Información Tecnológica
Vol. 24(5), 23-30 (2013)
doi: 10.4067/S0718-07642013000500004

Propuesta Metodológica para Caracterizar y Seleccionar Métodos de Ingeniería Inversa

Martín E. Monroy[(1)], José L. Arciniegas[(2)] y Julio C. Rodríguez[(1)]
(1) Universidad de Cartagena, Facultad de Ingeniería, Programa de Ingeniería de Sistemas, Piedra de Bolívar, Bolívar – Colombia (e-mail: mmonroyr@unicartagena.edu.co, jrodriguezr@unicartagena.edu.co)
(2) Universidad del Cauca, Departamento de Telemática, Sector Tulcán, Cauca – Colombia
(e-mail: jlarci@unicauca.edu.co)

Recibido Abr. 16, 2013; Aceptado Jun. 03, 2013; Versión final recibida Jun. 19, 2013

Resumen

Se define un instrumento para caracterizar los métodos de ingeniería inversa que fue construido haciendo uso de la técnica de revisión de características establecida en la metodología DESMET. Se obtuvo así, la identificación de catorce características distribuidas en tres aspectos. El instrumento fue aplicado a diez métodos de ingeniería inversa para validar su utilidad. Esto permitió concluir que la metodología facilita la selección del método más adecuado para un caso específico bajo un escenario de aplicación de la ingeniería inversa.

Palabras clave: ingeniería inversa, construcción del software, arquitectura de software, DESMET

A Methodological Approach for Characterization of Reverse Engineering Methods

Abstract

A tool for characterizing reverse engineering methods that was built using the technique of features revision established in the DESMET methodology is defined. This resulted in the identification of fourteen features distributed on three aspects. The instrument was applied to ten reverse engineering methods to validate its usefulness. This allowed concluding that the methodology facilitates the selection of the most appropriate method for a particular case under a scenario of reverse engineering application.

Keywords: reverse engineering, software development, software architecture, DESMET

INTRODUCCIÓN

Actualmente los procesos organizacionales e industriales se soportan en sistemas software (Van Geet et al, 2010), que requieren de un profundo conocimiento de su arquitectura e implementación al momento de realizarles mantenimiento para garantizar su evolución (Platenius et al, 2012; Bennett y Rajlich, 2000). Este conocimiento se encuentra registrado en forma explícita en la documentación o de manera implícita en la mente de los expertos que lo desarrollaron. Desafortunadamente, con frecuencia no se dispone de dicho conocimiento, sobre todo cuando se trata de sistemas heredados que carecen de documentación o se encuentra desactualizada y no se cuenta con el equipo de expertos que lo crearon (Demeyer et al, 2008; Cuenca et al, 2008).

La ingeniería inversa se ha convertido en la principal solución ante esta situación, permitiendo la adquisición de conocimiento para la reconstrucción de documentación a nivel de diseño y arquitectura, además de otros beneficios como el control de la calidad del producto software manteniendo la trazabilidad de los artefactos que lo conforman, la reutilización de código y el control de clonación no autorizada, el análisis y entendimiento del código malicioso, la coexistencia sincronizada del sistema y los modelos que lo representan, y el apoyo al aprendizaje de la ingeniería de software (Canfora et al, 2011). Cada uno de estos beneficios corresponde a un escenario de aplicación de la ingeniería inversa, convirtiéndola en uno de los mayores retos actuales de la ingeniería de software (Favre, 2012).

Además, la adopción de la ingeniería inversa en las prácticas de construcción de software, cada vez más, exige un mayor conocimiento del tema por parte del equipo de desarrollo, que garantice la comprensión de los métodos y facilite el uso de las herramientas de ingeniería inversa (Canfora et al, 2011). En este orden de ideas, surge la pregunta ¿cómo discernir cuál es el método más adecuado para un caso específico bajo un escenario de aplicación de la ingeniería inversa?. Esto hace evidente la necesidad de un instrumento, bien definido y fácil de comprender, que caracterice los métodos de ingeniería inversa en forma clara y completa, dando respuesta al interrogante planteado y permitiendo obtener el mayor beneficio posible del método seleccionado.

