Técnicas, Estrategias y Herramientas de Comprensión de Programas para facilitar el entendimiento de Sistemas Multiparadigmas

Enrique Miranda[(1)], Corina Abdelahad[(1)], Mario Berón[(1)], Daniel Riesco[(1)],

[(1)]Departamento de Informática / Facultad Ciencias Físico Matemáticas y Naturales/
Universidad Nacional de San Luis
Ejército de los Andes 950 – San Luis – Argentina
{eamiranda, cabdelah, mberon,driesco}@unsl.edu.ar

Resumen

Dentro del contexto del ciclo de vida del producto de software, una de las tareas que más tiempo y dedicación consume es la de Mantenimiento y Evolución de Software (MES). A partir de la necesidad de asistir al arduo proceso de comprensión requerido en la etapa mencionada anteriormente, surge una disciplina de la Ingeniería de Software denominada Comprensión de Programas (CP). La CP se presenta como un área de investigación interesante para impulsar el trabajo de MES a través de técnicas y herramientas que asistan al ingeniero de software en la difícil tarea de comprender sistemas.

Por otra parte, los desafíos recientes en la industria de software requieren cada vez más de lenguajes y frameworks de programación con características multiparadigmas, los cuales a su vez presentan desafíos a las estrategias de comprensión.

En esta línea de investigación se propone estudiar conceptos, definir estrategias e implementar herramientas de Comprensión de Programas para analizar sistemas desarrollados con lenguajes con soporte multiparadigma.

Palabras clave: Ingeniería Reversa, Comprensión de Programas, Lenguajes con soporte Multiparadigma, Sistemas de Software de gran Envergadura.

Contexto

La presente línea de investigación se enmarca en el Proyecto de Investigación "Ingeniería de Software: Conceptos, Prácticas y Herramientas para el Desarrollo de Software con Calidad" de la Facultad de Ciencias Físico Matemáticas y Naturales de la Universidad Nacional de San Luis. Proyecto No P-031516 que es la continuación de diferentes proyectos de investigación, a través de los cuáles se ha logrado vínculos con distintas universidades a nivel nacional e internacional. Además, se encuentra reconocido por el Programa de Incentivos.

Introducción

Siempre que se realice un cambio a una pieza de software, es importante que el ingeniero de software obtenga un completo entendimiento de la estructura, comportamiento y funcionamiento de la parte del sistema que se está modificando. Es sobre la base de este entendimiento, que se pueden generar las propuestas de modificación para lograr los objetivos en la etapa de mantenimiento.

Los ingenieros de software encargados del mantenimiento, empeñan mucho de su tiempo leyendo el código y la documentación complementaria para comprender su lógica, propósito y estructura. En este contexto, el entendimiento de un sistema no es tarea fácil, ya que generalmente, la persona que realiza el mantenimiento no es la misma que escribió el código o bien ha pasado un período de tiempo considerable desde que esta persona finalizó las tareas de desarrollo. En este sentido, Eagleson refleja claramente este aspecto: *"Cualquier pieza de código que no se haya visto desde hace seis meses o más, bien podría haber sido escrita por alguien más"*.

La ley expone una visión de lo difícil que puede resultar mantener un sistema de software, al punto tal que la misma persona que ha desarrollado un sistema particular puede resultarle desconocido luego de un determinado tiempo. A todo lo anterior se le debe sumar que los sistemas en su gran mayoría no cuentan con documentación que cumpla ciertos requisitos básicos como por ejemplo que la misma se encuentre actualizada, que sea consistente, que esté completa, entre otros.

Es evidente que todos los aspectos mencionados anteriormente han sido los motivos que han impulsado el surgimiento de distintos recursos para asistir al ingeniero de software en la difícil tarea de comprender un sistema. En este contexto, surge una disciplina denominada Comprensión de Programas (CP), la cual está dirigida a proveer modelos, métodos, técnicas y herramientas, basada en un proceso de aprendizaje específico y procesos de ingeniería, a fin de encontrar un conocimiento más profundo acerca de un sistema de software [1].

