prieta toda la ayuda prestada, como persona de contacto con dicho comité, para resolver todas las cuestiones relacionadas con la web, el uso de EasyChair, la edición de actas y la gestión económica. Por otro lado, contar con Oscar Díaz como enlace con SISTEDES me ha dado una gran tranquilidad en la toma de decisiones. Agradezco también el apoyo recibido en todo momento por Diego Sevilla.

Los agradecimientos finales para aquellos que son los principales protagonistas: autores y revisores. A los primeros por apoyar a JISBD con el envío de publicaciones y la presentación de sus trabajos en Salamanca, y a los segundos por su dedicación a la tarea de mantener el nivel de calidad esperado de las contribuciones a JISBD y ayudar a los autores a mejorar sus trabajos.

Y por último, esta presentación de JISBD'2016 no puede acabar sin recordar a José María Troya, uno de los impulsores de estas Jornadas y de la Ingeniería del Software en nuestro país.

Salamanca, 13 de septiembre de 2016

Jesús J. García Molina
Presidente del Comité de Programa

Comité de programa

Presidente

Jesús Joaquín García Molina (Universidad de Murcia)

Coordinadores de los tracks

Arquitecturas Software y Variabilidad: David Benavides (Universidad De Sevilla) y Jennifer Pérez Benedí (Universidad Politécnica de Madrid)

Calidad y Pruebas: Carme Quer (Universitat Politècnica de Catalunya) y María José Suárez-Cabal (Universidad de Oviedo)

Desarrollo de Software Dirigido por Modelos: Cristina Vicente Chicote (Universidad de Extremadura) y Juan de Lara (Universidad Autónoma de Madrid)

Gestión de Datos: Sergio Ilarri (Universidad de Zaragoza) y José Ramón Paramá (Universidad de A Coruña).

Ingeniería del Software Guiada por Búsqueda: José Raúl Romero Salguero (Universidad de Córdoba) y José Francisco Chicano García (Universidad de Málaga)

Ingeniería Web y Sistemas Pervasivos: Elena Navarro (Universidad de Castilla-La Mancha) y Roberto Rodríguez Echeverría (Universidad de Extremadura)

Proceso Software y Metodologías: Mercedes Ruiz (Universidad de Cádiz) y Agustín Yagüe (Universidad de Politécnica de Madrid)

Tema Abierto: Jesús J. García Molina (Universidad de Murcia)

Coordinador de demostraciones

Diego Sevilla Ruiz (Universidad de Murcia)

Enlace con SISTEDES

Óscar Díaz García (Universidad del País Vasco)

Hacia un entorno extensible basado en ADM para la refactorización de sistemas heredados

Abel Lorente Ramírez, Ignacio García-Rodríguez de Guzmán, Mario Piattini

Instituto de Tecnologías y Sistemas de Información (ITSI)
Universidad de Castilla-La Mancha
Paseo de la Universidad, nº4, 13071
{Abel.Lorente, Ignacio.GRodriguez, Mario.Piattini}@uclm.es

Resumen. Hoy en día siguen activos muchos sistemas heredados que presentan problemas que afectan a las distintas características de la calidad del software. Para mejorar estos problemas, existen herramientas de refactorización, cuyo objetivo es mejorar aspectos de calidad concretos sin afectar al comportamiento del sistema heredado. ADM (Modernización Dirigida por la Arquitectura), se presenta como el paradigma que basa el entendimiento y evolución de los sistemas software en MDA. Existen multitud de entornos que implementan estrategias de refactorización clásicas para mejorar la calidad de los sistemas, pero estas herramientas ofrecen un catálogo fijo de refactorizaciones. La propuesta que se presenta en este artículo consiste en un entorno flexible basado en ADM que permite la definición de *"bad-smells"* (clásicos y nuevos), aplicables a contextos concretos y su identificación en sistemas heredados, teniendo así una herramienta totalmente flexible y extensible.

1 Introducción

Una de las herramientas por excelencia para mejorar la calidad del software sin afectar a su funcionalidad es la reingeniería, más concretamente la fase de refactorización [1] que es el proceso de cambiar un sistema software de forma que mejore su estructura interna y aspectos de calidad sin cambiar su comportamiento exterior [2].