Una revisión de la literatura disponible sobre el tema, ha permitido identificar que las investigaciones sobre ingeniería inversa se han centrado principalmente en dos objetos de estudio: Los métodos y las herramientas (Tonella et al, 2007). Son múltiples los trabajos que se han realizado para caracterizar las herramientas de ingeniería inversa (Bellay y Harald, 1997; Guéhéneuc et al, 2006; Monroy et al, 2012). Sin embargo, una revisión detallada de la documentación ha revelado que el estudio y análisis de los métodos se ha hecho por separado para cada método (Tonella et al, 2007), siendo pocos y muy limitados en su intención los que tratan de caracterizarlos y evaluarlos (Gannod y Cheng, 1999; Jarzabek y Woon, 1997; Tonella et al, 2007). Uno de los primeros intentos por caracterizar los métodos y las herramientas de ingeniería inversa fue realizado por Jarzabek y Woon (1997), quienes propusieron un marco de referencia para tal fin. Sin embargo, en la presentación que hacen de los resultados, se centran en la descripción de un método heurístico que establece un mapeo formal entre el código fuente y los modelos de diseño recuperado, sin hacer un análisis sobre las características comunes entre los métodos existentes.

Dos años más tarde Gannod y Cheng proponen un marco de referencia para clasificar y comparar técnicas de ingeniería inversa y recuperación de diseño. En su intención por distinguir el nivel de rigurosidad y posibilidad de verificar la consistencia entre el código y el diseño, establecieron dos grandes grupos de métodos: Formales e Informales. Los primeros se soportan en la lógica matemática, utilizan lenguajes formales con especificación de sintaxis y semántica bien definidas y por lo tanto permiten una verificación formal de la consistencia entre el código y el diseño, en oposición a los otros, cuya representación es informal y consecuentemente no permiten hacer una verificación rigurosa de la consistencia entre el código y el diseño (Gannod y Cheng, 1999).

También definieron los términos: distancia semántica (número de niveles de abstracción que separan la entrada y la salida de un método o técnica de ingeniería inversa), precisión semántica (el nivel de confianza de que una especificación es correcta con respecto al código fuente de entrada), exactitud semántica (describe el nivel de detalle de una especificación y el grado en que la especificación es formal), trazabilidad semántica (describe el grado en el cual una especificación puede ser usada para reconstruir el programa equivalente) (Gannod y Cheng, 1999). En su trabajo dieron más atención a las herramientas que a los métodos y por lo tanto no llegaron a caracterizarlos en forma completa.

Además, existen varios estudios que se limitan a describir en forma detallada métodos específicos (Grant et al, 2011; Higo et al, 2011; Biegel y Diehl, 2010; Selim et al, 2010; Marcus et al, 2005; Wilde et al, 11997; Beck y Eichmann, 1993; Gallagher y Lyle, 1991), sin establecer un modelo general que permita la caracterización de cualquier método y a partir del cual se puedan hacer distintos tipos de análisis y evaluaciones.

Esta situación llevó a plantear como objetivo de investigación la definición de una propuesta metodológica para caracterizar los métodos de ingeniería inversa, con el propósito de discernir cuál es el método más adecuado para un caso específico bajo un escenario de aplicación de la ingeniería inversa, la cual fue construida haciendo uso de la técnica de revisión de características, obteniendo como resultado la definición de un instrumento que establece 14 características distribuidas en tres aspectos, que fue aplicado a diez métodos de ingeniería inversa para validar su utilidad.

METODOLOGÍA

La investigación se dividió en dos fases: la primera se centró en la caracterización de los métodos de ingeniería inversa y la segunda en la evaluación de los métodos a partir de la caracterización obtenida. La primera fase parte de la identificación de métodos de ingeniería inversa realizada por Tonela et al (2007). Luego se revisaron las fuentes bibliográficas que describen cada uno de estos métodos con el propósito de abstraer los atributos comunes, usando como referente guía los objetivos de la ingeniería inversa (Chikovsky y Cross, 1990): Lidiar con la complejidad, generar puntos de vista alternativos, recuperar información perdida, detectar efectos secundarios, sintetizar la información en un mayor nivel de abstracción y facilitar la reutilización. La segunda fase tuvo como fin validar la utilidad del instrumento de caracterización obtenido como resultado de la primera fase, para lo cual se caracterizaron diez métodos de ingeniería inversa aplicando el método de revisión de características de la metodología DESMET (Kitchenham, 1996), enfocado en aspectos cualitativos, el cual se basa en la exploración de documentación.

