HACIA EL DISEÑO DE APLICACIONES WEB REUSABLES

Torres, A.; Escalona, M.; Gutiérrez, J.

Abstract

This paper presents a proposal for a process of software refactoring that allows increasing the degree of reusability of a Web application. The process involves the use of design patterns JEE, using methods of quantitative evaluation of refactoring software and reuse metrics. A case study of a project management module for NDT is showed.

Keywords: Reusable, refactoring, design patterns JEE, metrics.

Resumen

Este artículo presenta una propuesta de un proceso de refactorización de software que permite aumentar el grado de reusabilidad de una aplicación Web. El proceso incluye el uso de patrones de diseño JEE, el uso de métodos de evaluación cuantitativa de refactorizaciones de software y de métricas de reuso. Se muestra como caso de estudio una propuesta de módulo de gestión de proyectos para NDT.

Palabras clave: Reuso, refactorización, patrones de diseño JEE, métricas.

1. Introducción

El diseño de aplicaciones reusables es a menudo un objetivo difícil de alcanzar, lo que ya ha levantado las críticas hacia la tecnología orientada a objetos [1]. Además de la dificultad de obtener un diseño reusable, los programas deben evolucionar constantemente para adaptarse a los cambios de los requisitos ocurridos durante su ciclo de vida. Estos cambios son inevitables y vuelven al código más complejo, lo que acaba disminuyendo la calidad del software [2].

Las expectativas iniciales de la orientación a objetos fue que el software se comporte como los componentes de los circuitos eléctricos, es decir, que se pueda acoplar a medida que se los necesite de una manera predecible y determinista.

Este concepto ha sido reemplazado hoy en día con una idea más realista, con la reutilización de especificaciones de diseño que proporciona importantes beneficios, y además del uso de los *frameworks* que permiten la reusabilidad. Dado las expectativas iniciales, el objetivo es realizar un diseño para el reuso. Erich Gamma et al. [3] sostienen que el diseño de sistemas de software reusables se torna difícil pues una completa comprensión del sistema sólo se consigue cuando finaliza el proyecto.

Este retraso en cuanto a compresión del sistema corre el riesgo de generar niveles altos de abstracción, intentan soportar la reusabilidad basados en un futuro incierto y desconocido que muchas veces no se materializa.

Por lo tanto, una alternativa adecuada es, refactorizar para un futuro reuso; es decir, reestructurar el sistema completado sin modificar su comportamiento. En la refactorización, los desarrolladores ya no están preocupados con la adición tardía de alguna funcionalidad en el proyecto.

Por otro lado, los componentes de software que los desarrolladores tienden a reusar, la mayoría de veces son componentes pequeños tal como tipos abstractos de datos, funciones que manejan memoria, de entrada/salida, o librerías de clases. Sin embargo, la reusabilidad de pequeños componentes de software produce un ahorro mínimo, pues sólo representan un pequeño porcentaje en el producto final, según Poulin [4] tal vez sea un 20% de la aplicación entera, el otro 15% es específico de la aplicación; por tanto, no es normalmente reusable. Esto deja un 65% que es de dominio específico.

Entonces, se puede contar con un mayor ahorro si reusamos el software de dominio específico, pero para ello es necesario que el equipo de desarrollo separe claramente el dominio de lógica de negocio de la aplicación. El equipo de desarrollo puede lograr esta separación mediante el diseño de software a fin de que su estructura de clases exponga una alta cohesión y un bajo acoplamiento.

Desafortunadamente, el diseño de software reusable es complicado. Si el equipo de desarrollo permite que los requisitos nunca se materialicen, el software se torna complicado e ineficiente; los errores de requisitos son los más costosos de arreglar. En estas circunstancias, identificar los aspectos de diseño es una cuestión importante. Es así que el equipo de desarrollo necesita un conjunto de instrucciones para manejar estas situaciones. El diseño de patrones provee esta guía. El diseño de patrones deshace el acoplamiento entre los componentes del software, permitiendo la evolución de ciertos tipos del software con mínimos cambios. Sin embargo, el tiempo en el mercado Web - aplicaciones de estudio en este trabajo - es a menudo la principal prioridad y muchas veces los desarrolladores no tienen el tiempo adecuado para crear un diseño flexible. Pero, incluso si las aplicaciones no tienen la flexibilidad requerida, es posible introducir esa flexibilidad por medio de la refactorización de la aplicación.

Si bien es cierto, existen técnicas y métodos de diseño y de refactorización, se carece de procesos de desarrollo que integren dichas técnicas y que orienten hacia el desarrollo de aplicaciones reusables en el ambiente Web. Este trabajo presenta un proceso de refactorización que permite aumentar el grado de reusabilidad de aplicaciones Web. El proceso se divide en dos etapas: refactorización a nivel de clases mediante las refactorizaciones propuestas por Martin Fowler [5], y refactorización a nivel de aplicación mediante la aplicación de patrones de diseño JEE (Java Enterprise Edition, anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE) [6]. A la vez, en la primera etapa se propone medir la refactorización mediante el proceso de evaluación cuantitativa de refactorizaciones de software propuesta por Figueiredo [7] y al final del proceso se propone la aplicación de las métricas de reutilización de Poulin [8].

El resto del artículo está dividido de la siguiente manera: la sección 2 presenta conceptos básicos de refactorización, y se hace hincapié a las propuestas de Fowler; la sección 3 presenta los conceptos de patrones de diseño JEE; la sección 4 se divide en dos partes, en primer lugar muestra el proceso de evolución cuantitativa de refactorizaciones de software y en seguida las métricas de reuso; luego en la sección 5 se presenta la propuesta del artículo donde se describe las ventajas de su utilización, mostrando su aplicación en un caso de estudio presentado en la sección 6. Finalmente la sección 7 presenta las conclusiones y trabajos futuros.

