Descubrimiento de patrones de diseño basado en buenas prácticas: modelo y discusión

Rafael Barbudo, Aurora Ramírez, José Raúl Romero y Sebastián Ventura

Dpto. de Informática y Análisis Numérico, Universidad de Córdoba**
Campus de Rabanales, 14071 Córdoba, España
{rbarbudo, aramirez, jrromero, sventura}@uco.es

Resumen La complejidad de los sistemas actuales obliga a los ingenieros software a aprender de las buenas prácticas empleadas en proyectos previos como, por ejemplo, el uso de patrones de diseño. Dichos patrones tienen una gran importancia durante la fase de diseño y su posterior implementación genera varias ventajas. En este contexto, se propone el uso de técnicas de minería de datos que ayuden a comprender como otros ingenieros software han implementado dichos patrones. Con la representación adecuada, este conocimiento se podrá utilizar para identificar fragmentos de código susceptibles de ser convertidos en un determinado patrón. Además del modelo, se discuten ventajas y retos asociados.

Keywords: Minería de repositorios software, buenas prácticas, patrones de diseño, programación genética gramatical.

1. Introducción

El desarrollo de un sistema software es un proceso complejo cuyo éxito está fuertemente condicionado por las habilidades de los participantes del proyecto. Es por ello que aprender de la experiencia de proyectos anteriores resulta de especial interés, pues va a permitir al equipo adoptar buenas prácticas. Una de ellas consiste en la incorporación de patrones de diseño, ya que son soluciones de código efectivas y reutilizables que resuelven un problema de diseño en un contexto determinado. Identificar dónde debe implementarse un patrón o cómo el código existente puede adaptarse para acercarse a él son tareas que requieren conocer cuáles son las propiedades que mejor caracterizan al patrón y a sus posibles variantes. Por ello, existen propuestas que, a partir de propiedades estructurales, morfológicas y semánticas definidas por el experto, permiten identificar antipatrones en el código como paso previo a la recomendación [1].

Una alternativa interesante consiste en tratar de inferir esta información de manera automática. Sin embargo, extraer cualquier tipo conocimiento en el ámbito del desarrollo software constituye un reto en sí mismo, pues va a estar oculto entre miles de líneas de código y en otros artefactos software. En este contexto

** Trabajo financiado por el Ministerio de Economía y Competitivad, proyecto TIN2014-55252-P y el Ministerio de Educación, programa FPU (FPU13/01466).

surge la minería de repositorios software [2] (MRS), cuyo objetivo es descubrir información interesante y previamente desconocida a partir de los recursos y plataformas para la gestión y configuración del software, incluido el código. Para ello se sirve de técnicas de minería de datos y aprendizaje automático, como la minería de patrones frecuentes (PF) [3]. En el contexto de los patrones de diseño, las propuestas de MRS se han centrado principalmente en detectar patrones ya existentes bien mediante medidas software [4] o a partir de su morfología [5].

Sin embargo, la idoneidad de un patrón de diseño puede también venir dada por aspectos que, aún estando presentes en el código (relaciones en el diseño, nomenclatura, etc.), podrían no ser directamente capturados por métricas software. Además, al tratase de soluciones flexibles, la morfología de un patrón podría variar sensiblemente si el programador requiriese adaptarlo a ciertas particularidades del proyecto. Con esta idea, este trabajo propone un modelo de MRS que, adquiriendo una base de conocimiento extraída de repositorios software reales sobre el buen uso previo de patrones de diseño, sea capaz de descubrir aquellas partes del código–actualmente en desarrollo–en las que resultara apropiado adoptar un patrón concreto. El modelo se compone de tres fases: (1) búsqueda de PF aplicando un algoritmo basado en programación genética gramatical (*Grammar Guided Genetic Programming*, G3P), para extraer y caracterizar el conocimiento acerca de cómo un tipo de patrón de diseño está siendo llevado históricamente a la práctica, (2) identificación de estructuras de código potencialmente adecuadas para un patrón de diseño; y (3) análisis de la información extraída para el descubrimiento de oportunidades.