Para encontrar las fuentes de información se utilizó la biblioteca digital de la IEEE Computer Society, porque publica los artículos de las conferencias más importantes sobre ingeniería inversa: The Working Conference on Reverse Engineering (WCRE), The Conference on Software Maintenance and Reengineering (CSMR), The International Conference on Software Maintenance (ICSM) y The International Symposium on Empirical Software Engineering (ISESE), además de incluir las revistas especializadas en este campo de conocimiento: The Journal on Software Maintenance and Evolution: Research and Practice (JSME), The IEEE Transactions on Software Engineering (TSE), The ACM Transactions on Software Engineering Methodology (TOSEM) y Empirical Software Engineering: An International Journal (EMSE) (Tonella et al, 2007)..

RESULTADOS

Los métodos de ingeniería inversa identificados por Tonela et al (2007) son: Design Recovery, Code Browsing, Code Visualization, Annotation Processing, Traceability, Slicing, Impact Analysis, Concept/Feature Location, Clone Detection y Clustering. Para cada uno de estos métodos se identificaron las fuentes bibliográficas que los describen a partir de una búsqueda en la Digital Library que arrojó los resultados presentados en la tabla 1. El criterio de búsqueda fue que el título del documento incluyera el nombre del método y que el texto incluyera las palabras “reverse engineering”, para delimitar el campo de conocimiento. Los métodos que aparecen con asterisco (*) en la tabla arrojaron un resultado de cero artículos, por eso exclusivamente para ellos la búsqueda del nombre del método se hizo dentro del texto.

Tabla: Fuentes bibliográficas

Listado de métodos	Digital Library	Analizados
Design Recovery	19	7
Code Browsing*	25	2
Code Visualization*	39	6
Annotation Processing*	4	1
Traceability	44	11
Slicing	21	15
Clone Detection	32	7
Clustering	98	8
Impact Analysis	11	5
Concept Location	11	8
Total	304	70

Cada uno de los artículos encontrados se revisó para seleccionar sólo aquellos que de alguna manera describen el método correspondiente. Los artículos seleccionados fueron analizados, encontrando que cada

autor sólo brinda descripciones parciales sobre el método. Esta información se compiló en un documento por método, y al comparar estos documentos entre sí se identificaron 14 características comunes, las cuales se agruparon teniendo en cuenta tres aspectos inherentes a cualquier método: El primero, lo que el método necesita para cumplir su objetivo: Entrada, el segundo, el proceso que realiza el método como tal: Método y el tercero, el resultado que arroja el método: Salida (Ver figura 1). Cada una de las características se describe a continuación.

Entrada: Es la fuente que requiere el método para iniciar y alcanzar su propósito. Sus características son: El *tipo*, corresponde a la clase de fuente que utiliza el método, puede ser: Código fuente, colección de datos, documentos y conocimiento del experto. El *modelo* responde al meta-modelo que describe la entrada y su *representación* concierne a la forma como físicamente se muestra (Guéhéneuc et al, 2006), teniendo en cuenta que cada tipo tiene su respectivo modelo y representación (Monroy et al, 2012), por ejemplo, una entrada de tipo código fuente tiene un modelo determinado por el paradigma de programación y una representación establecida por el lenguaje de programación.

