Un programa de investigación en deuda técnica de software

Matalonga, Santiago
Solari, Martín

Documento de Investigación No.11
Centro de Investigación e Innovación en Ingeniería de Software
Facultad de Ingeniería
Universidad ORT Uruguay
26 de agosto de 2013
ISSN 1688-6372

Documento de Investigación

ISSN 1688-6372

Un programa de investigación en deuda técnica de software

Santiago Matalonga (Facultad de Ingeniería, Universidad ORT Uruguay)
Martín Solari (Facultad de Ingeniería, Universidad ORT Uruguay)

Documento de Investigación No.11

Facultad de Ingeniería
Universidad ORT Uruguay

26 de agosto de 2013

Un Programa de Investigación en Deuda Técnica de Software

Santiago Matalonga y Martín Solari
Centro de Investigación e Innovación en Ingeniería de Software
Universidad ORT Uruguay
Montevideo - Uruguay
Email: smatalonga@uni.ort.edu.uy, martin.solari@ort.edu.uy

ABSTRACT

La ingeniería de software es la disciplina del conocimiento que se ocupa del problema de producir software. El software es cada día más pervasivo en nuestra vida. Sin embargo, la ingeniería de software es una disciplina reciente y todavía basada en modas y referentes. Las teorías comprensivas y la investigación rigurosa con respecto a la producción de software son escasas. La ingeniería de software empírica tiene por objetivo producir conocimiento confiable y aplicable a la producción de software. Esta área de investigación aplica el método científico experimental en la investigación en ingeniería de software. En este taller se presentan mecanismos de investigación utilizados por la comunidad de investigación en Ingeniería de software empírica. Entre los métodos presentado se discuten métodos primarios de investigación (como experimentos y casos de estudio) y métodos secundarios (revisiones sistemáticas de la literatura). Para ejemplificar la aplicación de esta visión de la investigación, este taller presenta como se han aplicado estos conceptos para conducir un programa de investigación en deuda técnica.

PALABRAS CLAVES:
Ingeniería de Software Empírica; Experimentación; Deuda Técnica

1. INTRODUCCIÓN

La ingeniería de software esta cada vez más presente en nuestra vida diaria [1]. El software no sólo se encuentra presente dentro de nuestra computadora, sino que se está volviendo cada vez más presente en la gestión de las empresas, teléfonos móviles, semáforos en la calle, marcapasos o los medios audiovisuales.

Sin embargo, la evolución de los métodos de producción de software se rige más por modas que por teorías desarrolladas y sustentadas. La ingeniería de software empírica busca la aplicación del método científico para entender cómo mejorar los productos y los métodos de producción de software.

La ingeniería de software empírica es el área de investigación que promueve la utilización del método científico experimental para la generación de conocimiento sobre el proceso de desarrollo de software. Aunque la práctica del método experimental es algo habitual en otras disciplinas científicas, su aplicación a la investigación en ingeniería de software es reciente. Los orígenes de esta área se remontan a varias propuestas en los años 70 donde se aboga por la observación empírica de los métodos de desarrollo de software. Sin embargo, los primeros experimentos y la formalización del método experimental aplicado en ingeniería de software fue realizada por Basili en los años 80 [2].

Este taller presenta un resumen de los objetivos de la Ingeniería de Software empírica, detallando los métodos de investigación más usados en la actualidad. Y luego presentamos su aplicación práctica para la planificación en un programa de investigación en Deuda técnica.

Deuda técnica es uno de los temas de investigación que ha estado recibiendo cada vez más atención. El nombre representa una metáfora para explicar los fenómenos que ocurren cuando se prioriza una dimensión de gestión de la ingeniería de software (por ejemplo, cronograma) por sobre otras (típicamente siendo la calidad). Como metáfora, la deuda técnica es muy buena y educativa para explicar este fenómeno. Pero al igual que muchas otras herramientas utilizadas en la Ingeniería de Software, no tiene sustento teórico firme. Nuestro programa de investigación pretende aplicar el método científico para observar si la metáfora tiene una validez suficiente como para construir teorías que puedan ser útiles para su aplicación en la industria.