2. Refactoring

El término refactorización fue originalmente usado por William Opdyke [9] y significa “una alteración en la estructura interna del software para tornarlo mas fácil de entender y menos costoso de se modificado, sin alterar su comportamiento observable” [5]. Las refactorizaciones son aplicadas cuando se desea mejorar algún atributo de calidad del software, como modularidad, reusabilidad, complejidad, mantenibilidad, etc [2].

Existen diversos trabajos con descripciones de refactorizaciones para diferentes paradigmas de programación, como imperativo [10], orientado a objetos [11] y orientado a aspectos [12]. Una contribución importante es el catálogo propuesto por Fowler et al. [5], con un conjunto de 72 refactorizaciones orientadas a objetos. Se reunió por primera vez una gran cantidad de refactorizaciones, las cuales están organizadas en categorías y se encuentran en un formato.

Existen pares de refactorizaciones que ejecutan modificaciones inversas en el código, como Extract Method e Inline Method ó Pull Up Field y Push Down Field [5]. Estas refactorizaciones son conocidas como refactorizaciones opuestas, y son muy importantes porque permiten a los desarrolladores deshacer fácilmente una refactorización cuando los resultados no son adecuados [13]. Cuando una refactorización no tenga una refactorización opuesta definida, ella podrá deshacerse, pero será un proceso más complicado y propenso a inserción de errores en el programa. Evaluar una refactorización que no tenga una refactorización opuesta permite conocer sus impactos en atributos de calidad del software antes de ser aplicados, y evita la posibilidad de introducir errores caso ella sea deshecha. En la sección 4 se describe un proceso para realizar dicha evaluación [7].

3. Patrones de diseño JEE

El movimiento de patrones de software se popularizó inicialmente gracias al libro de Gamma et al. [14]. Sin embargo, los patrones no se aplican únicamente en los dominios de micro-arquitectura. Los patrones pueden encontrarse en muchas áreas de software y sistemas. En un nivel superior, los patrones pueden encontrase al realizar análisis en campos específicos. Las entidades y relaciones descubiertas durante estos análisis se repetirán en el sector de la empresa. Esta repetición desemboca en el descubrimiento de dichos patrones. Los arquitectos de Sun Java Center han creado un conjunto de patrones específicos de Java basados en varios años de implementaciones prácticas de sistemas de base JEE. El conjunto de patrones se encuentra on-line en Java Developer Connection [15]. A continuación se detalla algunos de estos patrones que utilizaremos en este artículo.

3.1 Business Delegate

Utilizamos el patrón Business Delegate para reducir el acoplamiento entre los clientes de la capa de presentación y los servicios de negocio. El Business Delegate oculta los detalles de la implementación del servicio de negocio, como los detalles de búsqueda y acceso de la arquitectura EJB. La Figura 1 muestra la estructura de este patrón.

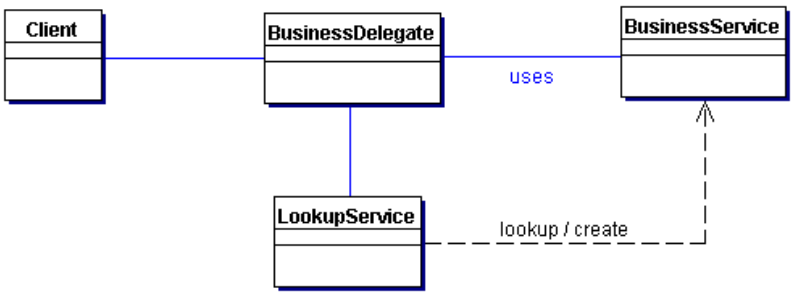

Figura 1. Estructura Business Delegate.

3.2 Session Facade

El patrón Facade para encapsular la complejidad de las interacciones entre los objetos de negocio participantes en un flujo de trabajo. El Session Facade maneja los objetos de

negocio y proporciona un servicio de acceso uniforme a los clientes. La Figura 2 muestra la estructura de este patrón.

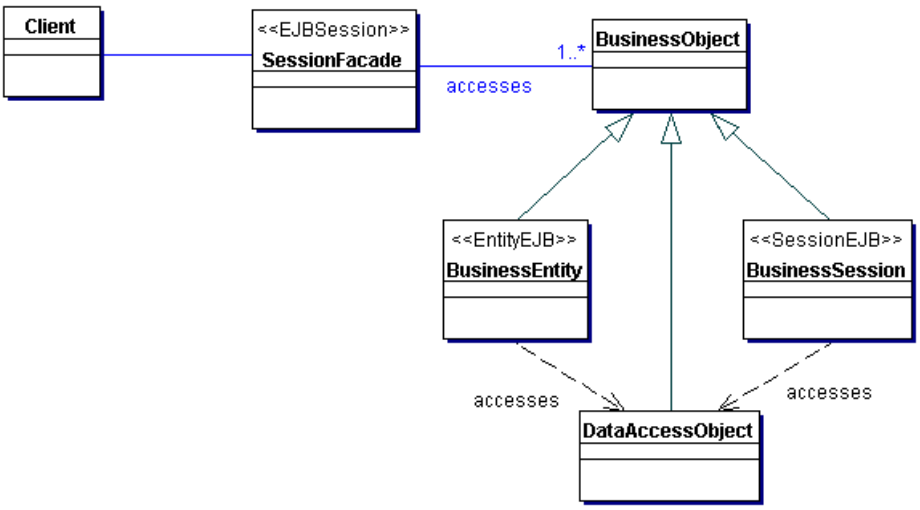

Figura 2. Estructura Session Facade.

3.3 Service Locator

Se utilizar Service Locator para abstraer toda la utilización JNDI y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y de re-creación de objetos EJB. Varios clientes pueden reutilizar el objeto Service Locator para reducir la complejidad del código, proporcionando un punto de control, y mejorando el rendimiento proporcionando facilidades de caché. La Figura 3 muestra la estructura de este patrón.

