Código de calidad: Integrando Patrones de diseño y Refactoring *

Emilio A. Sánchez, Patricio Letelier, and José H. Canós

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia - España
{emsanchez|letelier|jhcanos}@dsic.upv.es

Resumen La automatización en la generación de código es clave para conseguir un incremento en la calidad y productividad en el desarrollo de software. Sin embargo, todavía estamos lejos de una automatización total y sigue siendo necesario manipular el código más de lo deseable. El mantenimiento del código es aún más crítico cuando se utiliza un modelo de proceso iterativo e incremental. Es de vital importancia que este código presente una estructura sencilla para facilitar su mantenimiento. Con el fin de conseguir código de calidad encontramos dos técnicas que abordan el problema desde puntos distintos. Los patrones de diseño identifican situaciones de diseño clásicas que aparecen en la mayoría de desarrollos y les asocian una solución probadamente adecuada. Por otra parte, desde el mundo de las metodologías ágiles, el refactoring apuesta por detectar deficiencias en código existente y aplicar las transformaciones necesarias para obtener código que presenta idéntica funcionalidad pero con un mejor diseño. Este trabajo es el comienzo de una nueva línea de investigación que apuesta por abordar el problema uniendo lo mejor de estos dos enfoques apoyándose en la automatización.

1. Introducción

Uno de los problemas más importantes en el desarrollo de sistemas de información es el de conseguir código de calidad; esto es, aquél que cumple las propiedades de claridad y sencillez en estructura y diseño. Tal código es fácil de leer por otros desarrolladores y por tanto sencillo de mantener. Un código que por el contrario tenga un diseño deficiente, que presente, por ejemplo, funcionalidad duplicada en diferentes partes del mismo, es código difícil de mantener y una fuente de errores.

Consiguiendo código con un mejor diseño se evita la futura aparición de problemas durante el desarrollo, consiguiendo agilizar el proceso. Dos han sido las aproximaciones más importantes realizadas en este sentido. Por una parte intentando aprovechar la experiencia ante situaciones de desarrollo clásicas aparecen

* Este trabajo ha sido financiado por el proyecto DOLMEN-SIGLO de la Comisión Interministerial de Ciencia y Tecnología, TIC2000-1673-C06-01.

los patrones de diseño que, ante una situación clásica en el diseño de software, asocian un patrón de código extensamente utilizado y probado y al cual se le conocen las propiedades de calidad.

Mientras que los patrones de diseño apoyan al desarrollador mientras este crea el código del sistema, la otra aproximación aborda el problema desde un punto diametralmente opuesto. Partiendo de código existente se le aplican técnicas de transformación con el fin de obtener un nuevo código que no modifica en nada la funcionalidad del sistema pero que presenta un mejor diseño. Esta práctica se conoce como refactoring y proviene de la nueva corriente de procesos software conocida como metodologías ágiles [1].

Resulta interesante establecer un marco de estudio que aborde la utilización de las dos prácticas durante el proceso de desarrollo con el fin de obtener código de alta calidad durante el desarrollo de nuevos sistemas o durante el mantenimiento de sistemas legados. Hasta el momento no se ha realizado ninguna aproximación que una estas dos técnicas y el apoyo ofrecido por las herramientas para cada una de ellas por separado deja excesivo trabajo al desarrollador. Por tanto sería deseable plasmar las virtudes de los dos mundos en un marco de herramientas que apoyen automáticamente dicho trabajo.

2. Patrones de diseño

El concepto de patrón tiene su origen en los trabajos realizados por Christopher Alexander dentro el ámbito de la arquitectura [2][3]. Alexander afirma que "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma". Esta misma noción de patrón está presente en el desarrollo de software.

Los patrones de diseño permiten reutilizar la experiencia de los desarrolladores describiendo formas de solucionar problemas que se presentan con frecuencia durante el desarrollo de software. Erich Gamma [7], con el fin de documentar esta experiencia de forma que sea fácilmente reutilizable, propone una plantilla para definir patrones. Básicamente esta plantilla contiene, entre otra, la siguiente información.