2. INGENIERÍA DE SOFTWARE EMPÍRICA

La Ingeniería de Software Empírica (ESE, por su sigla en inglés) se basa en la aplicación del método científico experimental a la investigación en ingeniería de software [3]. Actualmente, una proporción considerable del conocimiento difundido en el área se basa en propuestas de autores, deducciones teóricas o pruebas de concepto [4]. En estos casos, las afirmaciones no cuentan con una evidencia rigurosa ni una base empírica formalizada. En cambio, en otras ciencias y ramas de la ingeniería, la experimentación y la observación sistemática son métodos habituales para fundamentar el conocimiento generado. El enfoque empírico permite obtener evidencia para probar nuestras hipótesis sobre técnicas y herramientas para construir software. De esa forma, podemos obtener conocimiento fiable para fundamentar las afirmaciones en las distintas áreas del proceso de software.

La ingeniería de software es una disciplina tecnológica de alta complejidad, donde el componente humano tiene una incidencia decisiva. Por este motivo, sería imposible estudiar el fenómeno de la producción de software desde un enfoque puramente teórico o formal. Actualmente la Ingeniería de Software Empírica es un método aceptado por la

comunidad investigadora que intenta combinar la teoría y la práctica en un enfoque que produzca conocimiento aplicable para el practicante [5].

El método científico experimental realiza observación rigurosa y manipulación de la realidad para obtener conocimiento. Esta observación y manipulación puede darse en distintos grados y distintos métodos de investigación. Los métodos de investigación empíricos van desde las encuestas y estudios de caso, hasta los estudios correlacionales y experimentos controlados [6]. En cada método de investigación existen distintos tipos de intervención en el fenómeno bajo estudio.

2.1. Métodos de Investigación

El experimento controlado es el método donde existe mayor grado de control, lo que normalmente se conoce como condiciones de laboratorio. El experimento controlado permite estudiar relaciones causa-efecto, aislando variables con influencia significativa en un fenómeno determinado. Los experimentos controlados utilizan la aleatoriedad, la fijación o balanceo de variables, los grupos de control, como forma de asegurar un estudio riguroso del fenómeno. Cuando alguna de estas condiciones de control está limitada, se utilizan los cuasi-experimentos que son un tipo de estudios correlacionales. Si bien el experimento controlado es el método empírico que produce conocimiento más abstracto, utiliza un enfoque reduccionista de la realidad y debe apoyarse en otros métodos empíricos.

Los estudios de caso son un enfoque holístico que permite estudiar un fenómeno en su contexto real [10]. Las encuestas, trabajos de campo, la investigación-acción, son ejemplos concretos de este método de investigación. Los estudios de caso son necesarios antes y después de los experimentos controlados. Permiten refinar las hipótesis a probar en experimentos controlados y posteriormente validar el conocimiento abstracto en condiciones más complejas, cercanas a la realidad. Los distintos métodos empíricos son complementarios y permiten profundizar en el estudio de un fenómeno paso a paso.

La realización de experimentos es una tarea compleja que requiere la aplicación rigurosa de un conjunto de métodos de investigación. Entre estos métodos se incluyen las técnicas para el diseño de experimentos, los métodos de análisis estadístico, además de conocimientos propios de la disciplina donde se inscriben las técnicas. Uno de los componentes de esta línea de investigación es el estudio de los instrumentos para la replicación de experimentos, como la comunicación entre investigadores y los paquetes de laboratorio [11]. Nuestro grupo de investigación viene trabajando hace varios años en la replicación de un experimento controlado sobre técnicas de prueba de software y el estudio metodológico de la replicación de experimentos [12].

Los experimentos controlados y otros métodos de investigación empírica no son un fin en sí mismos. Todo el conocimiento que se produce aplicando estos métodos es limitado por

naturaleza. Depende de las condiciones propias de contexto dónde se realiza el estudio. Por este motivo, es necesario que la comunidad de Ingeniería de Software Experimental pueda agregar resultados de estudios con variaciones en el contexto y las variables analizadas sobre un mismo fenómeno [13]. Es así como surgen los estudios secundarios: estudios que no realizan una observación ni manipulación directa de la realidad, pero sintetizan información de otros estudios primarios.