Aunque hoy en día pueden encontrarse multitud de herramientas que de un modo u otro asisten en el proceso de refactorización, existe una limitación común en todas ellas: no pueden ser extendidas, por lo que éstas sólo pueden actuar sobre el conjunto de *bad-smells* que tienen definido por defecto. Aunque es muy común encontrar un amplio conjunto de *bad-smells* [2] implementados para su detección, no se pueden extender los mecanismos para detectar nuevos *bad-smells* que lejos de estar catalogados, pueden considerarse como problemas de calidad en algún contexto concreto.

Como vía para soluciar este problema, se considera el paradigma MDA[1], que ha dado lugar a un nuevo enfoque de reingeniería denominado *Architecture-Driven Modernization* o ADM[2], que es el proceso de entendimiento y evolución de los sistemas

[1] http://www.omg.org/mda/
[2] http://www.omg.org/adm/

software existentes con el propósito de la mejora, modificación, interoperabilidad, refactorización, restructuración, reusabilidad, portabilidad, migración, traducción, SOA, y migración MDA del software, promovido por la OMG[3].

El metamodelo KDM (*Knowledge Discovery Metamodel*) es el principal pilar de ADM, y proporciona una visión integrada de la estructura del sistema heredado [3]. KDM puede verse como un conjunto de sub-metamodelos que permiten modelar cada una de las posibles vistas del sistema (código, interfaz de usuario, datos, aspectos arquitectónicos, etc.) pero manteniendo la coherencia y la trazabilidad entre los elementos de las mismas.

Por este motivo, en este trabajo se presenta una herramienta flexible y extensible basada en ADM que permite la definición de clásicos o nuevos *bad-smells* aplicables a contextos concretos y su identificación en sistemas heredados, con la finalidad de más adelante realizar, del mismo modo, refactorizaciones asociadas a esos *bad-smells*.

El artículo se organiza de la siguiente manera: la sección 2 ofrece una visión resumida de algunas de las propuestas más actuales para la refactorización de sistemas heredados; la sección 3 presenta el entorno de identificación de *bad-smells* y se detalla su arquitectura; y en la sección 4 se exponen las conclusiones finales del artículo.

2 Estado del arte

Actualmente existen diferentes herramientas de código abierto que realizan refactorizaciones. En la Tabla 1 se pueden observar las más relevantes.

Tabla 1. Comparativa de herramientas actuales de refactorización.

Herramienta	Refactorizaciones	Extensibilidad
Eclipse IDE[4]	Refactorizaciones entre las que se encuentran extraer o mover método, extraer clase o renombrados	No
BeneFactor[4]	Refactorizaciones básicas (extraer método, renombrar)	No
JDeodorant[5]	Mover método, extraer método, extraer clase, eliminar código duplicado	No
DNDRefactor[5]	Mover variables, métodos, clases entre paquetes o extraer método	No
AutoRefactor[6]	Entre 20 y 30 refactorizaciones sencillas, como eliminar modificadores, insertar sentencias if o simplificar expresión	No

[3] Object Management Group: http://www.omg.org
[4] https://eclipse.org/
[5] http://www.jdeodorant.org/
[6] https://marketplace.eclipse.org/content/autorefactor

Tras dicho análisis se extraen dos principales conclusiones en relación al tema que se desea tratar con este trabajo: (i) hay refactorizaciones que se encuentran prácticamente en la totalidad de las herramientas, y (ii) existe un aspecto que salta a la vista: ninguna de estas herramientas es flexible o extensible, pudiendo ser necesario el uso de herramientas distintas para resolver determinados problemas de calidad.

3 Entorno de identificación de *bad-smells*

Una de las principales motivaciones para el desarrollo de este framework de reingeniería es dotar al entorno de la flexibilidad necesaria para añadir nuevos mecanismos de detección de *bad-smells* y aplicación de refactorizaciones. Por este motivo, se ha planteado el desarrollo de este entorno como un *plug-in* Eclipse, por la facilidad que aporta a la hora de agregar nuevas funcionalidades a través de *plug-ins*. La Fig. 1 muestra un diagrama arquitectónico de la versión actual del entorno de refactorización.

