Marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento

Anteproyecto Tesis de Doctorado

Martín Emilio Monroy Ríos

Director: PhD. José Luis Arciniegas Herrera

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Doctorado en Ingeniería Telemática

Línea: Servicios avanzados de telecomunicaciones

Popayán, Mayo de 2014

Contenido

1. Planteamiento del problema 3
2. Estado del Arte 5
3. Aporte Investigativo 10
4. Objetivos 11
4.1. Objetivo general 11
4.2. Objetivo específicos 11
5. Actividades y cronograma 11
5.1 Caracterización de las propuestas existentes 11
5.2 Definición de lineamientos 12
5.3 Modelado 12
5.3 Evaluación 13
6. Recursos, Presupuesto y fuentes de Financiación 15
7. Condiciones de entrega 15
8. Referencias bibliográficas 16
9. Acta de propiedad intelectual 25

1. Planteamiento del problema

Actualmente la mayoría de 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; von Detten, 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 (Nierstrasz et al, 2005).

La situación se hace más compleja debido al ritmo acelerado del cambio, que se manifiesta con el surgimiento de nuevos paradigmas y tecnologías, mayores exigencias de calidad y nuevos requerimientos (Lehman et al, 1997), que obliga a la mayoría de las compañías a enfrentarse al problema de tener que modernizar o reemplazar sus sistemas. Estos sistemas representan inversión económica, de tiempo y otros recursos. Muchos de ellos son críticos para el negocio de la organización y reemplazarlos representa un alto riesgo (Favre et al, 2009).

Cuando se trata de sistemas heredados que han sido desarrollados durante varios años, (en oportunidades más de dos décadas) que son complejos por su volumen y su heterogeneidad (combinan diferentes tecnologías), se debe garantizar la inversión que se ha hecho actualizándolos sin causar traumas en los procesos organizacionales (Jouault et al, 2009) y conservando la familiaridad del producto (Lehman et al, 1997).

Pero el problema no sólo se presenta en el proceso de mantenimiento. Con frecuencia se requiere mantener la trazabilidad de los distintos artefactos que se producen en el ciclo de desarrollo de software, por ejemplo, para efectos de control de calidad se requiere garantizar la coherencia entre el código fuente y los modelos arquitectónicos y de diseño propuestos (Müller et al, 2000).

Adicionalmente, se debe tener en cuenta que en los últimos años la práctica del desarrollo de software incluye la reutilización de código abierto, el cual se caracteriza por contar con poca o nula información sobre su arquitectura (Constantinou et al, 2011). Esto debe hacerse siempre y cuando se mantenga el debido respeto a los derechos de autor, lo que a su vez ha llevado a la necesidad de contar con herramientas que permitan la detección de clonación de código (Canfora et al, 2011).

Del mismo modo, las actividades ilegales en el ciberespacio afectan la seguridad nacional y amenazan el derecho de los ciudadanos a la privacidad, lo que tiene significantes implicaciones políticas, económicas y sociales (Choo, 2008). Estas actividades han sido patrocinadas por el crimen organizado, que ha contratado hackers que desarrollan software malicioso o malware (virus, troyanos, gusanos,

etc.) que se instala en el computador sin autorización del propietario, lo que exige mecanismos rápidos que faciliten el análisis y el entendimiento del código malicioso para dar una respuesta oportuna sobre cómo bloquear y eliminar una determinada pieza de malware (Treude et al, 2011).

Por otra parte, existe una nueva área de aplicación en la Ingeniería dirigida por modelos (MDE Model Driven Engineering), adicional a la ingeniería directa, donde los sistemas han sido creados a partir de modelos, y a la ingeniería inversa, donde los modelos se extraen a partir del sistema, denominada modelos en tiempo de ejecución, que consiste en que el sistema y los modelos que lo representan deben coexistir en el tiempo y mantenerse sincronizados (Jouault et al, 2009).

Por último, pero no menos importante, la industria del desarrollo de software se encuentra en crecimiento y necesita de especialistas altamente calificados, sin embargo, la gran mayoría de instituciones académicas utilizan métodos tradicionales en los procesos de aprendizaje de la ingeniería de software, desaprovechando la oportunidad de aprender a partir del análisis de productos software existentes (Muhammad, 2005).

La ingeniería inversa se ha convertido en la principal solución a las situaciones expuestas, relacionadas con la adquisición de conocimiento para la reconstrucción de documentación a nivel de diseño y arquitectura, 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 autorizado, 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; lo que ha hecho que la ingeniería inversa sea en uno de los mayores retos actuales de la ingeniería de software (Favre et al, 2009).

Las dos últimas décadas han representado grandes avances para la ingeniería inversa (Canfora et al, 2011; Tonella et al, 2007), sin embargo, aún existen desafíos que le impiden dar solución de manera completa a las situaciones expuestas. Aunque se han desarrollado métodos y herramientas que permiten recuperar distintas vistas del sistema, la mayoría realizando análisis estático y en menor cantidad haciendo análisis dinámico (Monroy et al, 2012; Tonella et al, 2007), la capacidad para consultar la información obtenida en los artefactos (piezas de información) generados por los analizadores es muy limitada (Canfora et al, 2011).

La utilidad real de un proceso de ingeniería inversa se evidencia en el uso que se le dé a los artefactos recuperados (Canfora et al, 2011; Tonella et al, 2007; Müller et al, 2000), lo que exige capacidad para consultar la información obtenida en los artefactos generados, para que sea posible realizar cualquier tipo de análisis. Esta situación lleva a plantearse la siguiente pregunta:

¿Cómo extraer información a partir de los artefactos generados en un proceso de ingeniería inversa, para realizar análisis a nivel de arquitectura sobre el comportamiento de un sistema software?

En este orden de ideas, al permitir realizar análisis sobre la información obtenida a partir de un proceso de ingeniería inversa se contribuye al logro del principal objetivo de esta disciplina, porque se facilita: la adquisición de conocimiento de sistemas implementados a través de la reconstrucción de documentación a nivel de diseño y arquitectura, 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 autorizado, el análisis y entendimiento del código malicioso, la validación de la coexistencia sincronizada del sistema y los modelos que lo representan, además que permitiría apoyar el aprendizaje de la ingeniería de software.

Por lo tanto, al responder la pregunta planteada se beneficia la comunidad científica dedicada al estudio de la ingeniería de software, por el aporte que representa para la ingeniería inversa el dar respuesta a uno de los desafíos que afronta actualmente (Canfora et al, 2011). Esto a su vez se traduce en un beneficio para la comunidad dedicada al desarrollo de software, porque se facilita el control de la calidad del producto software al permitir una mejor trazabilidad de los artefactos que lo conforman y la validación de la coexistencia sincronizada del sistema y los modelos que lo representan. De igual manera, facilita la reutilización de código y el control de clonación no autorizado.

Por otra parte, también podrían recibir beneficios los especialistas en seguridad, en la medida en que se facilita el análisis y entendimiento del código malicioso, haciendo más eficiente y productivo su trabajo. No menos importante es el beneficio que recibe la comunidad académica, cuyo tema de estudio es la ingeniería de software, porque se facilita el aprendizaje a través de técnicas de ingeniería inversa sobre productos software existentes, que sirvan como referente de estudio y sobre los cuales se pueden realizar distintos tipos de análisis.