2.2. Estudios empíricos secundarios

Ejemplos de estudios secundarios son los mapeos y revisiones sistemáticas. Estos estudios permiten agregar resultados de investigación empírica y presentar el conocimiento de forma sintética para el practicante.

Los mapeos sistemáticos tienen como objetivo establecer un mapa conceptual del conocimiento en un área [14]. Se enfocan en preguntas de investigación más generales y no realizan síntesis de resultados.

Las revisiones sistemáticas son un método de búsqueda y síntesis rigurosa de la mejor evidencia disponible en la literatura científica sobre un fenómeno concreto. La revisión sistemática se basa en un protocolo que guía todo el proceso, a la vez que lo hace repetible y auditable por otros investigadores [15]. Nuestro grupo de investigación ha realizado varias experiencias con revisiones sistemáticas, por ejemplo sobre factores que afectan los procesos de software ágiles distribuidos [16].

3. INTRODUCCIÓN AL CONCEPTO DE DEUDA TÉCNICA

Deuda Técnica es una metáfora que se presenta útil para explicar fenómenos que ocurren en la producción de software. La deuda técnica se define como la metáfora que explica las consecuencias de priorizar una dimensión de la gestión de proyectos por sobre otra [9] . En general la deuda técnica se explica mejor cuando las prácticas de control de calidad son dejadas de lado frente a presiones de costos o cronograma.

La primera referencia al concepto fue introducida en 1992 por Ward Cunningham [17]. Él utilizo el concepto para referirse a la inmadurez del código frente soluciones rápidas en contraste con la aplicación correcta de patrones y principios de código limpio. Para la comunidad, la metáfora introducida por Cunningham se transformó en una forma de explicar el aumento de los costos a lo largo de la vida de los sistemas. Actualmente la metáfora ha ampliado su ámbito de aplicación y su uso abarca todo el ciclo de vida de la producción de software [18].

La metáfora se ha vuelto relevante para la comunidad de Ingeniería de Software por su facilidad para explicar los costos ocultos de la baja calidad. Por ejemplo el grupo Garner

estima que la deuda técnica mundial en el años 2010 fue cercana a los 500 millones de dólares y que puede duplicarse en cinco años [19]. Bill Curtis estima que la deuda técnica es un problema que es agnóstico a la metodología, pudiendo medir niveles de deuda técnica en proyectos de desarrollos tradicionales y ágiles [20].

La mayoría de los autores refiere a costos de mantenimiento de los proyectos de software [21], [22]. En especial a la aplicación de la metáfora para la toma de decisiones en situaciones donde las opciones actuales están limitadas por decisiones pasadas.

Los temas de investigación actuales rondan la aplicación práctica de la metáfora y al generación empírica de conocimiento que pueda ser aplicable a proyectos de software. Se destaca en esta última línea el trabajo de Seaman [23][24], que persigue el objetivo desarrollar una teoría con respecto a la utilización de la deuda técnica para formalizar su aplicación y el alcance del los términos económicos incluidos en ella.

4. UN PROGRAMA EMPÍRTICO DE INVESTIGACIÓN EN DEUDA TÉCNICA

El centro de innovación e Investigación en Ingeniería de Software de la Universidad ORT Uruguay lleva adelante un programa de investigación basado en la aplicación del método científico con las herramientas de la Ingeniería de Software Empírica. Nuestra visión consiste en acelerar los tiempos de transferencia entre el estado del arte y estado de la practica.

Para el tema de deuda técnica consideramos un enfoque holístico que intercambia métodos secundarios, casos de estudio y experimentos controlados.

4.1. Mapeos sistemáticos como medio para identificar preguntas de investigación

Los mapeos sistemáticos son útiles para el trabajo con temas abiertos de investigación, para identificar las tendencias de investigación o para identificar posibles preguntas de investigación [15].

Típicamente en el CIIIS abordamos las nuevas líneas de investigación mediante la realización de Mapeos sistemáticos. En el caso del programa de investigación de deuda técnica, el mapeo sistemático nos permitió determinar el estado actual de la investigación [9]. A partir de este trabajo, hemos podido concluir que:

- No existe una definición consensuada del concepto de deuda técnica.