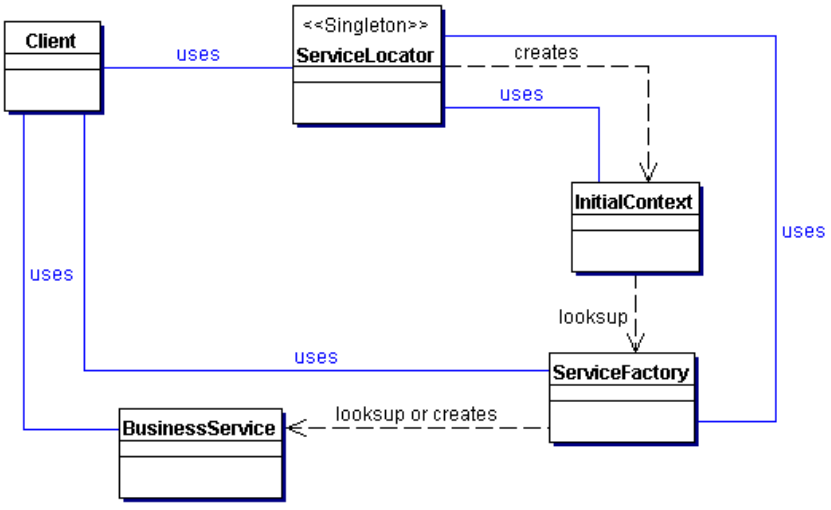

Figura 3. Estructura Service Locator.

4. Métricas

El proceso de refactorización no se resume en solo aplicar un conjunto de pasos en secuencia. Mens y Torwé [2] identifican otras cinco actividades dentro de este proceso, en el cual la aplicación de los pasos es apenas una etapa intermedia:

1) Identificar una oportunidad de refactorización.

2) Determinar que refactorizaciones deben ser aplicadas.

3) Garantizar que las refactorizaciones no alterarán el comportamiento del software.

4) Aplicar la refactorización.

5) Evaluar los efectos de la refactorización en características de calidad de software (complejidad, mantenibilidad, etc) o de proceso (productividad, costos, etc).

6) Mantener la consistencia entre el código refactorado y otros artefactos de software (documentación, especificación de requisitos, pruebas, etc.).

El proceso propuesto por Figueiredo [7] para evaluar cuantitativamente las refactorizaciones de software está relacionado a las actividades 2 y 5. La quinta, porque la aplicación del proceso permite obtener el impacto de las refactorizaciones en términos de calidad de software; y la segunda, pues en el caso de que la oportunidad de refactorización se obtenida a través de mediciones de atributos de calidad, será posible escoger las refactorizaciones a ser aplicadas tomando como base sus impactos en estos atributos.

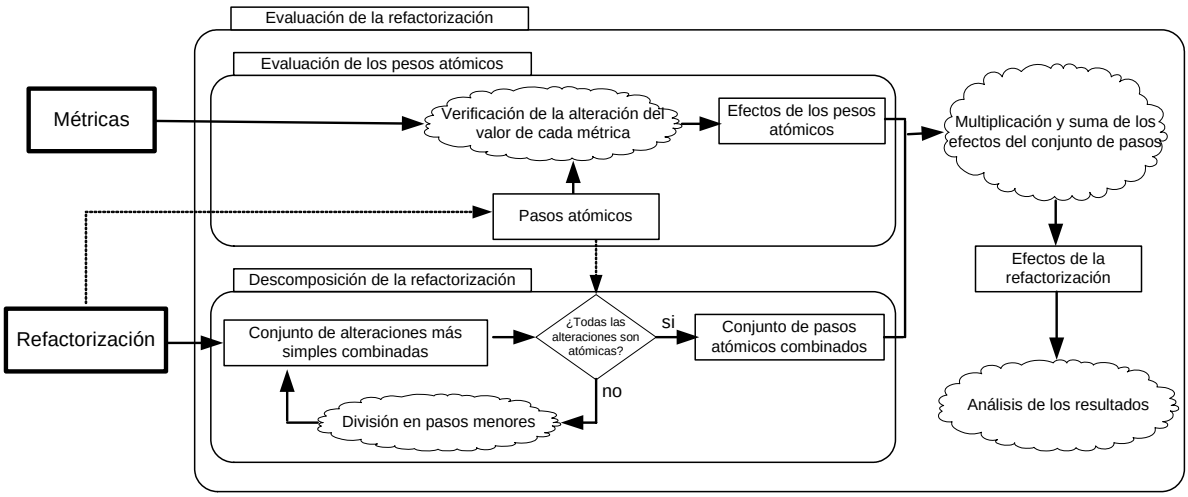

Figura 4. Proceso de evaluación.

El proceso de evaluación posee dos procesos menores que dividen la evaluación en dos partes: el análisis de los efectos de los pasos pequeños con el valor de las métricas y la descomposición de la refactorización en pasos cada vez menores.

Antes de iniciar la ejecución del proceso, es necesario definir la refactorización a ser evaluada y con cuales métricas será hecha la evaluación. Tanto la refactorización como las métricas pueden ser definidas por el propio evaluador o aprovechadas de otros trabajos.

5. Proceso Propuesto

En la actualidad existen una serie de técnicas y artefactos de diseño, pero se carece de un proceso que los integre y conduzca hacia un diseño reusable. Esta sección presenta una propuesta de un proceso de refactorización de software que permite aumentar el grado de reusabilidad de una aplicación Web.