A través de un extenso estudio y análisis de herramientas de comprensión [1,2,3], se pudo comprobar que para lograr una comprensión efectiva del sistema bajo estudio, el sujeto cognoscente debe interrelacionar el Dominio del Problema con el Dominio del Programa. El primer dominio, hace referencia a la salida del sistema. El segundo, a las componentes de software usadas para producir dicha salida. Claramente, existe una relación (o relaciones) concreta y robusta entre ambos dominios. Durante la implementación del sistema el desarrollador va construyendo (desde su perspectiva) la relación entre los dominios y de alguna manera la misma se encuentra activa durante el ciclo de desarrollo. Sin embargo, en la etapa de mantenimiento y evolución del software es muy posible que dicha relación pierda fuerza y en determinados casos llegue a desaparecer. La reconstrucción de este tipo de relación es compleja y requiere de estrategias de comprensión efectivas que sean elaboradas tomando como base diferentes temáticas como lo son las Técnicas de Extracción de la Información, Administración de la Información, Estrategias de Vinculación de Dominios, Visualización de Software, entre otras [1,2,3,4].

Actualmente, existen numerosas herramientas de CP con sofisticadas técnicas de exploración de código. Muchas de estas funcionan de manera adecuada, sin embargo, ciertas tareas de comprensión son todavía muy complejas. Ciertas herramientas proveen diferentes vistas de la información extraída del sistema. No obstante, el uso de este tipo de técnicas no asiste al programador en las complejas tareas de abstracción y vinculación de los Dominios del Problema y del Programa, ya que las vistas provistas brindan solo un aspecto del código fuente [4]. Una forma de salvar este inconveniente es a través de la elaboración de estrategias que interconecten los Dominios del Problema y Programa [1,4,5]. La característica mencionada previamente simplifica la exploración porque el ingeniero de software solamente inspecciona las partes del sistema relacionadas con la funcionalidad de estudio. De esta manera, se alcanza una considerable reducción de esfuerzos humanos, lo que se traduce en reducción de costos del proyecto.

Tomando en cuenta este contexto, a través del estudio del estado del arte de CP es posible inferir los siguientes aspectos:

- El proceso de elaboración de estrategias de vinculación de Dominios es arduo y complejo. El mismo necesita de un estudio profundo de las distintas disciplinas que se mencionaron previamente; es por esto que es difícil encontrar numerosas estrategias de vinculación de Dominios con estas características.
- La mayoría de las técnicas y herramientas de CP abordan exclusivamente el Dominio del Programa. Es decir, realizan análisis de los artefactos relativos al código fuente de un programa sin tener en cuenta componentes del Dominio del Problema. Ciertos trabajos sólo proponen teorías y técnicas novedosas, pero no presentan herramientas, casos de estudios o Dominios de aplicación interesantes en donde se puedan poner en práctica efectivamente las mismas.
- Entre las herramientas y estrategias propuestas es difícil encontrar algunas que

abarquen lenguajes multiparadigma como por ejemplo Python. Este estilo de programación ha tomado cada vez más relevancia ya que es ampliamente utilizado en la actualidad.

Teniendo en cuenta los aspectos destacados en los ítems anteriores, es posible determinar que la mayoría de las técnicas y herramientas existentes de CP no proponen estrategias robustas y eficaces para interrelacionar los Dominios del Problema y del Programa. Además, del limitado conjunto de estrategias disponibles que abordan dicho enfoque, algunas poseen ciertas desventajas que impide su aplicación a sistemas de gran tamaño escritos en lenguajes ampliamente utilizados en el contexto de desarrollo de software a gran escala.

Como objetivo general de esta línea de investigación se pretende mejorar la comprensión de sistemas multiparadigmas, por medio de la definición de estrategias de CP que permitan vincular los Dominios del Problema y del Programa. Como medio para alcanzar dicho objetivo se propone el estudio de distintas temáticas relacionadas a modo de líneas de investigación comprendidas dentro de la temática principal.

Líneas de Investigación y Desarrollo

A continuación, se describe brevemente un conjunto no exhaustivo de líneas de investigación desprendidas de la línea principal.

Extracción de Información

Una de las actividades iniciales de cualquier enfoque que se enmarque dentro del contexto de Comprensión de Programas, es la extracción de la información del sistema bajo estudio. Por esta razón, se utilizan técnicas que permiten obtener distintos tipos de datos relacionados con determinados aspectos de la estructura y/o el comportamiento del programa. Dichas técnicas se conocen en el contexto de CP como Técnicas de Extracción de Información (TEI). Se puede decir que las TEI se subdividen en dos categorías dependiendo del tipo de información que se desee extraer: TEI estática y TEI dinámica. Las primeras permiten recuperar todos los atributos de los objetos definidos en el código fuente del programa [6]. Las segundas posibilitan conocer los componentes del programa utilizados para una ejecución específica del sistema [6,7].