- Los trabajos empíricos de deuda técnica son relativamente pocos, y parecen estar dominados por los reportes de casos de estudio.
- Los principales investigadores son Seaman, Knutchen y Glee.

4.2. Revisiones sistemáticas como medio para la recolección de evidencia para la construcción de teorías

Las revisiones sistemáticas presentan un protocolo de investigación más formal que los mapeos. En ellas se busca contestar con evidencia a preguntas de investigación concretas. En el CIIIS hemos aplicado revisiones sistemáticas de la literatura para realizar acciones de transferencia tecnología a empresas productoras de software [16].

Para el caso del programa de investigación en deuda técnica nos encontramos aplicando un protocolo de revisión sistemática con el objetivo de identificar evidencia empírica de formulas o métodos de cálculo de deuda técnica. El razonamiento es que si no se cuenta con una definición consensuada, es improbable que se cuente con un método claro de medición y por tanto, la comparación de resultados de distintos autores está sujeta a grandes variaciones. Este tipo de conjeturas puede contestarse con revisiones sistemáticas.

4.3. Casos de estudio para la validación de teorías

Mediante los casos de estudio podemos recolectar información ampliada y de primera mano sobre un fenómeno. A diferencia de los anteriores, los casos de estudio se consideran métodos primarios de investigación por que generan evidencia de primera mano sobre un fenómeno.

Para el programa de deuda técnica tenemos planificado la conducción de un caso de estudio en una empresa de software uruguaya. La misma está llevando adelante procesos ágiles de desarrollo y nuestra intención es observar si la incorporación del concepto de deuda técnica facilita reflexión del equipo de desarrollo con respecto al proceso de producción de software.

4.4. Experimentos controlados para la generación de evidencia

La realización de experimentos controlados es sumamente útil pro que permiten asilar una variable para observar su efecto en el fenómeno bajo estudio. Como contraparte los casos de estudio son difíciles de instrumentar, y como se argumento en la sección 2, en Ingeniería de Software los resultados no pueden faciliten ser desligados de los sujetos involucrados.

En el CIIIS hemos tenido un éxito relativo en la realización de casos de estudio utilizando a estudiantes como sujetos [12].

Para el programa de investigación en deuda técnica queremos reproducir este éxito. Actualmente tenemos dos experimentos en ejecución con estudiantes de grado. En el primero queremos medir el grado de deuda técnica que incurren obligatorios de programación que usan diferentes técnicas (Test Driven Developemnt [25], Code First, y Personal Software Procces [26]). De este experimento esperamos identificar si alguna de estas técnicas tiene efecto en la generación de deuda.

El otro experimento se basa en la identificación de Code Smells[27] en obligatorios de programación de ingeniería de software para verificar su correlación con las mediciones de deuda técnica. De este experimento esperamos identificar si existe una correlación entre la identificación de smells (heurísticas de buenas prácticas de código) con las mecanismos de medición de deuda técnica.

5. CONCLUSIONES

Este taller se ha enfocado en la relevancia de la Ingeniería de software Empírica como herramienta para la producción de conocimiento en esta área. La ingeniería de software es una disciplina dominada por expertos y con pocas teorías formales. Las mayorías de las tendencias son transmitidas en conceptos de buenas prácticas o cuerpos de conocimientos. La Ingeniería de software Empírica tiene por objetivo la generación de conocimiento teórico fundamental mediante la aplicación del método científico.

Este taller presento algunas de las herramientas que están siendo utilizadas por la comunidad. Se presentaron algunos de los resultados obtenidos y los trabajos relevantes del campo. Además se prestó especial atención a la aplicación de estas herramientas en un programa de investigación en Deuda Técnica. Un tema actual de investigación en Ingeniería de software.

REFERENCIAS

[1] E. M. Gray and W. L. Smith, “On the limitations of software process assessment and the recognition of a required re-orientation for global process improvement,” *Software Quality Journal*, vol. 7, no. 1, pp. 21–34, 1998.

[2] V. R. Basili, R. W. Selby, and D. H. Hutchens, “Experimentation in Software Engineering,” *IEEE Transactions on Software Engineering*, vol. 12, no. 7, pp. 733–743, 1986.