El proceso se divide en dos etapas: refactorización a nivel de clases mediante las refactorizaciones propuestas por Martin Fowler [5], y refactorización a nivel de aplicación mediante la aplicación de patrones de diseño JEE (Java Enterprise Edition, anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE) [6]. En la Fig. 4 se grafica el proceso de manera informal; es decir, no sigue ninguna notación estándar. En dicho gráfico se muestra que las refactorizaciones a nivel de clase siguen los lineamientos que propone Martin Fowler. Seguidamente, se muestra que la refactorización a nivel de aplicación sigue los patrones de diseño JEE. Lo que resalta dicho esquema es que una vez una vez realizadas las refactorizaciones a nivel de clase, todas ellas intervienen en la refactorización a nivel de aplicación en mayor o menor grado.

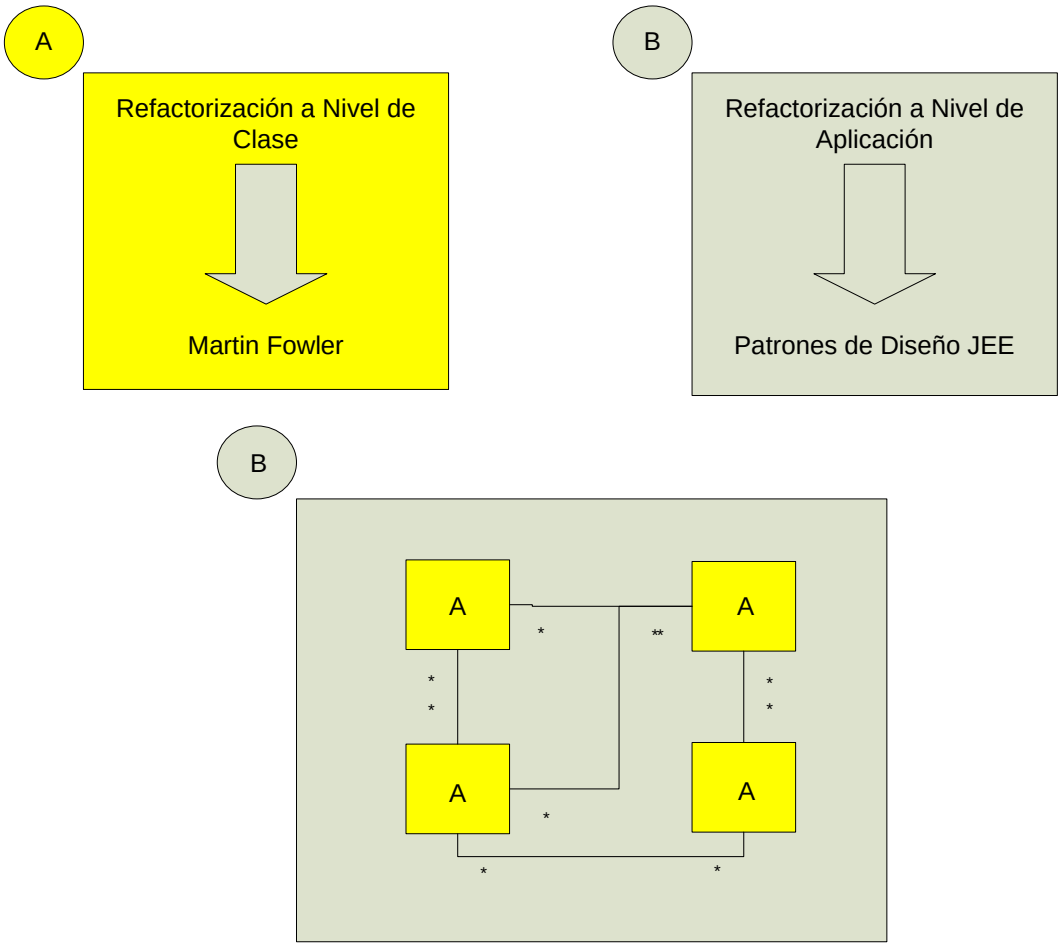

Figura 5. Proceso propuesto de refactorización.

A la vez, en la primera etapa se propone medir la refactorización mediante el proceso de evaluación cuantitativa de refactorizaciones de software propuesta por Figueiredo [7] y al final del proceso se propone la aplicación de las métricas de reutilización de Poulin [8].

A seguir se detallan cada una de las actividades del proceso propuesto:

A. Realizar Diagrama de Clases de Diseño de Lógica de Negocio: El proceso parte de la premisa de tener construido un diagrama de clases de diseño de la lógica de negocio de la aplicación. Para generar este diagrama se puede utilizar la metodología que se prefiera.

B. Realizar Diagrama de Clases de Diseño de acuerdo a la arquitectura elegida: Una vez obtenido el diagrama de clases de diseño correspondiente a la lógica de negocio, se procede a integrarlo con las clases que derivan de la arquitectura elegida para el desarrollo de la aplicación. En la actualidad está muy difundida la arquitectura Modelo Vista Controlador (MVC) con el *framework* Struts, Struts 2 y Java Server Faces (JSF).

C. Identificar oportunidades de aplicación de patrones de diseño: Como siguiente actividad dentro del proceso es la identificación de oportunidades de aplicación de patrones de diseño.

D. Determinar el tipo de patrón JEE a utilizar: Es necesario tener conocimiento de los patrones JEE y tener la capacidad de poder escoger un patrón de diseño que se aplique a la oportunidad identificada.

E. Actualizar Diagrama de Clases de Diseño: Una vez determinado el tipo de patrón a ser utilizado se procede a actualizar nuestro diagrama de clases.

F. Aplicar métricas de reuso: Un componente de software es reusado cuando es usado por una organización que no desarrolló y no presta mantenimiento al componente. El papel tradicional de la métrica es ayudar a la gestión para cuantificar el proceso del software. Con una tecnología emergente, sin embargo, las métricas deben extender más allá de su papel tradicional. Las métricas de reuso deben también fomentar la práctica de la reutilización. La mayoría de las organizaciones no práctica de manera formal la reutilización o se muestran reacios a invertir en un programa formal de la reutilización. Las métricas de reuso deben ayudar en el proceso de inserción de tecnología, proporcionar un proceso favorable que mejore las estadísticas de la organización y hacer hincapié en las actividades conducentes a la reutilización.