2. Marco conceptual

La MRS permite aprender de las prácticas realizadas en proyectos software anteriores. Para ello, la datos son extraídos utilizando diversas fuentes, como repositorios software, documentos y artefactos software, foros de programadores o *bug trackers*, entre otros. Estas fuentes se analizan para descubrir aspectos escondidos en esos datos pero relevantes al proceso de la ingeniería del software. Técnicas como clasificación, asociación, regresión o *clustering* son habituales.

Más específicamente, la minería de PF tiene carácter descriptivo, pues busca descubrir conjuntos de ítems (*itemsets*) que ocurren con una determinada frecuencia (llamada soporte) en el conjunto de datos. La estructura de un *itemset* viene definida conforme a una gramática de contexto libre (GCL), cuyos símbolos representan el conjunto de atributos y operadores que caracterizan el código. Estos operadores se pueden adaptar para trabajar con diferentes lenguajes. En general, cada ítem se representa mediante un predicado del tipo <*atributo*><*operador*><*valor*>, que puede tener naturaleza numérica (LOC>10K) o categórica (X! =*class*). Estos son unidos mediante operadores lógicos para formar los *itemsets*. Bajo el paradigma G3P, este tipo de gramática ya se ha utilizado para extraer reglas en minería de asociación [6] o para la predicción de costes en proyectos software [7], demostrando la flexibilidad que ofrece una GCL para modelar y representar el conocimiento en diferentes dominios.

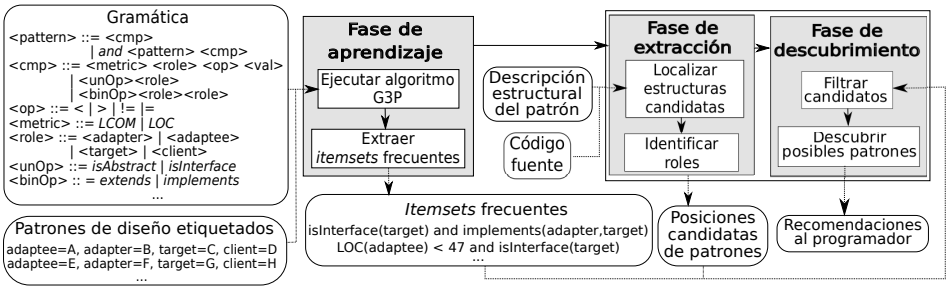

Figura 1. Visión general del modelo

3. Modelo de descubrimiento de patrones

La Figura 1 muestra las fases del modelo propuesto, incluyendo sus entradas y salidas. Inicialmente se aplica la *fase de aprendizaje* sobre un repositorio de código que contiene múltiples desarrollos de un patrón de diseño dado, y en el que son conocidos los roles que implementa cada clase o interfaz[1]. Para ello se ejecuta un algoritmo G3P [6], adaptado para la búsqueda de *itemsets* conformes a la GCL dada. Por ejemplo, considerando el patrón *Adapter*, la gramática representa el lenguaje que declarará los operadores necesarios para definir su estructura (p.ej. una clase abstracta o la implementación de una interfaz). Pueden también incluirse métricas software de interés. Como resultado, se obtiene el conjunto de características asociadas al patrón en el uso histórico. En el ejemplo mostrado sobre el patrón *Adapter* se observa, entre otros muchos aspectos descubiertos por el conjunto de *itemsets*, como el rol *adapter* se desarrolla en una clase, mientras que el rol *target* lo implementa una interfaz. Además, el rol *adaptee* suele ser implementado por una clase con pocas líneas de código (LOC<47).