[3] N. Juristo and A. M. Moreno, *Basics of Software Engineerig Experimentation*. Kluwer, 2001.

[4] M. V. Zelkowitz, “An update to experimental models for validating computer technology,” *Journal of Systems and Software*, vol. 82, no. 3, pp. 373–376, Mar. 2009.

[5] B. W. Boehm, H. D. Rombach, and M. V. Zelkowitz, *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili*. Springer, 2005, p. 431.

[6] B. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, “Preliminary guidelines for empirical research in software engineering,” *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, 2002.

[7] G. Quintana and M. Solari, “Mapeo Sistemático sobre Experimentos de Generación Automática de Casos de Prueba Estructurales,” in *Conferencia Latinoamericana de Informática (CLEI 2012)*, 2012.

[8] D. Macchi and M. Solari, “Mapeo Sistemático de la Literatura sobre la Adopción de Inspecciones de Software,” in *Conferencia Latinoamericana de Informática (CLEI 2012)*, 2012.

[9] A. Villar and S. Matalonga, “Definiciones y tendencia de deuda técnica: Un mapeo sistemático de la literatura,” in *Memorias de la XVI Conferencia Iberoamericana de Ingeniería de Software CIbSE 2013*, 2013, pp. 33–46.

[10] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2008.

[11] N. Juristo, S. Vegas, M. Solari, S. Abrahao, and I. Ramos, “A Process for Managing Interaction between Experimenters to Get Useful Similar Replications,” *Information and Software Technology*, 2012.

[12] N. Juristo, S. Vegas, M. Solari, S. Abrahao, and I. Ramos, “Comparing the Effectiveness of Equivalence Partitioning, Branch Testing and Code Reading by Stepwise Abstraction Applied by Subjects,” in *International Conference on Software Testing, Verification and Validation (ICST 2012)*, 2012.

[13] N. Juristo and S. Vegas, “The role of non-exact replications in software engineering experiments,” *Empirical Software Engineering*, vol. 16, no. 3, pp. 295–324, Aug. 2010.

[14] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, “Using Mapping Studies in Software Engineering,” vol. 2, 2007.

[15] B. A. Kitchenham, “Guidelines for performing Systematic Literature Reviews in Software Engineering.” Evidence based - Software Engineering Group. School of Computer Science and Mathematics. Keele University and Department of Computer Science. University of Durham, 2007.

[16] S. Matalonga, M. Solari, and G. Matturro, “Factors Affecting Distributed Agile Projects: A Systematic literature Review,” *International Journal of Software Engineering and Knowledge Engineering*, vol. In Press, 2013.

[17] W. Cunningham, “The WyCash portfolio management system,” in *Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum)*, 1992, pp. 29–30.

[18] P. Kruchten, R. L. Nord, I. Ozkaya, and J. Visser, “Technical debt in software development: from metaphor to theory report on the third international workshop on managing technical debt,” *SIGSOFT Softw. Eng. Notes*, vol. 37, no. 5, pp. 36–38, 2012.

[19] E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt,” *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, Jun. 2013.

[20] B. Curtis, J. Sappidi, and J. Subramanyam, “Measuring the Structural Quality of Business Applications,” in *2011 AGILE Conference*, 2011, pp. 147–150.

[21] E. Lim and A. Informatics, “A Balancing Act : What Software Practitioners Have to Say about Technical Debt,” pp. 22–27, 2012.

[22] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, “An enterprise perspective on technical debt,” in *Proceeding of the 2nd working on Managing technical debt - MTD ’11*, 2011, p. 35.

[23] C. Seaman and Y. Guo, “Measuring and monitoring technical debt,” in in *Advances in Computer Volume 82*, Elsevier, Ed. 2011, pp. 25–46.

[24] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. Da Silva, A. L. M. Santos, and C. Siebra, “Tracking technical debt — An exploratory case study,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 528–531.

[25] S. Fraser, K. Beck, B. Caputo, T. Mackinnon, J. Newkirk, and C. Poole, “Test Driven Development,” *Test*, vol. 06, pp. 459–462, 2003.

[26] W. S. Humphrey, *A Discipline for Software Engineering*, vol. 640. 1995, p. 816.

[27] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, vol. 1. 2008, p. 464.