Por todo lo anterior, se beneficia la industria del software, uno de los pocos sectores de la economía que no se ha visto afectado por la crisis (SiliconWeek, 2012; Fedesoft, 2012), lo que la convierte en una posibilidad para las economías emergentes, como es el caso de Colombia, manteniendo coherencia con la política del gobierno, reflejada en el marco del Plan Vive Digital y del Programa Nacional de Gobierno en Línea del Ministerio de Tecnologías de la Información y las Comunicaciones MINTIC, que plantea apoyar el programa de Transformación Productiva para los sectores de software y tecnologías de la información (Ministerio de las TIC, 2011).

2. Estado del Arte

Los sistemas software requieren de cambios continuos y mejoras para satisfacer los nuevos requerimientos y expectativas de los usuarios, con el propósito de

adaptarlos a los nuevos modelos de negocio, a los cambios de legislación y a los avances tecnológicos para evitar su devaluación (Lehman y Belady, 1985), lo que ha generado que la mayor parte del presupuesto se dedique al mantenimiento (IEEE Computer Society, 2004). Para realizar cualquier modificación del producto software es indispensable previamente comprenderlo, esto implica adquirir el conocimiento necesario sobre los artefactos que conforman el sistema.

Sin embargo, en la mayoría de los casos, la comprensión del software se convierte en un desafío por la falta de documentación adecuada y actualizada, con frecuencia debido a limitaciones de recursos y tiempo durante la construcción del producto o porque el equipo de desarrollo no cuenta con las habilidades necesarias para elaborarla (Canfora et al, 2011). Es por eso que la ingeniería inversa surge como técnica de mantenimiento (Chikofsky y Cross, 1990), con el fin de recuperar el conocimiento sobre un sistema software existente para facilitar su comprensión (Tonella et al, 2007; Rugaber y Stirewalt, 2004),

El proceso de ingeniería inversa comprende un momento inicial que corresponde a la captura de la información de entrada: constructos de código fuente, información simbólica textual, información dinámica, organización física, organización humana, información histórica, conocimiento del experto, estilos arquitectónicos, puntos de vista arquitectónicos (Ducasse y Pollet, 2009). Luego esta información es analizada con el fin de obtener una representación del sistema en un mayor nivel de abstracción, logrando como resultado una base de información, la cual a su vez es tratada para conseguir nuevas vistas del producto (Chikofsky y Cross, 1990).

Hasta el momento la mayor parte de las soluciones propuestas se han centrado en la recuperación y visualización de los artefactos a nivel de diseño (Canfora et al, 2011; Gupta, 2011; Qiu et al, 2010; Dong et al, 2009; Lee et al, 2007), en el desarrollo de herramientas para la extracción del árbol abstracto de sintaxis (AST Abstrac Syntax Tree) (Chen et al, 1990), el entendimiento y la reestructuración de bases de datos existentes (Blaha, 1997), el desarrollo y mejoramiento de métodos de análisis estático como la técnica de corte (De Lucia, 2001; Weiser, 1984) y el desarrollo de herramientas (Grammatech Inc, 2002), así como su caracterización y clasificación (Monroy et al, 2012; Guéhéneuc et al, 2006; Bellay y Gall, 1997). Incluso se han presentado propuestas que van hasta el dominio del problema y los requerimientos, a lo que se ha denominado Arqueología de software (Perez, 2012; Booch, 2011).

La reconstrucción de la arquitectura es un enfoque de la ingeniería inversa que consiste en el proceso de construir las vistas arquitectónicas de un producto software a partir de su implementación (Ducasse y Pollet, 2009). En la literatura se utilizan diferentes términos que hacen referencia a lo mismo: arquitectura inversa, extracción, minería, recuperación o descubrimiento de la arquitectura, teniendo en cuenta que los dos últimos términos son más específicos, mientras que la recuperación es un proceso de abajo hacia arriba, el descubrimiento hace referencia a un proceso de arriba hacia abajo (Medvidovic et al, 2003)

A partir de una revisión sistemática de literatura (Kitchenham et al., 2010) se han identificado 175 documentos, utilizando como términos clave los mencionados previamente, en las bases de datos: IEEE, ACM, Science Direct, Springer y SEI (Software Engineering Institute), de los cuales sólo 16 (9,14%) recuperan aspectos dinámicos de la arquitectura y 3 (1,7%) recuperan aspectos dinámicos y estáticos al mismo tiempo, lo que evidencia que se ha centrando más la atención en los aspectos estáticos del sistema (Platenius et al, 2012; von Detten, 2012; Constantinou et al, 2011; Roy y Graham, 2008; Lee et al, 2007) y se ha relegado a un segundo plano la recuperación del comportamiento del sistema, debido al mayor nivel de complejidad que representa (Martínez et al, 2011; Pirzadeh y Hamou-Lhadj, 2011; Wang et al, 2009).

Las propuestas orientadas a abstraer artefactos de bajo nivel y reconstruir información de alto nivel, han logrado refactorización (Schmidt et al., 2012; Terra et al., 2012), recuperar capas arquitectónicas (Cai et al., 2013; El Boussaidi et al, 2012; Scanniello et al, 2010), componentes y conectores (Platenius et al, 2012; von Detten, 2012; García et al, 2011; Lungu et al, 2006; Hassan y Holt, 2004; Koschke, 2000; Siff y Reps, 1999); aplicando técnicas de agrupación (Cai et al., 2013; Risi et al, 2012; Wang et al, 2010; Scanniello et al, 2010; Maqbool y Babri, 2007), métodos probabilísticos (Castrejón y Lozano, 2012), modelos de características (Acher et al, 2011; Haslinger et al 2011), minería de datos (Vasconcelos y Werner, 2011; Yang y Riva, 2006; Sartipi et al, 2000) y coincidencia de patrones (Heesch et nal. 2012; Sartipi y Kontogiannis, 2001). Todas estas propuestas se han centrado en la recuperación de vistas arquitectónicas estructurales.

Sartipi y Dezhkam (2007) hacen una propuesta de carácter híbrido que fusiona vistas estáticas y dinámicas, en las que la recuperación de la estructura está basada en patrones, identificando componentes de alto nivel, mientras que las interconexiones se definen utilizando un lenguaje de consulta de arquitectura (Architecture Query Languaje) definido por ellos; entre tanto, Riva y Rodríguez (2002) proponen una técnica para combinar el análisis de la información arquitectónica estática y dinámica para soportar las tareas de recuperación de la arquitectura, enfatizando en la selección correcta de los conceptos significativos de la arquitectura para el proceso de reconstrucción y soportándose en técnicas de abstracción para su manipulación. En los dos casos no se recuperan aspectos relacionados con la lógica de la aplicación a nivel de arquitectura.