G. Implementar: Se procede a la implementación del diagrama de clases de diseño.

H. Aplicar Proceso de Evaluación Cuantitativa de Refactorización: En esta actividad, una vez obtenido el código de implementación se procede a aplicar el proceso de evaluación cuantitativa de refactorización enunciado en la sección 4.

I. Pruebas: Es importante realizar el tipo de pruebas correspondiente para asegurar que se está cumpliendo con los requisitos de la aplicación.

J. Mantener consistencia entre código y otros artefactos: Actualizar la trazabilidad entre el diseño y el código es requerido.

6. Caso Práctico

NDT [16] es una propuesta metodológica compuesta por un proceso en el que se plantean técnicas para capturar, describir y validar los requisitos de un sistema Web y, partiendo de esos requisitos, generar de manera sistemática los modelos de análisis del sistema.

Los procesos sistemáticos que se han comentado en el apartado anterior pueden ser implementados, consiguiéndose así una herramienta que permite aplicarlos de manera automática. Con este objetivo se ha desarrollado NDT-Tool. NDT-Tool no es más que una

herramienta en la que, los algoritmos y técnicas propuestas por NDT, han sido implementados con idea de que su aplicación sea más sencilla y con menos errores.

NDT-Tool está dividido en tres apartados: Gestión de proyectos, ingeniería de requisitos y modelos de análisis. El primer apartado es el que vamos a tomar como caso de estudio. A continuación se detalla su descripción:

Gestión de proyectos: esta opción permite dar de alta nuevos proyectos que vayan a ser desarrollados mediante NDT o modificar los datos de los ya existentes. Además, si el proyecto lo requiere, permite dar de alta y gestionar los datos de los participantes del proyecto, tanto equipo de desarrollo como grupos de usuarios y clientes, para posteriormente hacer un seguimiento del mismo.

NDT-Tool está desarrollado con Struts y acceso a la base de datos MySql mediante DAO. Lo que se pretende realizar en este trabajo es la implementación de dicho módulo de Gestión de Proyectos con JSF y acceso a la base de datos MySql mediante iBatis.

Durante el transcurso de este apartado se mostrará las actividades principales de la aplicación del proceso propuesto.

A. Realizar Diagrama de Clases de Diseño de Lógica de Negocio: Se construye el diagrama de clases básico. En la Figura 6 se muestra las clases de nuestro módulo. En este apartado se obvia la arquitectura a usar.

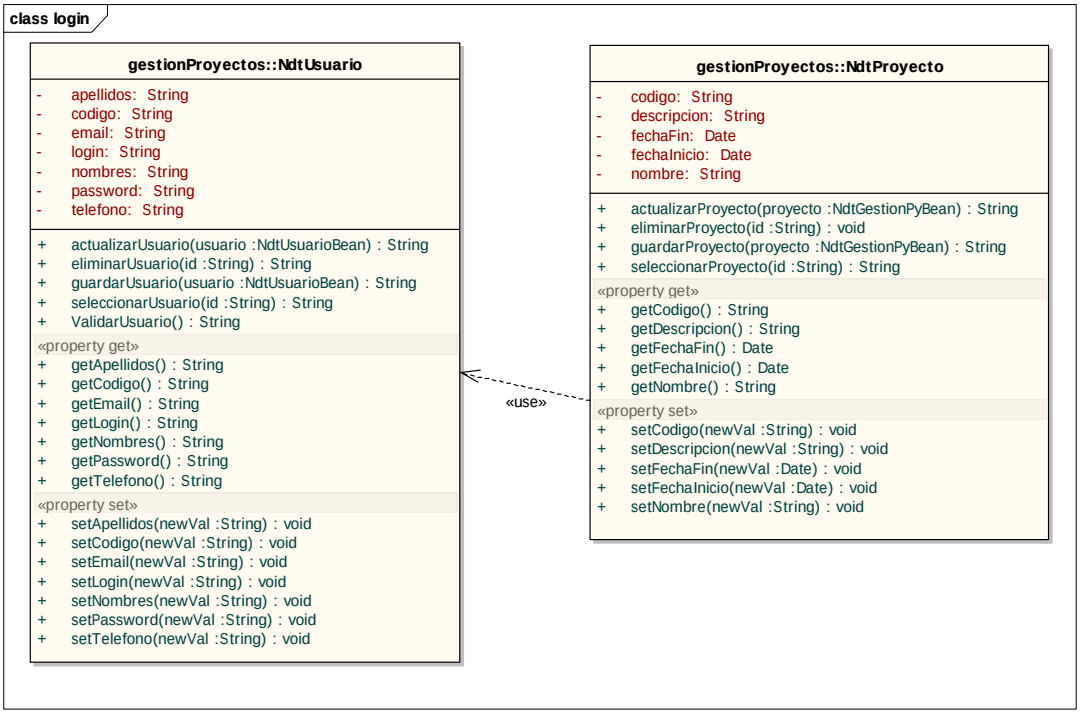

Figura 6. Diagrama de clases básico.

B. Realizar Diagrama de Clases de Diseño de acuerdo a la arquitectura elegida: En nuestro caso de estudio se ha decidido utilizar la arquitectura Modelo Vista Controlador (MVC) con el framework Java Server Faces (JSF) y como acceso a la capa de datos usaremos iBatis. En la sección VI se muestra las características y las ventajas de usarlos. En la Figura 7 se muestra nuestro diagrama de clases asociado con nuestras tecnologías utilizadas. Se ha añadido las clases Backing Bean, que son las que usa JSF para interactuar con la capa de presentación. Estas clases se encargar de invocar las reglas

de navegación que están especificadas en el archivo de configuración faces-config.xml. En el diagrama también tenemos la presencia de las clases Model, que son aquellas que invocan las funciones que acceden a la base de datos por medio de iBatis. Finalmente se muestran los bean, los cuáles van a ser nuestros contenedores auxiliares en la aplicación.

