Análisis de código fuente orientado a la calidad del producto

Dapozo, Gladys; Greiner, Cristina; Irrazabal, Emanuel; Chiapello, Jorge

Departamento de Informática. Facultad de Ciencias Exactas y Naturales y Agrimensura. Universidad Nacional del Nordeste
{gndapozo; cgreiner}@exa.unne.edu.ar¸ emanuelirrazabal@gmail.com; jorgechiapello@gmail.com

Resumen

Ante un panorama en el cual los productos de software seguirán creciendo en complejidad y tamaño, la tarea de asegurar la calidad del mismo y su confiabilidad se hace cada vez más difícil. La calidad del producto a largo plazo depende en gran medida del nivel de calidad de las actividades relacionadas con el desarrollo de software si no se realizan con los niveles de calidad adecuados. Para afrontar esta situación, se propone la elaboración de buenas prácticas para prevenir, medir y gestionar la deuda técnica en el desarrollo de software, y una herramienta que permita, a través de un repositorio de métricas de software y reportes de errores, relacionar los mismos para detectar clases o paquetes en donde sea más probable encontrar errores o propensión a fallas.

Palabras clave: Deuda técnica, métricas de software, predicción de fallos.

Contexto

La línea de I/D presentada forma parte del proyecto F010-2013: "Métodos y herramientas para la calidad del software", acreditado por la Secretaría General de Ciencia y Técnica de la Universidad Nacional del Nordeste (UNNE) para el periodo 2014-2017.

Introducción

Un "mal olor" en el código (*code smell* en inglés) es cualquier síntoma en el código fuente de un programa que posiblemente indica un problema más profundo. Usualmente no son errores de programación, es decir, no son técnicamente incorrectos y no impiden que el programa funcione correctamente. Sin embargo, indican deficiencias en su construcción que pueden disminuir la velocidad del desarrollo o aumentar el riesgo de errores o fallos en el futuro [1].

La situación en la cual la calidad del código fuente a largo plazo se ve disminuida por las ganancias a corto plazo fue dada a conocer como la metáfora de "deuda técnica" (technical debt) por Ward Cunningham, quien la menciona por primera vez en el congreso de orientación a objetos OOPSLA en el año 1992 [2].

Así, la deuda técnica es aquella que se contrae cuando las actividades relacionadas con el desarrollo de software no se realizan con los niveles de calidad adecuados y, por lo tanto, pueden disminuir la mantenibilidad del código fuente u ocasionar mayores costos en el proyecto. Por ejemplo, cuando el programador deja de incorporar comentarios en el código fuente y, pasado un tiempo, necesita modificarlo. El código fuente resultará más difícil de ser analizado y requerirá invertir mayor tiempo y esfuerzo. De forma similar a las

deudas financieras, se generan intereses, haciendo que los problemas de calidad en el código fuente vayan aumentando conforme pasa el tiempo. Por otra parte, una de las Leyes de Lehman [3] estima que el software seguirá creciendo en complejidad. Y como indica Eick et al. [4], la calidad de los cambios en sistemas con poca calidad tiende también a ser pobre.

Otros autores reconocidos en el campo de la Ingeniería de Software realizaron sus propias definiciones basadas en la metáfora original. Martin Fowler asocia la deuda técnica con la deuda financiera y agrega formas de pago e intereses generados [5]. Chris Sterling suma el concepto de degradación de los componentes [6].

En este sentido, existen diferentes aproximaciones útiles para la identificación del código fuente con deuda técnica: violaciones en la modularidad [7], problemas en los patrones de diseño [8], [9], code smells [5], [9], [10], [11], [12], [13], [14], defectos encontrados con herramientas de análisis estático de código fuente [15], [16], [17], [18].

Debido a la incidencia de la deuda técnica resulta imprescindible gestionarla adecuadamente. Para ello es necesario contar con definiciones exactas así como medidas cuantitativas de los componentes que la conforman. Esta información contribuirá a una mayor visibilidad sobre la deuda técnica, así como también, permitirá elaborar y difundir métodos y herramientas que permitan gestionarla adecuadamente.

Un modo de dirigir este esfuerzo es mediante el uso de predictores con el objeto de anticipar las secciones o módulos más propensos a defectos en el futuro, a fin de enfocar los casos de prueba en los mismos.

Existen diferentes enfoques en cuanto a los predictores, pudiendo utilizarse herramientas de análisis estático a partir de patrones, métricas de cambio o métricas de código [19].

Las herramientas de análisis estático a partir de patrones para encontrar a bajo nivel errores de programación son cada vez más comunes. Entre ellas se encuentran FindBugs y PMD, ambas open-source. Lo que aportan esta clase de predictores son avisos en cuanto al *code smell*, haciendo una apreciación de acuerdo a la gravedad de los mismos [20] [21].