La recuperación de vistas arquitectónicas de comportamiento se ha logrado a través de estrategias como el registro de anotaciones en partes especiales del código fuente para capturar momentos específicos de la ejecución de la aplicación, a partir de los cuales se pueden construir diagramas de objetos que presentan las relaciones existentes en un momento determinado, permitiendo incluso establecer jerarquías (Abi-Antoun et al, 2010; Abi-Antoun y Aldrich, 2009; Abi-Antoun y Aldrich, 2008; Weijun et al, 2008). Al analizar detalladamente estos trabajos, se hace evidente que se está recuperando el diseño y no la arquitectura,

además, el comportamiento que se obtiene se limita a la información que puede brindar un modelo estructural como es el diagrama de objetos.

Otras investigaciones se centran en la representación de los diagramas de secuencia, que se obtienen a partir de la interacción del usuario final con la interface gráfica, omitiendo cualquier aspecto relacionado con la lógica de la aplicación (Amalfitano et al, 2010; Grati et al, 2010; Alalfi et al, 2009). Por otra parte, Ziadi et al (2011) proponen un sistema de representación transitorio denominado LTS (Labeled Transition Systems), el cual permite definir mezclas de trazas de ejecución como mezclas LTS usando un algoritmos definido por los autores. Aunque estos trabajos presentan vistas de comportamiento a través de diagramas de secuencia, sólo lo hacen a nivel de diseño y no de arquitectura.

Qingshan et al. (2005) proponen obtener la estructura del grafo de procesos de sistemas software orientados a objetos, recuperando de esta manera información dinámica a partir de información estática, para lo cual primero se identifica un fragmento estático de código correspondiente a un proceso dinámico, luego un algoritmo de mapeo identifica la correspondencia entre el proceso dinámico y los módulos estáticos. Este algoritmo se fundamenta en un algoritmo incremental de construcción de la estructura del grafo de procesos (PSG) y en un algoritmo de corte para la estructura de las clases. Algo similar plantean Schmerl et al. (2006 y 2005) y Yan et al. (2004), quienes definen una técnica que usa observación en tiempo de ejecución para construir una vista arquitectónica del sistema, estableciendo un mapeo fundamentado en algunos aspectos comunes (regulares) entre la implementación y los estilos arquitectónicos. El inconveniente de esta estrategia radica en que el mapeo hace que la propuesta se limite a las tecnologías existentes.

También se han encontrado propuestas que se centran principalmente en los temas de la extracción de datos y el modelo de abstracción, teniendo en cuenta la característica dinámica de la orientación a objetos del software, proporcionando un grupo de modelos, mecanismos y algoritmos que se pueden utilizar para extraer información dinámica y modelos abstractos de alto nivel de este tipo de sistemas (Qingshan, 2005; Huang et al. 2004; Huang et al. 2006); pero limitan su representación a lenguajes de descripción de arquitecturas que no son considerados estándares. Otras propuestas recuperan parcialmente elementos de las vistas arquitectónicas en UML, identificando casos de uso y construyendo un modelo básico al utilizar enrejado de conceptos (concept lattice) (Bojic y Velasevic, 2000; Ivkovic y Godfrey, 2002), pero no recuperan aspectos relacionados con la interacción de los componentes.

En los últimos años ha tomado fuerza el concepto de arquitectura en tiempo de ejecución (Runtime software architecture RSA) (Song et al. 2011), cuya finalidad es mantener una conexión causal entre la arquitectura y el sistema, asegurando que la arquitectura representa al sistema actual y que las modificaciones en la arquitectura causan los correspondientes cambios en el sistema (Nour et al., 2012). Para lograrlo Song et al. (2011) han propuesto un mecanismo de

sincronización entre la configuración de la arquitectura y los estados del sistema utilizando una transformación bidireccional que establece un puente para los vacios que hay a nivel de abstracción entre la arquitectura y el sistema, diseñando un algoritmo orientado a solucionar los conflictos entre la arquitectura y los cambios del sistema. El principal inconveniente de estas propuestas radica en que la representación de la arquitectura se realiza a través de grafos, lo cual es más apto mostrar aspectos estáticos del sistema.

Desde otro enfoque Vasconcelos A. y C. Werner (2011) utilizan análisis dinámico y minería de datos para apoyar el entendimiento y reutilización de software, a partir de la detección de clases cohesivas que implementan un conjunto de funcionalidades relacionadas, lo que permite interpretarlas como elementos arquitectónicos. Además desarrollan la herramienta ArchMine para ser integrada a un entorno de desarrollo, complementando el enfoque de evaluación basado en inspección (ArqCheck), que evalúa la conformidad de los elementos de la arquitectura con los requerimientos arquitectónicos no funcionales más representativos, y el enfoque que permite la detección de cambios en el dominio (ArchToDSSA) propuesto por los mismos autores. Como resultado del análisis dinámico recuperan escenarios a través de casos de uso, la vista dinámica con diagramas de secuencia y la vista estática con diagramas de paquetes en UML.

Sin embargo, el reto más grande para la recuperación de arquitectura es la interpretación de los resultados producto del proceso de ingeniería inversa y el descubrimiento de los modelos de software de manera automática, sin intervención humana. (Schrettner et al, 2010). La representación gráfica es el recurso más utilizado para lograr la interpretación de los resultados, pero tiene el inconveniente del manejo del espacio, el cual se hace más evidente cuando se está trabajando con sistemas complejos (Canfora et al, 2011).

Como es el caso de la propuesta de Zao y Zhang (2008), en la que presentan una gramática gráfica basada en un marco de la ingeniería inversa para la verificación del comportamiento de sistemas de alta seguridad, lo que convierte el problema de verificación de un programa en un problema de análisis de un lenguaje visual, es decir se analiza la representación gráfica del comportamiento del programa con restricciones especificadas por el usuario y las propiedades expresadas en una gramática gráfica. Este enfoque permite a los desarrolladores comprobar la secuencia de llamadas a métodos correspondientes a las especificaciones de algunos de los requisitos, siempre y cuando comprendan la gramática y encuentren una herramienta de visualización que le permita fácilmente ubicar las partes de interés particular. No obstante, en la revisión realizada no se ha encontrado ninguna herramienta para esta propuesta.

Adicionalmente, existen múltiples técnicas y métodos de análisis de arquitecturas, que permiten hacer evaluación de impacto a través de escenarios (Goeminne y Mens, 2010; Störmer, 2007), que utilizan algoritmos genéticos para hacer predicciones de rendimiento a partir de ingeniería inversa (Krogmann et al, 2010) y para detectar comportamientos anómalos (Platenius et al, 2012; von Detten, 2012;

Wong et al, 2010; Lanza y Ducasse, 2003), otras utilizan técnicas visuales para comparar arquitecturas (Beck y Diehl, 2010; Van den Brand et al, 2010) o para permitir su evolución (Lungu et al., 2012), por mencionar algunas. El problema consiste en que no están integradas y por lo tanto requieren de diferentes procesos y herramientas, además utilizan diferentes formas de representación de la arquitectura, haciendo más compleja esta tarea (Buchgeher y Weinreich, 2010; Petrov y Buy, 2011), sobre todo cuando se trata de arquitecturas recuperadas en procesos de ingeniería inversa.