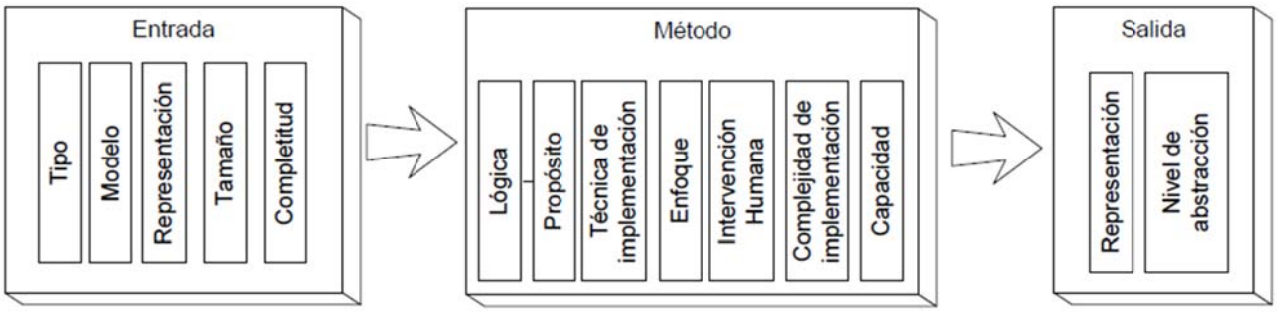

Fig. 1: Características de los métodos de ingeniería inversa

El *tamaño* de la entrada indica la cantidad de información que soporta el método en calidad de entrada (Torchiano et al. 2005). Esta característica determina la capacidad del método, depende mucho de su técnica de implementación y aplica sobre todo a los métodos que sólo reciben como tipo de entrada código fuente. La última característica de la entrada es la *completitud* que revela si el método requiere trabajar con toda la entrada o si permite hacerlo con algunas partes del sistema objeto de estudio.

Método: Es el procedimiento que se sigue para hacer el trabajo de ingeniería inversa, obedece a una *lógica* que describe el algoritmo de su aplicación, tiene un *propósito* que indica el alcance que establece como objetivo, un *enfoque* que expresa si está diseñado para recuperar la estructura del sistema (enfoque estático), su comportamiento (dinámico) o los dos (hibrido) al mismo tiempo, puede estar sujeto o no a la *intervención humana* para el logro de su propósito. Requiere de un nivel de conocimiento que debe poseer el ingeniero de software para poder aplicar el método sobre un sistema, lo que determina la *complejidad de implementación,* o sea el grado de esfuerzo que debe utilizarse para desarrollar una herramienta que aplique el método.

Dependiendo el tamaño de la entrada el método tiene una *capacidad*, que corresponde a la habilidad de trabajar con sistemas grandes (más de 20K líneas de código) o pequeños (Torchiano et al. 2005). Esta característica está determinada por la *técnica de implementación* del método, o sea la forma cómo las herramientas aplican el método.

Salida: Es el producto obtenido a partir de la aplicación del método como respuesta a su propósito. Está determinada por una representación y un nivel de abstracción. La *representación* de la salida se refiere a la manera específica de publicar los resultados del método para permitir su interpretación, implica la utilización de un lenguaje (escrito o gráfico) con reglas sintácticas, gramaticales y semánticas bien definidas para evitar ambigüedades en la interpretación. Este criterio de caracterización está directamente relacionado con la herramienta que lo implementa, ya que los resultados que produce son dependientes de la técnica de transformación empleada para la representación de la salida.

El *nivel de abstracción* es el grado de cercanía existente entre la realidad y su representación con base en un lenguaje definido sobre el que es posible visualizar, construir y documentar los artefactos de un sistema software (Gannod y Cheng, 1999). Como se observa en la figura 2, es básico si la salida produce resultados representados en código fuente, intermedio si corresponde al diseño y la arquitectura del sistema y superior si recupera conceptos del dominio del problema o identifica requerimientos.