Es difícil determinar cual es la técnica más eficaz, ya que se podrían encontrar numerosos trabajos que presentan resultados interesantes usando alguna de las técnicas. Incluso diversos enfoques combinan ambos tipos de técnicas, logrando así destacar distintos aspectos del sistema mediante la información obtenida con técnicas dinámicas en conjunto con la información estática [1,6].

Esta temática se torna más determinante cuando los sistemas analizados están desarrollados en lenguajes con soporte multiparadigma. Este tipo de lenguajes presentan numerosos desafíos en el contexto de esta temática [5,8].

Reducción de información

En el contexto de análisis de software de gran envergadura, es necesario contar con técnicas de reducción de información para poder generar abstracciones. Por lo general, en dichos sistemas se extrae cierta cantidad de información que no está vinculada con aspectos relativos a la solución del problema para el cual fue desarrollado el sistema. Para que las representaciones de base contengan información precisa referente al problema que el sistema intenta solucionar, es necesario filtrar aquellos artefactos de software que no están relacionados con la lógica subyacente del programa. Para llevar a cabo la abstracción es posible valerse de distintas técnicas y herramientas, como por ejemplo, métricas de software, técnicas de *clustering*, estrategias de identificación de artefactos irrelevantes, entre otras [9,10].

Transformaciones entre Modelos

Un enfoque que permite lidiar con las características y dificultades que presentan los sistemas multiparadigmas es la Arquitectura Dirigida por Modelos (MDA) [11,12]. La idea principal de MDA, es separar la funcionalidad del sistema de su implementación sobre plataformas específicas, considerando la evolución desde modelos abstractos hasta su implementación, acrecentando el grado de automatización y logrando interoperabilidad con

múltiples plataformas, lenguajes formales y lenguajes de programación.
Usando MDA, cualquier especificación puede ser expresada con modelos, en consecuencia, una estrategia de comprensión puede comprender un proceso de refinamiento y transformación entre modelos de manera tal que el nivel de abstracción vaya aumentando hasta llegar a una representación independiente de ciertos aspectos específicos tales como una plataforma, paradigmas de programación, lenguajes de programación, entre otros [11,13].

Modelos del Dominio del Problema
A la hora de interconectar los Dominios del Problema y del Programa es necesario contar con representaciones de cada uno de estos [1,3,4,5]. En el contexto de CP existen diversas estrategias que realizan extracción y representación del Dominio del Programa. Sin embargo, se han encontrado relativamente escasas propuestas de CP que realicen extracción de información del Dominio del Problema para posteriormente construir la representación del mismo. Uno de los principales factores que justifican este hecho es que el Dominio del Problema no cuenta con la cantidad de artefactos que pueden relevarse en el Dominio del Programa. Por lo general, el primero se encuentra plasmado en narrativas escritas en lenguaje natural, especificaciones formales u ontologías. También pueden considerarse la salida del sistema o su comportamiento como un artefacto del Dominio del Problema [1,3,6]. Como puede observarse, los artefactos de este dominio no son fáciles de analizar y por consiguiente no es sencillo extraer información de los mismos. La mayoría de las estrategias de comprensión que consideran el Dominio del Problema, lo hacen teniendo en cuenta representaciones del mismo ya definidas. Por esto motivo se considera relevante ahondar en el estudio de esta temática para lograr obtener otras representaciones de este dominio y así contar con mayores recursos a la hora de definir estrategias efectivas de CP.

Estrategias de Interconexión entre Dominios
Una vez construidas (o definidas) las representaciones de los Dominios del Problema y del Programa, es necesario definir la estrategia de vinculación que permitirá al ingeniero de software relacionar aquellos artefactos del Dominio del Programa que implementan funcionalidades del Dominio del Problema. Este aspecto es fundamental y el mismo presenta numerosos desafíos dentro del contexto en que se plantea esta línea, es decir, sistemas de software con soporte multiparadigma.

Estrategias de Visualización de Información
Uno de los aspectos fundamentales a tener en cuenta en cualquier estrategia efectiva de comprensión es la manera en que se mostrará la información que ha sido procesada.
La Visualización de Software (VS) es una disciplina de la Ingeniería de Software cuyo propósito es visualizar la información relacionada con los sistemas de software con el objetivo de simplificar el análisis y la comprensión de los mismos [1,14,15]. En el contexto de CP, este aspecto se torna trascendental debido a que, es el vehículo por medio del cual el ingeniero de software puede efectivamente reconstruir, de diversas maneras, la relación entre los dominios [1,4,5].