La revisión sistemática de literatura y el análisis de los documentos encontrados, han permitido confirmar que, si bien se han logrado avances representativos en el campo de la ingeniería inversa, aún existen retos por superar, tendientes a mejorar la capacidad para: a) hacer análisis de caja negra, b) visualizar apropiadamente la información obtenida, c) consultar la base de información obtenida en los artefactos generados por los analizadores, y d) abstraer artefactos de bajo nivel y reconstruir información de alto nivel (Canfora et al, 2011); lo que ha motivado a convertir los dos últimos aspectos en objeto de estudio de la presente investigación.

En este orden de ideas, se evidencia que los trabajos encontrados en su mayoría permiten la recuperación de vistas arquitectónicas estructurales y en un menor grado de comportamiento. Además, el análisis que se realiza sobre la base de información obtenida a partir de los procesos de ingeniería inversa es limitado, debido a la heterogeneidad de los procesos, las herramientas y las formas de representación de la arquitectura.

El marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento que se propone, definirá una técnica para abstraer artefactos de bajo nivel y reconstruir información de alto nivel sobre el comportamiento de sistemas software, fundamentado en el estándar ISO/IEC/IEEE 42010 para la descripción de arquitectura y el estándar del Lenguaje Unificado de Modelado UML, utilizando el lenguaje XML para garantizar la integración con otras propuestas, al mismo tiempo que establecerá un mecanismo de consulta para realizar análisis sobre dicha información, presentando de esta manera una solución al problema planteado.

3. Aporte Investigativo

El principal aporte investigativo de la tesis doctoral estará representado por un marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento, conformado por:

- Un modelo conceptual para abstraer artefactos de bajo nivel y reconstruir información de alto nivel sobre el comportamiento de sistemas software, fundamentado en el estándar ISO/IEC/IEEE 42010 para la descripción de arquitectura y el estándar del Lenguaje Unificado de Modelado UML,

utilizando el lenguaje XML para garantizar la integración con otras propuestas.

- Un mecanismo de consulta para realizar análisis a nivel de arquitectura sobre el comportamiento de sistemas software, a partir de la información base obtenida en los artefactos generados por los analizadores en el proceso de ingeniería inversa

- Una herramienta que implemente el modelo conceptual y el mecanismo de consulta propuestos.

- Un modelo de caracterización de las herramientas de ingeniería inversa

- Un modelo de caracterización de las técnicas de ingeniería inversa.

4. Objetivos

4.1. Objetivo general

Definir un marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento de sistemas software.

4.2. Objetivo específicos

- Definir un modelo conceptual para abstraer artefactos de bajo nivel y reconstruir información de alto nivel sobre el comportamiento de sistemas software, integrando técnicas de recuperación de arquitectura.

- Diseñar un mecanismo de consulta para realizar análisis a nivel de arquitectura sobre el comportamiento de sistemas software, a partir de la base de información obtenida en los artefactos generados por los analizadores en el proceso de ingeniería inversa.

- Evaluar el modelo conceptual y el mecanismo de consulta mediante la construcción de un prototipo y la realización de un estudio de caso de ingeniería inversa.

5. Actividades y cronograma

Para el cumplimiento de los objetivos específicos planteados en el proyecto se realizarán las siguientes fases:

5.1 Caracterización de las propuestas existentes

Esta fase comprende tres aspectos: la caracterización de herramientas de ingeniería inversa, la caracterización de métodos de ingeniería inversa y la caracterización de los mecanismos de consulta. Para cada uno se aplicarán los siguientes pasos:

1. Hacer un inventario de las propuestas (herramientas y métodos de ingeniería inversa, mecanismos de consulta) existentes a partir de estudio de mapeo (Kitchenham et al, 2010)
2. Hacer un inventario de los enfoques de caracterización existentes a partir de estudio de mapeo (Kitchenham et al, 2010)
3. Integrar los enfoques de caracterización existentes aplicando la técnica de comparación (benchmarking) entre ellos.
4. Identificar los aspectos (características) que no cubre el resultado de la integración de los enfoques de caracterización existentes.
5. Definir una estructura de caracterización
6. Evaluar la utilidad del instrumento de caracterización aplicándolo en un escenario real de uso.

5.2 Definición de lineamientos

En esta fase se establecerán los lineamientos que deben cumplir tanto el modelo conceptual para abstraer artefactos de bajo nivel y reconstruir información de alto nivel sobre el comportamiento de sistemas software, como el mecanismo de consulta para realizar análisis a nivel de arquitectura sobre el comportamiento de sistemas software, así como la herramienta que se construirá. Para lograrlo se realizarán las siguientes actividades, aplicadas a cada uno de estos elementos:

1. Evaluar las propuestas existentes aplicando el instrumento de caracterización obtenido en la fase previa.
2. Identificar las brechas a partir de los resultados de la evaluación hecha.
3. Establecer los lineamientos teniendo en cuenta el alcance de la tesis doctoral.

5.3 Modelado

En esta fase se definirá el modelo conceptual para abstraer artefactos de bajo nivel y reconstruir información de alto nivel sobre el comportamiento de sistemas software, integrando las técnicas de recuperación de arquitectura que se ajusten a los requisitos establecidos para el modelo, según los resultados obtenidos en la primera fase.

De igual manera se diseñará el mecanismo de consulta para realizar análisis a nivel de arquitectura sobre el comportamiento de sistemas software, a partir de la información base obtenida en los artefactos generados por los analizadores en el proceso de ingeniería inversa.

Para lograr cada uno de estos propósitos se aplicarán los siguientes pasos (Grant et al., 2001):

1. Definir los objetivos (del modelo y del mecanismo de consulta)
2. Definir los límites del sistema de interés (modelo / mecanismo de consulta)
3. Clasificar los componentes del sistema de interés
4. Identificar las relaciones de los componentes del sistema de interés
5. Representar de manera formal el modelo conceptual / mecanismo de consulta.
6. Describir los patrones esperados del comportamiento del modelo conceptual / mecanismo de consulta.

5.3 Evaluación

Para garantizar la calidad de los resultados obtenidos se evaluará el modelo conceptual y el mecanismo de consulta mediante la construcción de un prototipo y la realización de un estudio de caso de ingeniería inversa, aplicando los siguientes pasos (Kitchenham, 1996):

1. Identificar el contexto del estudio de caso
2. Definir y validar la hipótesis
3. Seleccionar los proyectos a los cuales se le aplicará el estudio de caso
4. Identificar el método de comparación
5. Minimizar el efecto de los factores de confusión
6. Definir el plan de estudio de caso
7. Supervisar el estudio de caso a partir del plan definido
8. Analizar los resultados y emitir el reporte

En cuanto a la herramienta de ingeniería inversa, se construirá usando la metodología de desarrollo de software RUP (Jacobson et al., 2000) porque es centrada en la arquitectura, aplicando cada una de sus fases: Inicio, elaboración, construcción y transición.

Las actividades a desarrollar se presentan en el cronograma contenido en la siguiente tabla.