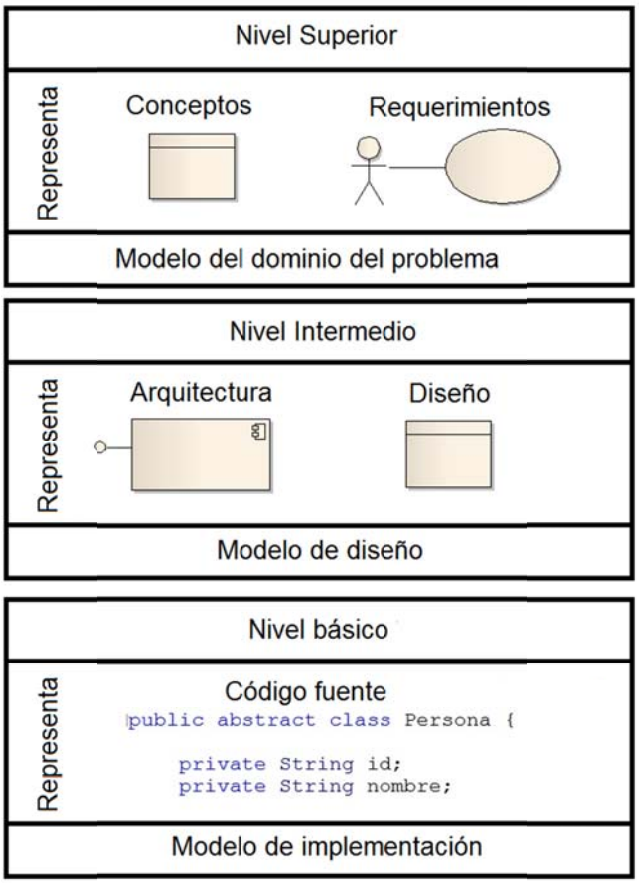

Fig. 2: Niveles de abstracción

ANÁLISIS DE LOS RESULTADOS

Para validar la utilidad del instrumento de caracterización obtenido, se aplicó a los diez métodos identificados por Tonella et al (2007), haciendo uso de la técnica de revisión de características, definida en la metodología DESMET (Kitchenham, 1996), la cual se basa en la exploración de documentación existente. El resultado se sintetiza en la tabla 2, que muestra en las zonas sombreadas los valores que toma cada método para las características que fueron valoradas. Al analizar la tabla se resaltan los siguientes aspectos:

i) El único método que trabaja con los tres tipos de entrada es Design recovery, Traceability utiliza como entrada código fuente y documentación, mientras que los demás utilizan sólo con código fuente.

ii) Los métodos Design recovery, Traceability y Slicing permiten trabajar con porciones parciales de la entrada y los demás lo hacen sólo si la entrada está completa.

iii) El único método que representa la salida en texto y gráfico es Slicing, en sólo texto la representan Code Browsing, Clone Detection y Concept Location, los demás lo hacen en forma gráfica acudiendo la mayoría de ellos al uso de grafos.

iv) Sólo el método Traceability permite llegar al nivel superior de abstracción en la salida, los métodos Design Recovery y Clustering llegan hasta el nivel de arquitectura, Code Visualization, Annotation Processing e Impact Analysis lo hacen hasta el diseño, mientras que los demás sólo llegan al nivel básico.

v) Los métodos Design recovery, Annotation Processing y Slicing requieren de la intervención humana.

vi) Se confirma que la mayoría de los métodos tienen un enfoque exclusivamente estático y ninguno es solamente dinámico

La Tabla 2 resume el resultado de la caracterización y sirve para seleccionar el método que mejor se ajuste a las características de análisis requeridas, haciendo más fácil esta tarea. Por ejemplo, si se pretende realizar un trabajo de ingeniería inversa que llegue hasta el mayor nivel de abstracción el método recomendado es Traceability, si se quiere llegar hasta la arquitectura se puede utilizar Design Recovery o

Clustering, si sólo se desea recuperar el diseño en bajo nivel del sistema se pueden aplicar los métodos Code Visualization, Annotation Processing o Impact Analysis, mientras que si se desea hacer análisis en el nivel más básico se pueden utilizar los métodos Code Browsing, Slicing, Clone Detection o Concept Location.

Por otra parte, el instrumento de caracterización obtenido sirve para confirmar seis criterios de clasificación de los métodos de ingeniería inversa, definidos previamente en la literatura sobre el tema, correspondientes a las características establecidas para el método como tal. El primer criterio obedece al enfoque y determina tres clases de métodos: dinámicos, estáticos e híbridos (Tonella et al, 2007; Canfora et al, 2011), dependiendo el aspecto que recupera del sistema: el comportamiento, la estructura o ambos, respectivamente. El segundo criterio identifica dos clases de métodos: por un lado aquellos que requieren de la intervención humana para lograr el propósito del método y por otro, los que no requieren de la intervención humana (Tonella et al, 2007; Canfora et al, 2011).