Por su parte, las métricas de cambio consideran que un complejo proceso de cambio de código afecta negativamente al producto software. En [22] se exhibe un estudio donde se fundamenta que las clases donde se realizaron más cambios son las más propensas a presentar errores en el futuro. A una mayor cantidad de cambios en el producto, mayor será la posibilidad de que presente fallos. En este sentido, un ejemplo lo representa la herramienta Bug Maps-Granger [23], que aplica la Prueba de Causalidad de Granger al histórico del producto software (dónde se cambió el código, cuándo, cuantas veces a lo largo del tiempo). Estos predictores están más relacionados con el histórico del producto y a partir de él otorga las recomendaciones.

Las métricas de código fuente permiten establecer relaciones en cuanto a, por ejemplo, la complejidad y lo propensa a fallas que puede ser una clase. En base a la bibliografía, se puede señalar que lo más complicado en este proceso es la definición del conjunto de métricas que permitan decir algo con respecto a la complejidad del software y la probabilidad de encontrar errores, tomando diferentes enfoques (por ejemplo, minería de datos). En [24] se

señala que las métricas más usadas son las CK (Chidamber y Kemerer) [25] y las técnicas usadas para conseguir predictores son las de Aprendizaje Automático (*Machine Learning*), dentro del campo de la Inteligencia Artificial.

La vinculación entre los valores de las métricas y las fallas puede ser útil en diferentes etapas del ciclo de vida del software. En una primera instancia, puede dar indicios sobre en qué clases o porciones de código debe enfocarse la prueba de software [26].

En un objetivo más a largo plazo, utilizando los incidentes históricos y vinculándolos con los valores de las métricas, puede permitir el ajuste de los umbrales de las mismas.

Líneas de investigación y desarrollo

Desde su creación en 1992, la metáfora de la "deuda técnica" ha venido desarrollándose en el ámbito científico. Por lo tanto, es necesario realizar una revisión sistemática [27], para conocer la evolución del concepto.

De manera similar, es necesario el estudio de las herramientas libres que miden la deuda técnica y su relación con conceptos similares: code smell, anti patrones y modelos de medición de mantenibilidad [28].

Teniendo estos dos puntos en cuenta, se propone elaborar un conjunto de buenas prácticas para prevenir, medir y gestionar la deuda técnica en el desarrollo de software.

Por otra parte, se propone el diseño y desarrollo de una herramienta que permita registrar y relacionar, en un proyecto de desarrollo de software, los reportes de errores con métricas aplicadas al código fuente. Los resultados ofrecidos por la herramienta (las relaciones), podrán ser utilizados en la etapa de prueba, durante el desarrollo del software.

Adicionalmente, permitirá trabajar con distintas versiones del producto, en su evolución, tomando valores de métricas en un momento inicial, y repitiendo sucesivamente el proceso de medición, de manera que se puedan establecer umbrales ajustados al proyecto.

Esta metodología permitirá establecer un patrón en cuanto a umbrales más adecuados en cada métrica, de modo que, en futuros desarrollos, sólo sea necesario medir el código fuente para anticipar posibles clases o paquetes que contengan errores o defectos, donde deberían enfocarse los esfuerzos en la etapa de prueba.

Resultados y Objetivos

Algunos resultados en esta línea son:

En primera instancia, se realizó una Revisión Sistemática de la Literatura, para obtener información actualizada y relevante sobre experiencias de gestión de proyectos de software que incorporen herramientas específicas que contribuyan a la gestión cuantitativa de los mismos, cuyos resultados se publicaron en [29].

Se realizó una aplicación para evaluar la mantenibilidad de un producto software. La herramienta automatiza un método de medición, basado en GQM, calcula métricas, almacena las mediciones y compara los atributos evaluados. Se aplicó a la evaluación del software un gestor de contenidos Joomla. Los resultados permitieron observar el comportamiento de los indicadores que surgen de las métricas OO que se vinculan a determinados atributos de calidad en un producto software específico que se encuentra en estado de explotación y susceptible de mantenimiento evolutivo [30].

Formación de Recursos Humanos

En el Grupo de Investigación sobre Calidad de Software (GICS) en esta línea de trabajo están involucrados 3 docentes investigadores, 1 becario de investigación de pregrado y un alumno de grado. Dos alumnos desarrollaron su Trabajo Final de Aplicación (TFA) en esta línea para obtener el título de Licenciados en Sistemas de Información de la UNNE en el último año.

Referencias

[1] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2011, p. 75.

[2] W. Cunningham, "The wycash portfolio management system," in OOPSLA '92: Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum), 1992, pp. 29–30.

[3] M. M. Lehman, J. F. Ramil y D. E. Perry, «Metrics and Laws of Software Evolution - The nineties View,» de *Proc. 4th International Software Metrics Symposium (METRICS '97)*, Albuquerque, NM, 1997.

[4] Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., & Mockus, A. (2001). Does code decay? assessing the evidence from change management data. Software Engineering, IEEE Transactions on, 27(1), 1-12.

[5] [M. Fowler, "Technical debt," 2009. [Online]. Available: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[6] C. Sterling, Managing Software Debt, 1st ed. Westford,Massachusetts: Addison Wesley, 2010.

[7] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in Proc. 33th International Conference on Software Engineering, May 2011, pp. 411–420.