Tabla 1 Cronograma de actividades

Fase	Actividades	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24
Caracterización de las propuestas existentes	Inventario de propuestas existentes																								
	Inventario de enfoques de caractarización existente																								
	Análsis y evaluación de propuestas existentes																								
	Integración de los enfoques de caracterización existentes																								
	Identificación de los aspectos que no cubre el resultado de la integración de los enfoques de caracterización existentes																								
	Definición de la estructura de caracterización																								
	Evaluación de la utilidad del instrumento de caracterización																								
Definición de lineamientos	Evaluación de las propuestas existentes																								
	Identificación debrechas																								
	Establecimiento de los lineamientos																								
Modelado	Definición de los objetivos																								
	Definición de los límites del sistema de interés																								
	Clasificación de los componentes del sistema de interés																								
	Identificación de las relaciones de los componentes del sistema de interés																								
	Representación formal del modelo conceptual																								
	Representación formal del mecanismo de consulta.																								
	Descripción de los patrones esperados del comportamiento del modelo conceptual																								
	Descripción de los patrones esperados del comportamiento del mecanismo de consulta																								
Evaluación	Identificación del contexto del estudio de caso																								
	Definición y validación de la hipótesis																								
	Seleccionación de los proyectos a los cuales se le aplicará el estudio de caso																								
	Identificación del método de comparación																								
	Minimización del efecto de los factores de confusión																								
	Definición del plan de estudio de caso																								
	Supervisación del estudio de caso																								
	Analisis de los resultados																								
	Generación del reporte de evaluación																								
	Construcción de la herramienta																								
	Fase de inicio																								
	Fase de elaboración																								
	Fase de construcción																								
	Fase de transición																								
	Pasantía internacional																								
	Presentación de resultados a través de publicaciones																								
	Elaboración del documento final de tesis																								
	Sustentación de la tesis																								

6. Recursos, Presupuesto y fuentes de Financiación

En la siguiente tabla se presenta el presupuesto del proyecto, elaborado atendiendo los parámetros establecidos por la Facultad de ingeniería Electrónica y Telecomunicaciones (FIET). La Universidad de Cartagena me ha aprobado un 50% de descarga académica para la realización de estudios doctorales, además me brinda la subscripción a bases de datos bibliográficas y la oportunidad de participar en las convocatorias internas para financiación de publicaciones y movilidad para la realización de pasantías.

Tabla 2 Presupuesto del proyecto

Rubros	Investigador	Univesidad del Cauca	Universidad de Cartagena	Total
	Efectivo	Especie	Especie	
Recurso Humano				**$ 67.011.960,00**
Director del proyecto		$ 20.040.960,00		
Estudiante de doctorado	$ 11.742.750,00		$ 35.228.250,00	
Recursos Técnicos				**$ 16.500.000,00**
Recursos Hardware	$ 2.000.000,00		$ 500.000,00	
Recursos Software	$ 5.000.000,00		$ 2.000.000,00	
Recursos bibliográficos			$ 5.000.000,00	
Recursos varios	$ 2.000.000,00			
Viajes y salidas de campo	$ 13.000.000,00			**$ 13.000.000,00**
Servicios Técnicos	$ 5.000.000,00			**$ 5.000.000,00**
Publicaciones			$ 4.000.000,00	**$ 4.000.000,00**
Otros				**$ 39.000.000,00**
Matrícula	$ 24.000.000,00			
Pasantía Internacional	$ 3.000.000,00		$ 12.000.000,00	**$ 15.000.000,00**
AUI	$ 6.574.275,00			**$ 6.574.275,00**
Total	**$ 72.317.025,00**	**$ 20.040.960,00**	**$ 58.728.250,00**	**$ 166.086.235,00**

7. Condiciones de entrega

Como resultado del desarrollo del proyecto de investigación se obtendrán los siguientes productos:

1. Documento de monografía
2. Publicaciones:
    a. 3 artículos en revistas A1.
    b. 2 artículos en congresos internacionales.
3. Herramienta software que implementa el modelo conceptual y el mecanismo de consulta definidos.
4. Manuales de la Herramienta construida.
    a. Manual del sistema de la herramienta software.

b. Manual de usuario de la herramienta software.

8. Referencias bibliográficas

Abi-Antoun M. and J. Aldrich. A Field Study in Static Extraction of Runtime Architectures. 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Atlanta, Georgia USA, 2008, pp. 22–28.

Abi-Antoun M. and J. Aldrich. Static extraction and conformance analysis of hierarchical runtime architectural structure using annotations. 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications, Orlando, Florida, USA, 2009, pp. 321–340.

Abi-Antoun M., Selitsky T. and T. LaToza. Developer refinement of runtime architectural structure. ACM/IEEE 32th International Conference on Software Engineering, Cape Town, South Africa, 2010, pp. 80–87.

Acher M., Cleve A., Collet P., Merle P., Duchien L.a and P. Lahire. Reverse Engineering Architectural Feature Models, ECSA'11 Proceedings of the 5th European conference on Software architecture, Verlag Berlin, 2011, pp. 220–235.

Alalfi M. H., Cordy J. R. and T. R. Dean. Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications. IEEE International Conference on Software Testing Verification and Validation Workshops, Denver, Colorado, 2009, pp. 287–294.

Ali N., J. Rosik J. and J. Buckley. Characterizing real-time reflexion-based architecture recovery: an in-vivo multi-case study, 8th international ACM SIGSOFT conference on Quality of Software Architectures, Bertinoro, Italy, 2012, pp. 23–32.

Amalfitano D., Fasolino A. R. and P. Tramontana. An Iterative Approach for the Reverse Engineering of Rich Internet Application User Interfaces," presented at the 2010 Fifth International Conference on Internet and Web Applications and Services, Barcelona, Spain, 2010, pp. 401–410.

Beck F. and S. Diehl, Visual Comparison of Software Architectures, SOFTVIS'10, Salt Lake City, Utah, USA, 2010, pp. 183 - 192.

Beck M, and J. Döllner. Towards Automated Analysis and Visualization of Distributed Software Systems. SOFTVIS'10, Salt Lake City, Utah, USA, 2010, pp. 213 – 214.

Bellay B. and H. Gall. A Comparison of four Reverse Engineering Tools. Fourth Working Conference on Reverse Engineering (WCRE '97), Amsterdam, Netherlands, 1997, pp. 2 – 13.

Blaha M. R. Dimensions of database reverse engineering. Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE ’97), Amsterdam, Netherlands, 1997, pp. 176 –180.

Bennett K. H. and V. T. Rajlich. Software maintenance and evolution: a roadmap. ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, 2000, pp 73-87.

Bojic D. y D. Velasevic. A use-case driven method of architecture recovery for program understanding and reuse reengineering. Conference on Software Maintenance and Reengineering, Washington, DC, USA, 2000, pp. 23 – 32.

Booch G. The Soul of a New Watson. IEEE Software, vol. 28, no. 4, pp. 9 – 10, Jul. 2011.

Buchgeher G. and R. Weinreich. An Approach for Combining Model-based and Scenario-based Software Architecture Analysis. Fifth International Conference on Software Engineering Advances, Nice, 2010, pp. 141 – 148.

Cai Y., Wang H, Wong S and Linzhang Wang, Leveraging design rules to improve software architecture recovery. International ACM Sigsoft conference on Quality of software architectures, Vancouver, BC, Canada., 2013.

Canfora G. and M. Di Penta. New Frontiers of Reverse Engineering. Future of Software Engineering, Minneapolis, MN, USA, 2007, pp. 326 – 341.