El tercer criterio hace referencia al nivel de complejidad de implementación del método, permite definir tres clases de métodos: Fáciles de implementar, difíciles de implementar y con un nivel intermedio de dificultad en su implementación (Guéhéneuc et al, 2006). En la revisión que se hizo de la literatura no se encontró un referente o una métrica para cuantificar esta característica, lo que implica que existe un alto grado de subjetividad al momento de valorarla.

Tabla 2: Caracterización de métodos de ingeniería Inversa

	Aspectos																
	Entrada					Método					Salida						
Características	Tipo			Completitud		Enfoque			Intervención humana		Representación			Nivel de abstracción			
Listado de métodos	Código fuente	Documentos	Conocimiento del experto	Completa	Incompleta	Dinámico	Estático	Híbrido	Si	No	Texto	Gráfico	Híbrido	Básico	Medio (Diseño bajo nivel)	Medio (Diseño alto nivel)	Superior
Design Recovery	■	■	■		■			■	■			■				■	
Code Browsing	■			■			■			■	■			■			
Code Visualization	■			■			■			■		■			■		
Annotation Processing	■			■			■		■			■			■		
Traceability	■	■			■		■			■		■					■
Slicing	■				■			■	■				■	■			
Clone Detection	■			■			■			■	■			■			
Clustering	■			■			■			■		■				■	
Impact Analysis	■			■				■		■		■			■		
Concept Location	■			■				■		■	■			■			

El cuarto criterio corresponde a la capacidad del método y determina dos clases: métodos para sistemas grandes (más de 20K líneas de código) y métodos para sistemas pequeños (Torchiano et al. 2005). Se debe resaltar que esta característica en la actualidad depende más de la técnica de implementación del método (la herramienta) que del mismo método. El quinto criterio corresponde al nivel de abstracción de la salida que se obtiene como resultado del método, se pueden clasificar en tres grupos: Básico, Intermedio y superior (Gannod y Cheng, 1999).

Por último, si se tiene en cuenta el nivel de rigurosidad del lenguaje que se utiliza para la representación de la salida y la posibilidad de verificar la consistencia entre la entrada y la salida, se confirma la clasificación establecida por Gannod y Cheng (1999), quienes definieron dos grandes grupos: métodos Formales e Informales.

CONCLUSIONES

Los resultados presentados permiten concluir lo siguiente: 1) La aplicación del instrumento propuesto facilita la selección del método más adecuado para un caso específico bajo un escenario de aplicación de la ingeniería inversa, 2) las características identificadas pueden ser utilizadas como criterios de clasificación de los métodos de ingeniería inversa, 3) también se pueden utilizar en calidad de plantilla para caracterizar cada uno de los métodos de ingeniería inversa existentes y de esta manera facilitar su estudio y comprensión.

AGRADECIMIENTOS

Este trabajo es parte de los resultados parciales de la tesis doctoral titulada Marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento, desarrollada por Martín Monroy Ríos en el programa de Doctorado de la Universidad del Cauca. Agradecemos a la Red Latinoamericana de Coordinación y Cooperación para unificar buenas prácticas de desarrollo de Software - Proyecto SPU CoCoNet-LA por su apoyo.

REFERENCIAS

Beck, J. y D. Eichmann, Program and Interface Slicing for Reverse Engineering. ICSE '93: 15th Conference on Software Engineering, 509 – 518, Baltimore, USA, 17 al 21 de mayo (1993).

Bellay, B. y G. Harald, A Comparison of four Reverse Engineering Tools, 4th Working Conference on Reverse Engineering, 2 – 11, Amsterdam, Holanda, 6 al 8 de octubre (1997).

Bennett, K. H. y V. T. Rajlich, Software maintenance and evolution: a roadmap. ICSE 2000: The 22nd International Conference on Software Engineering: Proceedings of the Conference on The Future of Software Engineering, 73-87, Limerick, Irlanda, 4 al 11 de junio (2000).

Biegel, B. y S. Diehl, Highly Configurable And Extensible Code Clone Detection. 17th Working Conference on Reverse Engineering WCRE, 237 – 24, Boston, USA, 13 al 16 de octubre (2010).