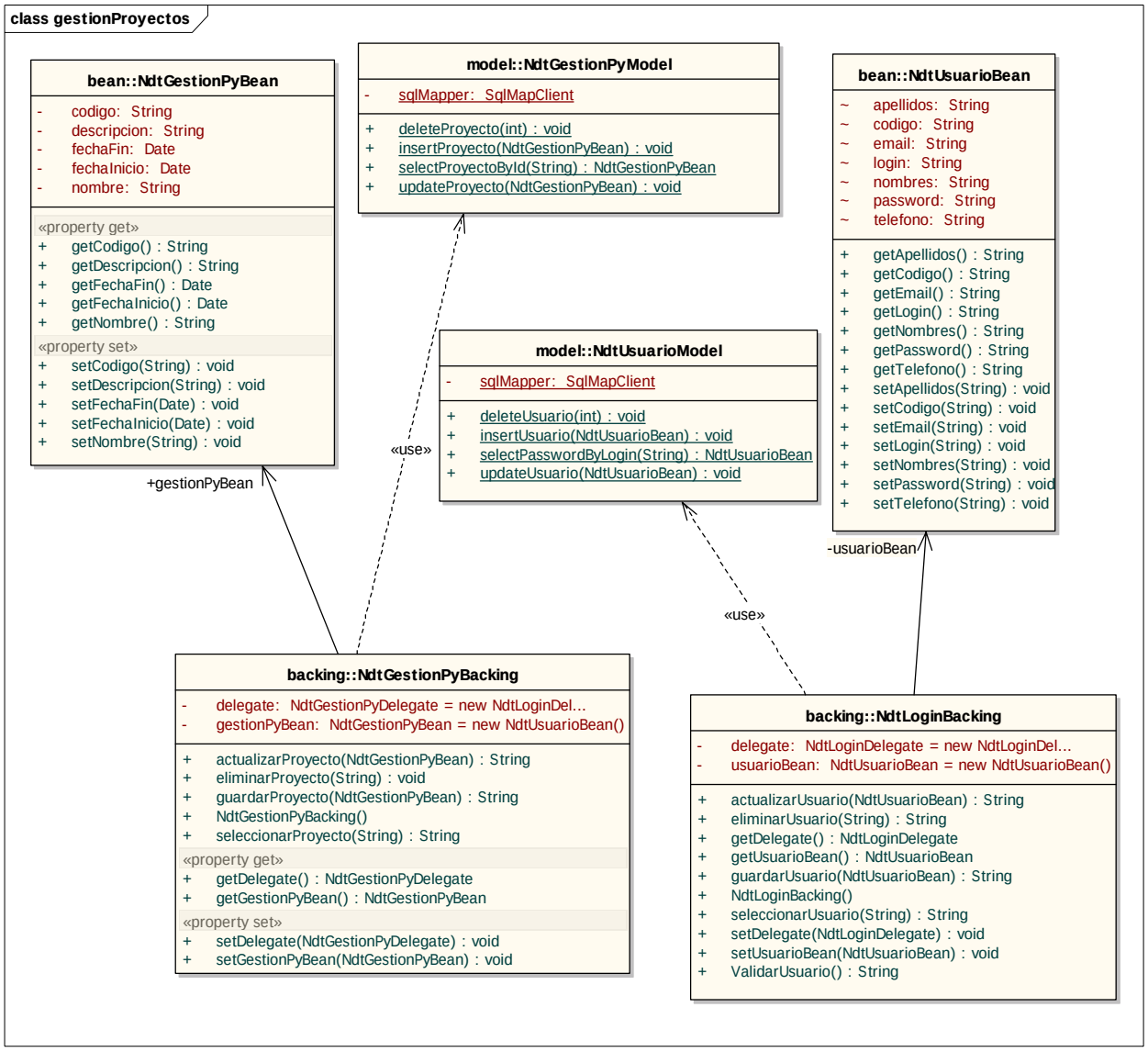

Figura 7. Diagrama de clases – Gestión de proyectos.

C. Identificar oportunidades de aplicación de patrones de diseño: Como siguiente actividad dentro del proceso es la identificación de oportunidades de aplicación de patrones de diseño.

D. Determinar el tipo de patrón JEE a utilizar: De acuerdo a nuestra actividad anterior, determinamos que los patrones a utilizar son el patrón Session Facade y el patrón Business Delegate.

E. Actualizar Diagrama de Clases de Diseño: Una vez determinado el tipo de patrón a ser utilizado se procede a actualizar nuestro diagrama de clases básico. Nuestro caso de

estudio lo estamos diseñando con la herramienta case Enterprise Architect [17]. A continuación en las Figuras 8 y 9 se muestra la aplicación de los patrones a nuestro caso de estudio.