Canfora G., Di Penta M. and L. Cerulo. Achievements and Challenges in Software Reverse Engineering, communications of the ACM, vol. 54, n.º 4, pp. 142–151, abr-2011.

Castrejón J y R. Lozano. Web2MexADL: Discovery and Maintainability Verification of Software Systems Architecture. European Conference on Software Maintenance and Reengineering, IEEE Computer Society. 2012. pp. 531 – 534

Chen Y.-F., Nishimoto M.Y, and C.V Ramamoorthy. The C information abstraction system. Software Engineering, IEEE Transactions on, vol. 16, no. 3, pp. 325 – 334, Mar. 1990.

Chikofsky E. and J. Cross. Reverse Engineering and Design Recovery: a Taxonomy. IEEE Software, vol. 7, no. 1, pp. 13–17, Jan-1990.

Choo K.-K. Organised crime groups in cyberspace: a typology, Trends in Organized Crime, vol. 11, pp. 270–295, 2008.

Constantinou E., Kakarontzas G. and I. Stamelos. Towards Open Source Software System Architecture Recovery Using Design Metrics. Panhellenic Conference on Informatics, Kastonia, 2011, pp. 166 – 170.

De Lucia A. Program slicing: Methods and applications. 1st IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2001), Florence, Italy, 2001, pp. 144–151.

Dong J., Zhao Y. and T. Peng. Architecture and Design Pattern Discovery Techniques – A Review. International Journal of Software Engineering and Knowledge Engineering, vol. 19, no. 6, pp. 823 – 856, Sep. 2009.

Ducasse S. y D. Pollet. Software Architecture Reconstruction: A Process-Oriented Taxonomy. IEEE Transactions on Software Engineering, vol. 35, n.º 4, pp. 573 - 591, ago. 2009.

El Boussaidi G., Belle A., Vaucher S. and H. Mili. Reconstructing Architectural Views from Legacy Systems, 19th Working Conference on Reverse Engineering, Kingston, ON, 2012, pp. 345 – 354.

Favre L., Martinez L and C. Pereira. MDA-Based Reverse Engineering of Object Oriented Code. Enterprise, Business-Process and Information Systems Modeling. vol. 29, Springer, 2009, pp. 251–263.

Fedesoft. Sector de tecnologías quiere superar el 2011. Disponible en http://www.fedesoft.org/novedades/sector-tecnologias-quiere-superar-el-2011 Visitado 24 de noviembre de 2012.

García J., Popescu, Mattmann D., Medvedovic N. and C. Yuanfang. Enhancing Architectural Recovery Using Concerns, Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference, Lawrence, KS, 2011, pp. 552 – 555.

Goeminne M. y T. Mens, A Framework for Analysing and Visualising Open Source Software Ecosystems, IWPSE-EVOL ’10, Antwerp, Belgium, 2010, pp. 42 - 47.

Grammatech Inc. The CodeSurfer slicing system, 2002.

Grant W, Pedersen E. y S. Marín. Ecología y manejo de recursos naturales: análisis de sistemas y simulación. Editorial Agroamérica. San José de Costa Rica. 340 p. 2001.

Grati H., Sahraoui H. and P. Poulin. Extracting Sequence Diagrams from Execution Traces using Interactive Visualization. 17th Working Conference on Reverse Engineering, Beberly, Massachusets USA, 2010, pp. 87–96.

Guéhéneuc Y.G., Mens K. and R. Wuyts. A Comparative Framework for Design Recovery Tools. Conference on Software Maintenance and Reengineering, Bari, Italy, 2006, pp. 123–134.

Gupta M. Design pattern mining using greedy algorithm for multi-labelled graphs. International Journal of Information and Communication Technology, vol. 3, no. 4, pp. 314–323, Nov. 2011.

Haslinger E., Lopez R. and A. Egyed. Reverse Engineering Feature Models From Programs' Feature Sets, 18th Working Conference on Reverse Engineering, pp. 308 – 312, 2011.

Hassan A. E. and R. C. Holt. Using Development History Sticky Notes to Understand Software Architecture,12th IEEE International Workshop on Program Comprehension, Bari, Italy, 2004, pp. 183 – 192.

Heesch U., Avgeriou P., Zdun U. y N. Harrison. The supportive effect of patterns in architecture decision recovery— A controlled experiment. Science of Computer Programming, 77(5), 2012. pp. 551–576.

Huang G., Mei H. y F. Yang. Runtime software architecture based on reflective middleware. Science in China Series F: Information Sciences, vol. 47, no. 5, pp. 555–576, 2004.

Huang G, Mei H. y F. Yang. Runtime recovery and manipulation of software architecture of component-based systems. Automated Software Engineering, vol. 13, no. 2, pp. 257–281, Apr. 2006.

Ivkovic I. M. y Godfrey. Architecture recovery of dynamically linked applications: a case study. International Workshop on Program Comprehension, 2002, pp. 178 – 184.

Jacobson I, Booch G y J Rumbaugh. El proceso unificado de desarrollo de software. Addison Wesley. 2000.

Jouault F., Bézivin J. and M. Barbero. Towards an advanced Model Driven Engineering Toolbox. Innovations in Systems and Software Engineering, vol. 5, no. 1, pp. 5–12, mar. 2009.

Kitchenham B. DESMET: A method for evaluating Software Engineering methods and tools. Technical Report TR96-09. Department of Computer Science, University of Keele. August 1996

Kitchenham B., Pretorius R., Budgen D., Brereton P., Turner M., Niazi M. y S. Linkman. Systematic literature reviews in software engineering A tertiary study. Information and Software Technology, vol. 52, no. 8, pp. 792–805, 2010.

Koschke R.. Atomic Architectural Component Recovery for Program Understanding and Evolution. PhD thesis, University of Stuttgart, Germany, 2000.

Lanza M. y S. Ducasse. Polymetric Views—A Lightweight Visual Approach to Reverse Engineering, IEEE Transactions on Software Engineering, vol. 29, n.º 9, pp. 782 - 795, 2003.

Lee H., Youn H. and E. Lee. Automatic Detection of Design Pattern for Reverse Engineering. Fifth International Conference on Software Engineering Research, Management and Applications, Busan, South Korea, 2007, pp. 577–583.

Lehman, M.M. and L.A.Belady. Program Evolution- Processes of Software Change. Academic Press, London, 1985.

Lehman M. M., Ramil J F., Wernick, P D. Perry D E. and W. M. Turski. Metrics and Laws of Software Evolution - The Nineties View. 4th IEEE International Software Metrics Symposium, Albuquerque, NM, USA., 1997, pp. 20 – 32.

Li Z., Gittens M., Murtaza S. and N. Madhavji. A towards an extended relational algebra for software architecture. ACM SIGSOFT Software Engineering Notes, vol. 35, no. 3, pp. 1–4, May 2010.

Lungu M., Lanza M. and T. Gırba. Package Patterns for Visual Architecture Recovery, Conference on Software Maintenance and Reengineering (CSMR'06), Bari, 2006, pp. 196 – 205.

Lungu M., Lanza M. y Oscar Nierstrasz, Evolutionary and collaborative software architecture recovery with Softwarenaut. Science of Computer Programming, vol. 79, no. 1, pp. 204–223, 2013.