- La motivación y el contexto de aplicación del patrón.
- Prerequisitos que se deben satisfacer antes de utilizar el patrón.
- Una descripción de la estructura del programa que define el patrón.
- Lista de los participantes que intervienen.
- Consecuencias (positivas y negativas) de la utilización del patrón.
- Ejemplos.

Algunos autores [10] recomiendan la utilización de patrones de diseño basándose en los siguientes valores:

- **El éxito es más importante que la novedad.** Cuanto más se ha utilizado un patrón, más valioso es el mismo ya que esto demuestra su eficacia. Por otra parte la novedad aunque puede parecer una solución eficaz tiene el inconveniente de no haber sido probada con anterioridad.
- **Énfasis en la escritura y claridad de comunicación.** Utilizar un formato estándar para describir los patrones favorece la comunicación y la creación de manuales para ingenieros de software.
- **Validación cualitativa del conocimiento.** Los patrones pretenden describir de forma cualitativa soluciones concretas a problemas software y no cuantificarlos o establecer teorías sobre ellos. El objetivo es reconocer y premiar el proceso creativo que los desarrolladores expertos utilizan para crear sistemas software de alta calidad.
- **Los buenos patrones se obtienen de la experiencia.** Cualquier desarrollador con experiencia posee un valioso conjunto de patrones que sería interesante compartir.
- **Reconocer la importancia del factor humano en el desarrollo de software.** No es objetivo de los patrones reemplazar la creatividad de los desarrolladores, ni reemplazarlos por herramientas CASE automáticas. Al contrario, se reconoce la importancia del factor humano a la hora de crear y utilizar estos patrones como medio para llegar a comprender sistemas software complejos.

3. Refactoring

Las metodologías ágiles representan una nueva propuesta de procesos de desarrollo de software que está teniendo un fuerte impacto en el ámbito industrial y académico. Apuestan por un desarrollo de sistemas de información más veloz y fácil de adecuar a los cambios en los requisitos. Para conseguir estos objetivos las llamadas metodologías ágiles restan protagonismo a una serie de prácticas presentes en las metodologías tradicionales como etapas de modelado, documentación, etc.

Sin lugar a dudas el máximo exponente hasta el momento de esta corriente metodológica es XP (eXtreme Programming), propuesta por Kent Beck [4]. Esta metodología se centra principalmente en el código al tiempo que rodea a la programación de una serie de prácticas que intentan asegurar el éxito del proceso de desarrollo. Prácticas como Programación en Parejas, Pruebas, Refactoring o Diseño Simple son sólo algunas de las doce prácticas que conforman la metodología.

A pesar de todo, y aun siendo XP la metodología ágil sobre la que se dispone de mayor cantidad de información, estas prácticas se encuentran descritas muy vagamente y existen muchas dudas sobre las mejores formas de aplicar cada una de ellas en un proceso de desarrollo real, en qué medida afectan al proceso de desarrollo o si resulta posible descartar algunas de ellas sin perjudicar el éxito del proyecto. Ante esta situación, muchos son los trabajos que intentan validar, extender o aprovechar algunas de las características presentes en XP.

Así encontramos desde trabajos que intentan mejorar algunos elementos como las historias de usuario [5][9] hasta trabajos que enfrentan y ponen a prueba la metodología frente a metodologías tradicionales [8][11].

Una de las prácticas que más interés suscita es el Refactoring. Beck define el refactoring como la reorganización o reprogramación del código que no altera el funcionamiento del sistema sino que mejora otras propiedades no funcionales del mismo como son la legibilidad del código o la simplicidad del diseño. Martin Fowler [6] realiza un estudio sobre las técnicas de refactoring dando patrones de modificación de código con el fin de obtener un resultado con la misma funcionalidad pero con un diseño más simple y sencillo de comprender. El problema es que esta tarea de se hace siempre de forma manual, en ocasiones con alguna ayuda del entorno de programación. Es el programador quien debe averiguar cuándo se deben aplicar técnicas de refactoring sobre el código, en qué partes del mismo y qué técnica concreta de refactoring utilizar en cada caso.