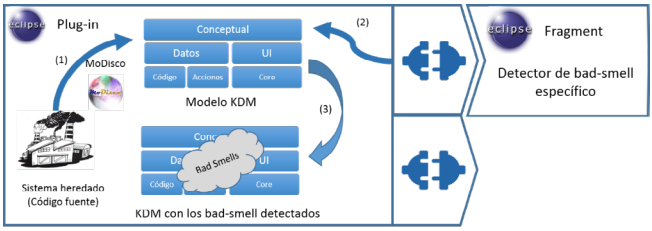

Fig. 1. Diagrama arquitectónico de la herramienta.

La estrategia que se ha seguido para conseguir que la arquitectura cumpla con estos requisitos radica en la modularidad, posibilitando el desarrollo de los detectores de bad-smells y las refactorizaciones como *plug-ins* del tipo "fragment" de Eclipse, que se asocian directamente con el núcleo de la herramienta (*plug-in host*).
La detección de *bad-smells* se lleva a cabo siguiendo los siguientes pasos: (i) ingeniería inversa y generación del modelo KDM del sistema heredado mediante el *plug-in* de MoDisco[7] integrado en el entorno de refactorización para hacer su uso trasparente al usuario (Fig.1. (1)); (ii) selección de *bad-smells* a detectar (cuyo "detector" o *plug-in* ha sido previamente seleccionado) sobre el KDM recuperado (Fig.1. (2)); (iii) generación de un nuevo modelo KDM marcado identificando las distintas ocurrencias del *bad-smell* en el modelo anterior (Fig.1. (3)).

Hasta el momento se ha desarrollado parte del entorno, y una de las cosas presentes es la consecución de integración de una primera versión de detector de *bad-smell* basado en el anti-patrón "Clase Dios" comprobando así la viabilidad de la propuesta.

[7] https://eclipse.org/MoDisco/

Conseguido este objetivo, se aprecia la diferencia con respecto a las alternativas existentes, ya que esta herramienta permite la incorporación de numerosas instancias de detectores personalizadas.

4 Conclusiones

En este artículo se han presentado los primeros pasos hacia un entorno para la detección de *bad-smells* en sistemas heredados. Este entorno pretende solucionar un problema del que adolecen las herramientas actuales de refactorización, que es la falta de flexibilidad para definir procedimientos de detección de problemas de calidad que no han sido clasificados como los clásicos *bad-smells*, y que podrían aplicarse a distintos ámbitos de la calidad del software. Otra de las ventajas es la posibilidad de crear repositorios abiertos para que los desarrolladores puedan proponer sus propias implementaciones.

El objetivo final es el desarrollo de un entorno completo de refactorización, que no sólo (i) detecte los *bad-smells*, sino que (ii) permita resolverlos en el modelo KDM y el sistema heredado. Por este motivo, este artículo presenta una visión general de la primera parte del objetivo principal, la detección de los problemas de calidad que se describan en los detectores de *bad-smell* desarrollados como *plug-ins*.

5 Agradecimientos

Este trabajo ha sido financiado por los proyectos: GINSENG (TIN2015-70259-C2-1-R), SEQUOIA (TIN2015-63502-C3-1-R), (Ministerio de Economía y Competitividad y Fondo Europeo de Desarrollo Regional FEDER, y MOTERO (Consejería de Educación, Ciencia y Cultura de la JCCM, y FEDER, PEII11- 0399-9449)

6 Referencias

[1] E. J. Chikofsky and J. H. C. II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Softw.,* vol. 7, pp. 13-17, 1990.

[2] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*: Addison-Wesley, 1999.

[3] R. Pérez-Castillo, I. G.-R. d. Guzmán, and M. Piattini, "Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems," *Comput. Stand. Interfaces,* vol. 33, pp. 519-532, 2011.

[4] X. Ge and E. Murphy-Hill, "BeneFactor: a flexible refactoring tool for eclipse," presented at the Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, Portland, Oregon, USA, 2011.

[5] Y. Y. Lee, N. Chen, and R. E. Johnson, "Drag-and-drop refactoring: intuitive and efficient program transformation," presented at the Proceedings of the 2013 International Conference on Software Engineering, San Francisco, CA, USA, 2013.