Maqbool O. and H. Babri. Hierarchical Clustering for Software Architecture Recovery, Software Engineering, IEEE Transactions on, vol. 33, no. 11, pp. 759 – 780, Nov-2007.

Martinez L., Pereira C. and L. Favre. Reverse Engineering Activity Diagrams from Object Oriented Code: An MDA-Based Approach. Computer Technology and Application, vol. 2, pp. 969-978, 2011.

Medvidovic N., Egyed A. and P. Gruenbacher. Stemming Architectural Erosion by Coupling Architectural Discovery and Recovery, International Conference on Software Engineering, Portland, Oregon, 2003, pp. 61 – 68.

Ministerio de Tecnologías de la Información y las Comunicaciones. Vive Digital Colombia. Marzo de 2011. Visitado mayo 11 de 2012. Disponible en: http://vivedigital.gov.co/files/Vivo_Vive_Digital.pdf

Monroy M., Arciniegas J. L. and J Rodríguez. Caracterización de herramientas de ingeniería inversa. Información Tecnológica. Vol. 23, num. 6, pp. 31-42. Dic. 2012.

Muhammad R. Why Teach Reverse Engineering?, ACM SIGSOFT Software Engineering Notes, vol. 30, n.º 4, pp. 1–4, jul. 2005.

Müller H. A., Jahnke J. H., Smith D. B., Storey M., Tilley S. R., and K. Wong. Reverse Engineering: A Roadmap. Conference on The Future of Software Engineering, Limerick, Ireland, 2000, pp. 47 – 60.

Nierstrasz O, Ducasse S. and S. Demeyer. Object-oriented reengineering patterns - an overview. 4th international conference on Generative Programming and Component Engineering, Tallinn, Estonia, 2005, pp. 1–9.

Nour A., Jacek R. y J. Buckley. Characterizing real-time reflexion-based architecture recovery: an in-vivo multi-case study. International ACM SIGSOFT conference on Quality of Software Architectures, Bertinoro, Italy, 2012, pp. 23–32.

Pereira C. T., Martínez L. I., and L. M. Favre. Recovering Use Case Diagrams from Object Oriented Code: an MDA-based Approach. International Journal of Software Engineering, IJSE Vol.5 No.2 July 2012

Perez R. MARBLE: Modernization approach for recovering business processes from legacy information systems. 28th IEEE International Conference on Software Maintenance, Trento, Italy, 2012, pp. 671–676.

Petrov. P. and U. Buy. A Systemic Methodology for Software Architecture Analysis and Design. Eighth International Conference on Information Technology: New Generations, Las Vegas, NV, 2011, pp. 196 – 200.

Pirzadeh H. and A. Hamou-Lhadj. A Software Behaviour Analysis Framework Based on the Human Perception Systems.33rd International Conference on Software Engineering, Honolulu, Hawaii, 2011, pp. 948–951.

Platenius M. C., von Detten M. and B. Steffen. Archimetrix: Improved Software Architecture Recovery in the Presence of Design Deficiencies. 16th European Conference on Software Maintenance and Reengineering, Genova, Italy, 2012, pp. 255 – 264.

Qingshan L. Dynamic Model Design Recovery and Architecture Abstraction of Object Oriented Software. Conference on Software Maintenance and Reengineering, 2005 pp. 284 – 287.

Qingshan L., Hua C., Shengming H., Ping C. y Z. Yun. Architecture Recovery and Abstraction from the Perspective of Processes. Working Conference on Reverse Engineering, Pittsburgh, PA, USA, 2005, pp. 57 – 66.

Qiu M., Jiang Q., Gao A., Chen E., Qiu D., and S. Chai. Detecting design pattern using subgraph discovery. The 2nd Asian Conference on Intelligent Information and Database Systems, Hue City, Vietnam, 2010, pp. 350–359.

Risi M., Scanniello G., y Genoveffa Tortora. Using fold-in and fold-out in the architecture recovery of software systems. Formal Aspects of Computing, 24(3), 2012, pp. 307–330.

Riva C. y J. V. Rodriguez. Combining static and dynamic views for architecture reconstruction. European Conference on Software Maintenance and Reengineering, Budapest, 2002, pp. 47 – 55.

Roy B. and T. C. Graham. An Iterative Framework for Software Architecture Recovery: An Experience Report, 2nd European conference on Software Architecture, Cyprus, 2008, pp. 210–224.

Rugaber S. and K. Stirewalt. Model-Driven Reverse Engineering. IEEE Software, vol. 21, no. 4, pp. 45–53, Jul. 2004.

Sartipi K., Kontogiannis K. and F. Mavaddat. Architectural Design Recovery using Data Mining Techniques, Fourth European Software Maintenance and Reengineering, Zurich, 2000, pp. 129 – 139.

Sartipi K. y K. Kontogiannis. A Graph Pattern Matching Approach to Software Architecture Recovery, Software Maintenance, Florence, 2001, pp. 408 – 419.

Sartipi K. y N. Dezhkam. An Amalgamated Dynamic and Static Architecture Reconstruction Framework to Control Component Interactions 259. Working Conference on reverse Engineering, Vancouver, BC, 2007, pp. 259 – 268.

Siff M. and T. W. Reps. Identifying modules via concept analysis. International Conference on Software Maintenance (ICSM), Oxford, England, UK, 1999, pp. 170–179.

Scanniello G., D'Amico A., D'Amico C. and T. D'Amico. An Approach for Architectural Layer Recovery. ACM Symposium on Applied Computing, Sierre, Switzerland, 2010, pp. 2198 – 2202.

Scanniello G., Risi M. and G. Tortora. Architecture Recovery using Latent Semantic Indexing and K-Means: an Empirical Evaluation, Software Engineering and Formal Methods, Pisa, Italy, 2010, pp. 103 – 112.

Schmerl B., Aldrich J., Garlan D., Kazman R. y H. Yan. Discovering Architectures from Running Systems. IEEE Transactions on Software Engineering, vol. 32, no. 7, pp. 454 – 466, Jul. 2006

Schmerl B., Garlan D. y H. Yan. Dynamically discovering architectures with DiscoTect. European Software Engineering Conference, 2005, pp. 103 – 106.

Schmidt F., MacDonell S. y Andy Connor. An Automatic Architecture Reconstruction and Refactoring Framework (Vol. 377, pp. 95 – 11). Presented at the International Conference on Software Engenieering Research, Management and Application, Baltimore, USA: Springer. 2012

Schrettner L., Hegedus P., Ferenc R., Fülöp L. J. and T. Bakota. Development of a methodology, software–suite and service for supporting software architecture

reconstruction, 14th European Conference on Software Maintenance and Reengineering, Madrid, Spain, 2010, pp. 190 – 193.

Shareef J. W. and R. K. Pandey. Dependency Analysis using UML for Component-Based Software Systems: An XMI Approach. International Journal of Computer Applications, vol. 47, no. 18, pp. 6 – 15, Jun. 2012.

SiliconWeek. La Industria del software en Europa no entiende la crisis. Disponible en http://www.siliconweek.es/noticias/ranking-empresas-software-europa-truffle-100-29192. Visitado 6 de noviembre de 2012.