Resultados Obtenidos/Esperados

Algunos de los resultados destacados obtenidos por esta investigación son:

- Se desarrolló e implementó una estrategia de vinculación de Dominios que fue aplicada a sistemas en Python [5].
- Se utilizaron técnicas de transformación entre modelos para desarrollar una estrategia de Comprensión de Programas [13].
- Se establecieron temáticas de investigación que posibilitarán un abordaje más abarcativo del objetivo principal de la línea.

Entre los objetivos planteados a corto y largo plazo se pueden mencionar:

- Extender los tipos de representaciones del Dominio del Problema utilizados en estrategias de CP.
- Ampliar los tipos de plataformas y lenguajes abarcados por las estrategias vigentes.

- Establecer un conjunto de métricas orientadas a determinar el índice de comprensibilidad de sistemas multiparadigmas.
- Ampliar las estrategias de Ingeniería Reversa de sistemas multiparadigmas que toman como base MDA.
- Definir nuevos mecanismos de abstracción para las representaciones intermedias de las estrategias de CP.

Formación de Recursos Humanos

Las investigaciones realizadas, así como los resultados obtenidos en este trabajo contribuyen al desarrollo de tesis de posgrado, ya sea de doctorado o maestrías en Ingeniería de Software y desarrollo de trabajos finales de las carreras Licenciatura en Ciencias de la Computación, Ingeniería en Informática e Ingeniería en Computación de la Universidad Nacional de San Luis, en el marco del proyecto de investigación mencionado.

Bibliografía

[1] Mario Berón. "Program Inspection to interconnect Behavioral and Operational Views for Program Comprehension". Ph.D. Thesis Dissertation at University of Minho. Braga. Portugal, 2010.

[2] Margaret-Anne Storey. "Theories, Methods and Tools in Program Comprehension: Past, Present and Future". Proceedings of the 13th International Workshop on Program Comprehension, p. 181–191, 2005.

[3] Rugaber S. The Use of Domain Knowledge in Program Understanding. Annals of Software Engineering, 9 (1-2), p. 143–192, 2000.

[4] Carvalho, N. R., Almeida, J. J., Henriques, P. R., y Varanda, M. J. From source code identifiers to natural language terms. Journal of Systems and Software, 100, p. 117–128, 2015.

[5] Miranda, E. et al. Using reverse engineering techniques to infer a system use case model. Journal of Software: Evolution and Process, vol. 31, no 2, p. e2121, 2019.

[6] Eisenbarth, T., Koschke, R., y Simon, D. Aiding Program Comprehension by Static and Dynamic Feature Analysis. En Proceedings of the IEEE Intern. Conference on Software Maintenance (ICSM'01) p. 602–611. IEEE Computer Society, 2001.

[7] Cornelissen, B., Zaidman, A., Van Deursen, A., Moonen, L., y Koschke, R. A systematic survey of program comprehension through dynamic analysis. IEEE Transactions on Software Engineering, 35 (5), p. 684–702, 2009.

[8] Wampler, D. et al. Multiparadigm programming in industry: A discussion with Neal Ford and Brian Goetz. IEEE software, vol. 27, no 5, p. 61-64, 2010.

[9] Abdelwahab Hamou-Lhadj and Timothy Lethbridge. Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System. In 14th International Conf. on Program Comprehension, ICPC '06. p. 181–190. IEEE, 2006.

[10] Jones, K. Automatic summarising: The state of the art. Information Processing & Management, vol. 43, no 6, p. 1449-1481, 2007.

[11] Roxana Giandini, Claudia Pons, and Gabriela Pérez. A two-level Formal Semantics for the QVT Language. In Proceeding of Conferencia Iberoamericana de Software Engineering, CIbSE '09, p. 73–86, 2009.

[12] Pereira, C., Martinez, L., y Favre, L. Recovering Use Case Diagrams from Object Oriented Code: an MDA-based Approach. En 8th International Conference on Information technology: New generations (ITNG), p. 737–742, 2011.

[13] Miranda, Enrique Alfredo, et al. Inferring Use-cases from GUI Analysis. IEEE Latin America Transactions, vol. 13, no 12, p. 3942-3952, 2015.

[14] Storey, M.-A., Fracchia, D., y Müller, H. Cognitive Design Elements to Support the Construction of a Mental Model During Software Visualization. En Proceedings of 5th Iternational Workshop Program Comprehension (IWPC'97), p. 17–28, 1997.

[15] Lanza, M., Ducasse, S., Gall, H., y Pinzger, M. Codecrawler- an information visualization tool for program comprehension. En proceeding on 27th

International Conference on Software Engineering, ICSE p. 672–673, 2005.