4. Objetivos

Dado que existen dos aproximaciones distintas al problema de generar código de calidad y que estas aproximaciones proponen soluciones que no se solapan entre ellas, resulta interesante plantearse la posibilidad de utilizar ambas durante el proceso de desarrollo. Para ello se establecerá un marco de trabajo, en el ámbito de una metodología de desarrollo ágil como es XP, en el que se aprovechen las mejores cualidades de los dos enfoques.

La principal razón del complejo y mal diseño del código en las aplicaciones actuales es que el proceso de refactoring es demasiado costoso en horas de desarrollo. Por tanto se tiende a dejar de lado esta práctica perdiendo todas las ventajas que ella conlleva. Se propone un estudio preciso de las herramientas existentes que apoyan tanto la actividad de refactoring como la programación utilizando patrones de diseño con el fin de validar la viabilidad de integración de algunas de estas herramientas, extensión de las mismas o, si fuese necesario, creación de herramientas adicionales.

Disponer de un entorno de desarrollo que combinase patrones de diseño y refactoring permitiría:

- Desarrollar nuevo código de forma más rápida, con más confianza y mejor calidad gracias a los patrones de diseño.
- El código existente requerirá menos esfuerzo en refactoring por tratarse de código generado a partir de patrones.
- Aún utilizando patrones puede ser necesario mejorar el código existente, las técnicas de refactoring permiten hacerlo de forma ordenada.
- La automatización de ambas tareas agiliza el desarrollo.
- Una herramienta capaz de detectar situación de refactoring o patrones de diseño descarga de gran parte de trabajo al programador.
- Al aplicar refactoring sobre código generado por patrones pueden surgir nuevos patrones de diseño e incluso detectar fallos en los antiguos.

Se plantea como objetivo a medio plazo obtener conclusiones sobre la viabilidad de utilizar estas dos aproximaciones en un mismo proceso de desarrollo y las ventajas o inconvenientes que ello conlleva. Se desea, también, recopilar un catálogo de técnicas de refactoring y patrones de diseño que puedan ser automatizadas posteriormente, al tiempo que se realiza un estudio de las herramientas existentes con el fin de evaluar el grado de automatización ofrecido y el número de técnicas soportadas.

A largo plazo se marca como objetivo la consecución de un entorno de desarrollo que permita tanto la programación basada en patrones de diseño como la posibilidad de aplicar técnicas de refactoring al código existente. Dado que en la actualidad la utilización de patrones de diseño y de técnicas de refactoring se realiza de forma manual principalmente, es deseable que la futura herramienta introduzca automatización en estas tareas, proponiendo al desarrollador patrones y técnicas de refactoring a aplicar en cada caso. Esto permitiría desarrollar aplicaciones en menos tiempo, con un código mucho más simple e legible y por lo tanto más sencillo de mantener y con menor probabilidad de contener errores.

Referencias

1. Agile Alliance Web Site on-line at http://www.agilealliance.org/
2. Alexander C., Ishikawa S., Silverstein M. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977.
3. Alexander C. The Timeless Way of Building. Oxford University Press, 1979.
4. Beck K. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.
5. Breitman, K. y Leite, J.. Managing User Stories. International Workshop on Time-Constrained Requirements Engineering 2002
6. Fowler M., Beck K., Brant J., Opdyke W., Roberts D. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.
7. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns Elements of Reusable Object-Oriented Software Addison-Wesley, 1994.
8. Letelier P., Canós, J.H. and Sánchez E.A. An Experiment Comparing RUP and XP. Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering, XP2003, Génova, pp. 41-46, LNCS 2675, Springer-Verlag, 2003.
9. Sánchez E.A., Letelier P. and Canós J.H. Mejorando la gestión de historias de usuario en eXtreme Programming. VIII Jornadas de Ingeniería del Software y Bases de Datos, 2003. (Aceptado)
10. Schmidt D.C., Johnson R.E., Fayad M. Software Patterns. Communications of the ACM, Special Issue on Patterns and Pattern Languages, Vol. 39, No. 10, October 1996.
11. Smith J. A Comparison of RUP and XP. Rational Software White Paper, página Web: `www.rational.com/media/whitepapers/TP167.pdf`