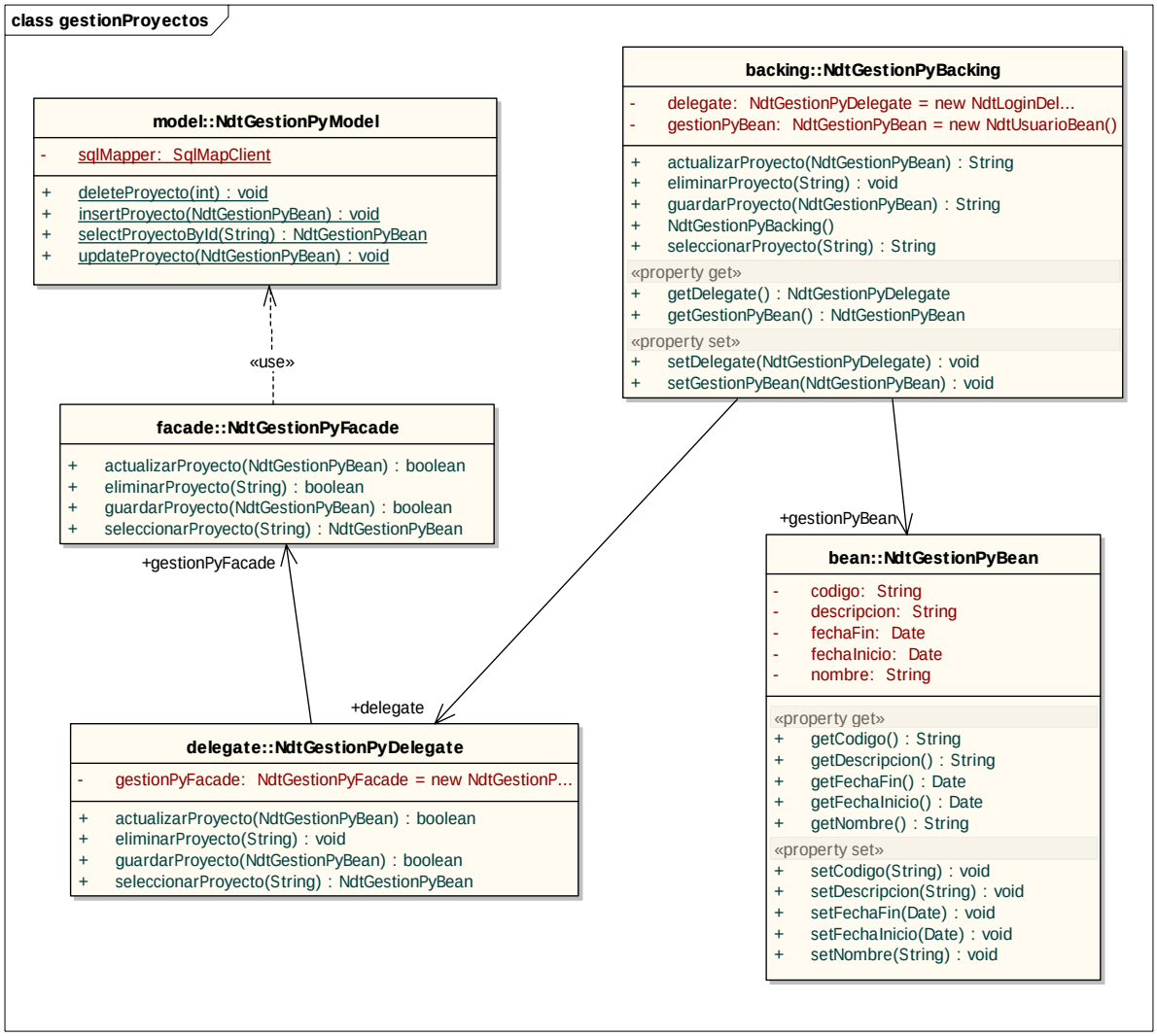

Figura 8. Diagrama de clases – Gestión de proyectos.

F. Aplicar métricas de reuso: No aplicado en este ejemplo.

G. Implementar: Se procede a la implementación del diagrama de clases de diseño.

H. Aplicar Proceso de Evaluación Cuantitativa de Refactorización: No aplicado en este ejemplo.

I. Pruebas: Es importante realizar el tipo de pruebas correspondiente para asegurar que se está cumpliendo con los requisitos de la aplicación.