[08] C. Izurieta and J.M. Bieman, "Testing consequences of grime buildup in object oriented design patterns,", 1st International Conference on Software Testing, ICST '08, april 2008, pp. 171 –179.

[09] W. J. Brown, R. C. Malveau, and T. J. Mowbray, "AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis." Wiley, Mar. 1998.

[10] M. Lanza and R. Marinescu, Object-oriented Metrics in Practice. Berlin: Springer-Verlag, 2006.

[11] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," IEEE International Conference on Software Maintenance, vol. 0, pp. 350–359, 2004.

[12] J. Schumacher, N. Zazworka, F. Shull, C. Seaman, and M. Shaw, "Building empirical support for automated code smell detection," in Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 8:1–8:10.

[13] N. Zazworka, M. A. Shaw, F. Shull, and C. Seaman, "Investigating the impact of design debt on software quality," in Proceeding of the 2nd working on Managing technical debt, ser. MTD '11. New York, NY, USA: ACM, 2011, pp. 17–23.

[14] S. M. Olbrich, D. S. Cruzes, and D. I. K. Sjoberg, "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in Proceedings of the 2010 IEEE International Conference on Software Maintenance, ser. ICSM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[15] A. Vetro', M. Morisio, and M. Torchiano, "An empirical validation of findbugs issues related to defects," in Evaluation and Assessment in Software Engineering (EASE), EASE 2011, April 2011.

[16] D. Hovemeyer and W. Pugh, "Finding bugs is easy," SIGPLAN Not., vol. 39, pp. 92–106, December 2004.

[17] N. Ayewah and W. Pugh, "The google findbugs fixit," in Proceedings of the 19th international symposium on Software testing and analysis, ser. ISSTA '10. New York, NY, USA: ACM, 2010, pp. 241–252.

[18] C. Boogerd and L. Moonen, "Evaluating the relation between coding standard violations and faults within and across software versions," in Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on, may 2009, pp. 41 –50.

[19] C. Couto, P. Pires, M. T. Valente, R. Bigonha, A. Hora y N. Anquetil , «BugMaps-Granger: A Tool for Causality Analysis between Source Code Metrics and Bugs,» de *Brazilian Conference on Software: Theory and Practice*, Brasilia, Brazil, 2013.

[20] J. E. M. Araújo, S. Souza y M. T. Valente, «Study on the relevance f the warnings reported by Java bug-findings tools,» *IET Soft,* vol. 5, nº 4, pp. 366-374, 2011.

[21] F. Arcelli Fontana, P. Braione y M. Zanoni, «Automatic detection of bad smells in,» *In Journal of Object Technology,* vol. 11, nº 2, pp. 1-38, 2012.

[22] M. Steff y B. Russo, «Characterizing the roles of classes and their fault-proneness through change metrics,» de *ESEM '12 Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, Lund, Suecia, 2012.

[23] C. Couto, P. Pires, M. T. Valente, R. Bigonha, A. Hora y N. Anquetil , «BugMaps-Granger: A Tool for Causality Analysis between Source Code Metrics and Bugs,» de *Brazilian Conference on Software: Theory and Practice*, Brasilia, Brazil, 2013

[24] R. Malhotra y A. Jain, «Software Fault prediction for Object Oriented Systems: A Literature Review,» *ACM SIGSOFT Software Engineering Notes,* vol. 36, nº 5, pp. 1-6 , 2011 .

[25] S. Chidamber y C. Kemerer, «A metrics suite for object oriented design,» *Software Engineering, IEEE Transactions,* vol. 20 , nº 6, pp. 476 - 493, 1994.

[26] P. Bourque y R. E. Fairley, SWEBOK, Tercera ed., IEEE, 2014, pp. 82-87.

[27] Kitchenham, B. (2004). Procedures for performing systematic reviews. Keele, UK, Keele University, 33(2004), 1-26.

[28] Izurieta, C., Vetrò, A., Zazworka, N., Cai, Y., Seaman, C., & Shull, F. (2012, June). Organizing the technical debt landscape. In Proceedings of the Third International Workshop on Managing Technical Debt (pp. 23-26). IEEE Press.

[29] Dapozo, G. N.; Greiner, C. L.; Chiapello, J. "Revisión sistemática de herramientas de apoyo a la gestión cuantitativa de proyectos de software" ". III Jornadas de Investigación en Ingeniería del NEA y países limítrofes: Nuevos escenarios para la ingeniería en el Norte Grande. UTN - Facultad Regional Resistencia. ISBN: 978-950-42-0157-1. 9 y 10 de Junio de 2014. Resistencia, Chaco, Argentina.

[30] Julio Acosta, Cristina Greiner, Gladys Dapozo. "Herramienta para evaluar atributos de mantenibilidad en aplicaciones PHP". 43 JAIIO – 43 Jornadas Argentinas de Informática. 15º Simposio Argentino de Ingeniería de Software. ISSN: 1850-2792. Pp 80 a 90. Universidad de Palermo. Buenos Aires. 1 al 5 de septiembre de 2014.