Ya sobre el código en desarrollo se lleva a cabo la *fase de extracción*, donde se localizan fragmentos con una estructura aproximada a la del patrón con el objetivo de localizar posiciones candidatas en las que éste podría ser aplicado. Considérese que la definición estructural de un patrón es previamente conocida. Por tanto, se tratará de asociar cada elemento del conjunto candidato con un posible rol, excluyendo a aquellos que incumplan las restricciones del patrón o se desvíen en exceso. En el caso del patrón *Adapter*, se buscarían estructuras formadas por cuatro elementos, tanto clases como interfaces, relacionados entre sí de manera que puedan desempeñar los roles definidos por su especificación (*adaptee*, *adapter*, *target* y *client*). Para la realización de esta fase se aplican técnicas de análisis de grafos [4,5].

Finalmente, sobre posiciones estructuralmente candidatas, la *fase de descubrimiento* aplica el resto de aspectos extraídos en fases anteriores para recomendar en qué partes del código puede incorporarse un patrón de diseño. Así pues, aquellas estructuras candidatas del código fuente que no presenten las características definidas por los *itemset* frecuentes son descartadas. Además, se

[1] Se ha utilizado P-MARt: `http://www.ptidej.net/tools/designpatterns`

analizará la similitud con las estructuras que cumplan dichas características en mayor o menor medida, constituyendo los fragmentos de código susceptibles de ser transformados para implementar convenientemente el patrón de diseño. En definitiva, el conocimiento extraído del uso de buenas prácticas en proyectos anteriores se utiliza de base para recomendar de forma comprensible la adaptación del nuevo código a un determinado patrón de diseño.

4. Discusión y temas abiertos

El modelo propuesto combina diferentes tipos de técnicas para conseguir un proceso de soporte al programador más completo que las propuestas existentes. Nótese que los métodos actuales tienden a centrarse en un número limitado de patrones. Además, se focalizan bien en estructuras estrictas para el patrón o bien en aquellas ajustadas a unos valores de métricas software. En este sentido, el uso de una GCL dota de mayor flexibilidad al proceso de aprendizaje al permitir incorporar simultáneamente atributos y operadores de distinta naturaleza.

Uno de los puntos más complejos es la correcta identificación de los fragmentos de código candidatos a ser convertidos en un patrón. Por un lado, es necesario abstraer el nuevo código fuente de su nomenclatura y, en cierta medida, también de su estructura para poder compararla con la del patrón. Esto es aún más complejo al tener presente que los tipos de patrones son muy diferentes entre sí. Por ello, para abstraerlo es importante establecer criterios de similitud que acepten desviaciones respecto del patrón original. Por otro lado, la fase de descubrimiento requiere el estudio de las medidas de interés (y sus umbrales) más apropiadas a la hora de filtrar las estructuras candidatas en base a los *itemset*.

Referencias

1. N. Nahar and K. Sakib, "Automatic recommendation of software design patterns using anti-patterns in the design phase: A case study on abstract factory.," in *3rd Int. Workshop on Quantitative Approaches to Software Quality*, pp. 9–16, 2015.
2. H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *J. Softw. Maint. Evol.*, vol. 19, no. 2, pp. 77–131, 2007.
3. J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data Min. Knowl. Discov.*, vol. 15, no. 1, pp. 55–86, 2007.
4. A. Chihada, S. Jalili, S. M. H. Hasheminejad, and M. H. Zangooei, "Source code and design conformance, design pattern detection from source code by classification approach," *Appl. Soft Comput.*, vol. 26, pp. 357–367, 2015.
5. B. B. Mayvan and A. Rasoolzadegan, "Design pattern detection based on the graph theory," *Knowl-Based Syst.*, vol. 120, pp. 211–225, 2017.
6. J. M. Luna, J. R. Romero, and S. Ventura, "Design and behavior study of a grammar-guided genetic programming algorithm for mining association rules," *Knowl. Inf. Syst.*, vol. 32, no. 1, pp. 53–76, 2012.
7. Y. Shan, R. I. McKay, C. J. Lokan, and D. L. Essam, "Software project effort estimation using genetic programming," in *IEEE Int. Conf. on Communications, Circuits and Systems and West Sino Expositions*, vol. 2, pp. 1108–1112, 2002.