J. Mantener consistencia entre código y otros artefactos: Actualizar la trazabilidad entre el diseño y el código.

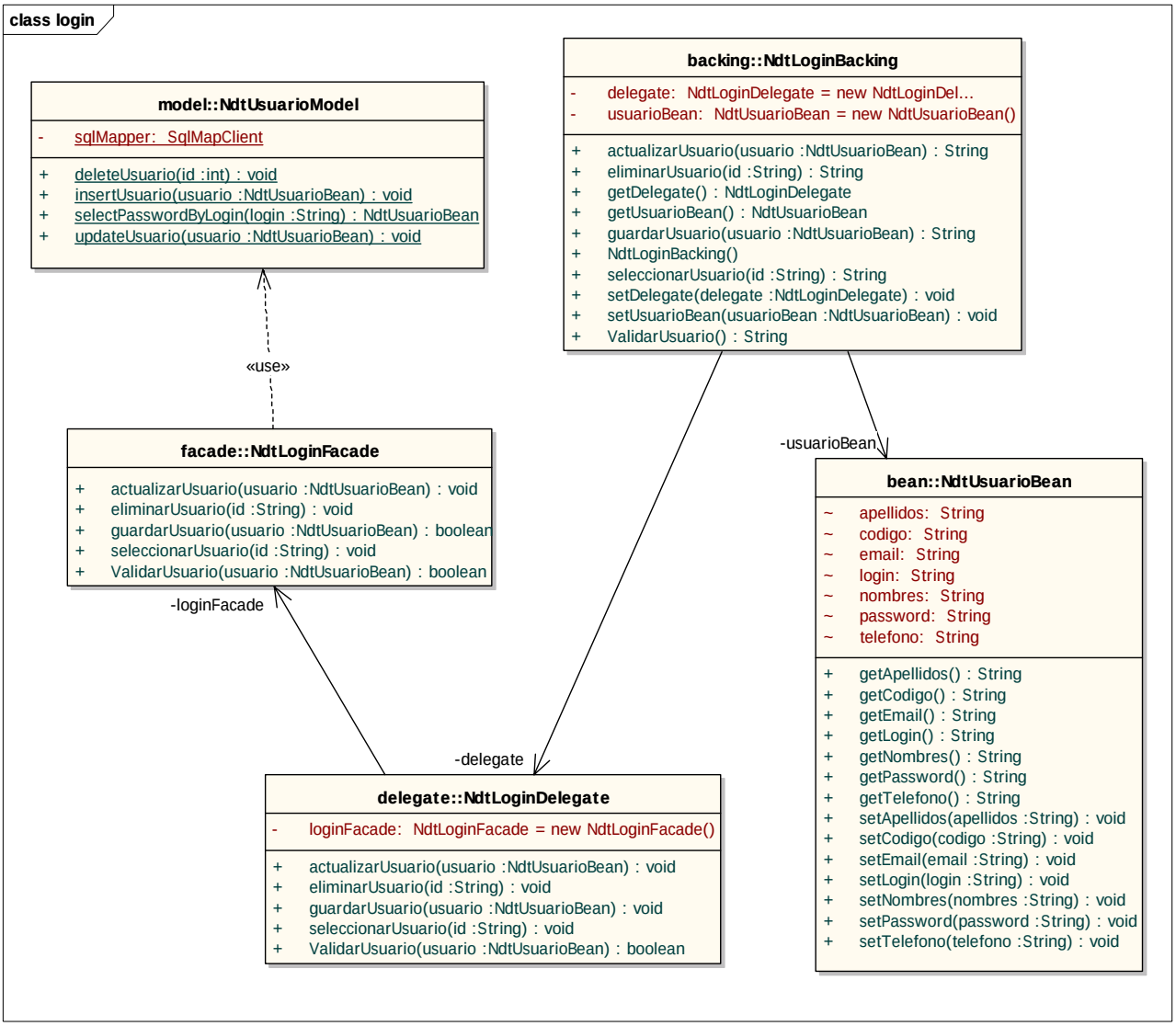

Figura 9 Diagrama de clases – Login.

7. Conclusión

Construir aplicaciones Web reusables es la motivación principal de este artículo. En la actualidad es indispensable contar con este tipo de procesos que permitan un mantenimiento eficiente. A pesar que existen técnicas aisladas con propósitos diferentes, hay la ausencia de procesos que los integren y orienten hacia un diseño de aplicaciones reusables. En este artículo se presentó una propuesta de un proceso de refactorización de software que permite aumentar el grado de reusabilidad de una aplicación Web. El proceso incluye el uso de patrones de diseño JEE, el uso de métodos de evaluación cuantitativa de refactorizaciones de software y de métricas de reuso. Para graficar la propuesta se presentó un caso práctico de aplicación de diseño de un módulo de gestión de proyectos para NDT. Como trabajos futuros está el mostrar de manera más detallada cómo se aplican las métricas de reuso y el proceso de evaluación cuantitativa.

Referencias

[1] L. Hatton, *Does OO Sync with how we think?.* IEEE Software, May/June 1998, pp. 46-54.

[2] T. Mens, and T. Tourw´e, (2004). *A survey of software refactoring.* IEEE Trans. Softw. Eng., 30(2):126–139.

[3] E. Gamma, R. Helm, R. Jonson, J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software* Addison Wesley, 1995.

[4] J. Poulin. *Measuring Software Reuse: Principles, Practices and Economics Models*, Addison-Wesley, 1997.

[5] M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts. *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

[6] S. Allmaraju, C. Beust, J. Davies, T. Jewell, R. Johson, A. Longshaw, R. Nagappan, D. O'Connor, P. Sarang, A. Toussaint, S. Tyagi, G. Watson, M. Wilcox, A. Williamson. Programación Java Server con J2EE Edición 1.3. Editorial Anaya, España 2001.L. Figueiredo and D. Nunes. *Um Processo para Avaliação Quantitativa de Refactoracoes de Software,* JPC'2007.

[7] J. Poulin and J. Caruso. A Reuse Metrics and Return on Investment Model.

[8] W. Opdyke. *Refactoring: A Program Restructuring Aid in Designing Object Oriented Application Frameworks*. Phd thesis, University of Illinois, Urbana-Champaign.

[9] A. Garrido and R. Johnson. *Challinges of refactoring c programas*. IWPE'02: Proceedings of the International Workshop on Principles of Software Evolution, pages 6-14, New York, USA. ACM Press, 2002.

[10] J Kerievsky. *Refactoring to Patterns*. Pearson Higher Education. 2004.

[11] M. Monteiro and J. Fernandes. *Towards a catalog of aspect-oriented refactorings*. Proceedings of the 4th international conference on Aspect-oriented software development (AOSD'05), pages 111-122, New York, USA, ACM Press. 2005.

[12] M. Monteiro. Refactorings to Evolve Object-Oriented Systems with Aspect-Oriented Concepts. Phd thesis, Universidade do Minho, 2005.

[13] http://java.sun.com/developer/onlineTraining/

[14] J Kerievsky. Refactoring to Patterns. Pearson Higher Education. 2004.

[15] M. Monteiro and J. Fernandes. Towards a catalog of aspect-oriented refactorings. Proceedings of the 4th international conference on Aspect-oriented software development (AOSD'05), pages 111-122, New York, USA, ACM Press. 2005.

[16] M. Monteiro. Refactorings to Evolve Object-Oriented Systems with Aspect-Oriented Concepts. Phd thesis, Universidade do Minho, 2005.

[17] E. Gamma, R. Helm, R. Johnson , J. M. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley.

[18] http://java.sun.com/developer/onlineTraining/

[19] M. J. Escalona. Modelos y técnicas para la especificación y el análisis de la navegación en sistemas software. PHD Thesis. Escuela Técnica Superior de Ingeniería Informática. Universidad de Sevilla, 2004.

[20] Enterprise Architect, http://www.sparxsystems.com.au/.

Correspondencia (Para más información contacte con):
Arturo Henry Torres Zenteno