Canfora, G., Di Penta M. y L. Cerulo, Achievements and Challenges in Software Reverse Engineering, Communications of the ACM: 54(4), 142–151 (2011).

Chikofsky, E.J. y J. H. Cross, Reverse engineering and design recovery: a taxonomy, IEEE Software: 7(1), 13 – 17 (1990).

Cuenca, Ll., Ortiz A. y A. Boza, Desarrollo de una Herramienta Software para la Vista de Información de la Arquitectura CIMOSA. Información Tecnológica, 19(3), 97-106 (2008)

Demeyer, S., Ducasse S. y O. Nierstrasz, Object-Oriented Reengineering Patterns. 3 -10, Square Bracket Associates, Berna, Suiza (2008).

Favre, L., MDA-Based, Reverse Engineering, Reverse Engineering - Recent Advances and Applications, A.C. Telea, 55 -83, (2012).

Gallagher, K. B. y J. Lyle, Using Program Slicing in Software Maintenance. IEEE Transactions on Software Engineering: 17, 751 - 761 (1991).

Gannod, G. y B. Cheng, A Framework for Classifying and Comparing Software Reverse Engineering and Design Recovery Techniques, 6th Working Conference on Reverse Engineering, 77 – 88, Atlanta, Georgia USA, 6 al 8 de octubre (1999)

Grant, S., Cordy J. R., y D. B. Skillicorn, Reverse Engineering Co-maintenance Relationships Using Conceptual Analysis of Source Code. 18th Working Conference on Reverse Engineering, 87 - 91, Limerick, Irlanda, 17 al 20 de octubre (2011)

Guéhéneuc, Y., Mens K. y R. Wuyts, A Comparative Framework for Design Recovery Tools, Proceedings of the 10th European Conference on Software Maintenance and Reengineering, 134 – 143, Bari, Italia, 22 al 24 de marzo (2006).

Higo, Y., Yasushi U., Nishino M. y S. Kusumoto, Incremental Code Clone Detection: A PDG-based Approach. 18th Working Conference on Reverse Engineering, 3 - 12, Limerick, Irlanda, 17 al 20 de octubre (2011)

Jarzabek, S. e I. Woon, Towards a Precise Description of Reverse Engineering Methods and Tools. Software Maintenance and Reengineering, 3 – 9, Singapure, 17 al 19 de marzo (1997)

Kitchenham, B., DESMET. A method for evaluating Software Engineering methods and tools. Computing & Control Engineering Journal: 8(3), 120 – 126 (1997)

Marcus, A., Rajlich V., Buchta J., Petrenko, M. y A. Sergeyev, Static Techniques for Concept Location in Object-Oriented Code. 13th International Workshop on Program Comprehension, IWPC 2005. Proceedings, 33 – 42, St. Louis, MO, USA, 15 al 16 de mayo (2005)

Monroy, M., Arciniegas J. L. y J. Rodríguez, Caracterización de herramientas de ingeniería inversa. Información Tecnológica, 23(6), 31-42 (2012)

Platenius, M. C., Von Detten M. y B. Steffen, Archimetrix: Improved Software Architecture Recovery in the Presence of Design Deficiencies, 16th European Conference on Software Maintenance and Reengineering, 255 – 264, Szeged, Hungía, 27 al 30 de marzo (2012).

Selim, G., King C. F. y Z. Ying, Enhancing Source-Based Clone Detection Using Intermediate Representation. 17th Working Conference on Reverse Engineering WCRE, 227 -236, Boston, USA, 13 al 16 de octubre (2010).

Tonella, P., Torchiano M., Du Bois B. y T. Systä, Empirical studies in reverse engineering: state of the art and future trends, Springer Science + Business Media, 12(5), 551-571 (2007).

Torchiano, M., Ricca F. y P. Tonella, A comparative study on the re-documentation of existing software: Code annotations vs. drawing editors. International Symposium on Empirical Software Engineering, 277 – 286, Noosa Heads, Queensland, Australia,17 y 18 de noviembre (2005).

Van Geet, J., Ebraert P. y S. Demeyer, Redocumentation of a legacy banking system: an experience report. International Workshop on Principles of Software Evolution, 33–41, Antwerp, Bélgica, 20 y 21 de septiembre (2010).