Song J., Huang G., Chauvel F., Xiong Y., Hu Z., Sun Y. y H. Mei. Supporting runtime software architecture: A bidirectional-transformation-based approach. The Journal of Systems and Software, vol. 84, no. 5, pp. 711–723, 2011.

Störmer C. Software Quality Attribute Analysis by Architecture Reconstruction (SQUA3RE). 11th European Conference on Software Maintenance and Reengineering (CSMR'07), Amsterdam, Netherlands, 2007, pp. 361 – 364.

IEEE Computer Society. Guide to the Software Engineering Body of Knowledge (SWEBOK). 2004.

Terra R., Valente M., Czarnecki K. y Roberto S. Bigonha. Recommending Refactorings to Reverse Software Architecture Erosion. European Conference on Software Maintenance and Reengineering. 2012.

Tonella P., Torchiano M., Du Bois B, and T. Systä. Empirical studies in reverse engineering: state of the art and future trends. Empirical Software Engineering, vol. 12, no. 5, pp. 551 – 571, Oct-2007.

Treude C., Figueira F. and M. Storey. An Exploratory Study of Software Reverse Engineering in a Security Context. 18th Working Conference on Reverse Engineering, Limerick, Ireland, 2011, pp. 184 – 188.

Vasconcelos A. and C. Werner. Evaluating reuse and program understanding in ArchMine architecture recovery approach. Journal Information Sciences: an International Journal, vol. 181, no. 13, pp. 2761–2786, Jul. 2011.

Van den Brand M., Protic Z. and T. Verhoeff. Generic tool for visualization of model differences. IWMCP '10, Malaga, Spain, 2010, pp. 66 – 75.

Van Geet J., Ebraert P. and S. Demeyer. Redocumentation of a legacy banking system: an experience report. International Workshop on Principles of Software Evolution, Antwerp, Belgium, 2010, pp. 33–41.

Vasconcelos A. y C. Werner. Evaluating reuse and program understanding in ArchMine architecture recovery approach. Journal Information Sciences: an International Journal, vol. 181, no. 13, pp. 2761–2786, Jul. 2011.

von Detten M, Archimetrix: A Tool for Deficiency-Aware Software Architecture Reconstruction. Working Conference on Reverse Engineering, 2012.

Wang F., Ke H. and J. Liu. Towards the Reverse Engineer of UML2.0 Sequence Diagram for Procedure Blueprint. World Congress on Software Engineering, Xiamen, China, 2009, pp. 118–122.

Wang Y., Liu P., Guo H., Li H. and X. Chen. Improved Hierarchical Clustering Algorithm for Software Architecture Recovery, International Conference on Intelligent Computing and Cognitive Informatics, Kuala Lumpur, 2010, pp. 247 – 250.

Weijun S., Shixian L. and L. Xianming. An Approach for Reverse Engineering of Web Applications. International Symposium on Information Science and Engineering, Shanghai, China, 2008, pp. 98–102.

Weiser M. Program slicing. IEEE Transactions on Software Engineering, vol. 10, no. 4, pp. 352–357, July 1984.

Yan H, Garlan D., Schmerl B. y J. Aldrich. DiscoTect: A System for Discovering Architectures from Running Systems. International Conference on Software Engineering, 2004, pp. 470–479.

Yang Y. y C. Riva. Scenarios for Mining the Software Architecture Evolution, International Workshop on Mining Software Repositories, Shanghai, China, 2006, pp. 10 – 13.

Zhao C. and K. Zhang. A Grammar-Based Reverse Engineering Framework for Behavior Verification. 11th IEEE High Assurance Systems Engineering Symposium, Nanjing, China, 2008, pp. 449–452.

Ziadi T., Almeida M., Messan L. and M. Ziane. A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams. 2011 16th IEEE International Conference on Engineering of Complex Computer Systems, Las Vegas, USA, 2011, pp. 107–116.

9. Acta de propiedad intelectual

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
ACTA DE ACUERDO SOBRE LA PROPIEDAD INTELECTUAL DE LA TESIS DE MAESTRIA Y DOCTORADO

En atención al acuerdo del Honorable Consejo Superior de la Universidad del Cauca, número 008 del 23 de Febrero de 1999, donde se estipula todo lo concerniente a la producción intelectual en la institución, los abajo firmantes, reunidos el día ___ del mes de _____ de _____ en el salón del Consejo de Facultad, acordamos las siguientes condiciones para el desarrollo y posible usufructo del siguiente proyecto. Materia del acuerdo: Tesis de Doctorado para optar el título de Doctor en Telemática, Área servicios avanzados de telecomunicaciones.

Título de la Tesis: Marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento.

Objetivo de la Tesis: Definir un marco de referencia para la recuperación y análisis de vistas arquitectónicas de comportamiento de sistemas software mediante la integración de técnicas de recuperación de arquitectura y la elaboración de un mecanismo de consulta

Duración de la Tesis: 17 meses.
Cronograma de actividades: Anexo I
Término de vinculación de cada partícipe en el mismo: No aplica
Organismo financiador: Universidad de Cartagena, naturaleza y cuantía de sus aportes $ 58.728.250,00, porcentaje de los costos del trabajo 35.36%

Los participantes de la Tesis, el señor estudiante de Doctorado **Martín Emilio Monroy Ríos** identificado con la cédula de ciudadanía número **80.411.302** de Bogotá, a quien en adelante se le llamará "estudiante", el ingeniero en calidad de Director del trabajo de grado, **Doctor José Luis Arciniegas Herrera** identificado con la cédula de ciudadanía **76'319.265** de Popayán (Cauca),a quien en adelante se le llamará "docente", y la Universidad del Cauca, representada por el Decano de la FIET, manifiestan que:

1.- La idea original del proyecto es de **José Luis Arciniegas PhD y Martín Monroy MSc.**, quienes la propuso y presentó al Grupo de investigación respectivo **Grupo de Ingeniería Telemática**, que la aceptó como tema para el proyecto de grado en referencia.
2.- La idea mencionada fue acogida por el estudiante como proyecto para obtener el grado de Doctor en Ingeniería Telemática, quien la desarrollará bajo la dirección del docente.

3.- Los derechos intelectuales y morales corresponden al docente y a los estudiantes.

4.- Los derechos patrimoniales corresponden al docente, a los estudiantes y a la Universidad del Cauca por partes iguales y continuarán vigentes, aún después de la desvinculación de alguna de las partes de la Universidad.

5.- Los participantes se comprometen a cumplir con todas las condiciones de tiempo, recursos, infraestructura, dirección, asesoría, establecidas en el anteproyecto, a estudiar, analizar, documentar y hacer acta de cambios aprobados por el Consejo de Facultad, durante el desarrollo del proyecto, los cuales entran a formar parte de las condiciones generales.

6.- El estudiante se compromete a restituir en efectivo y de manera inmediata a la Universidad los aportes recibidos y los pagos hechos por la Institución a terceros por servicios o equipos, si el comité de Postgrados, previo concepto del Comité de Maestría/Doctorado respectivo declara suspendido el proyecto por incumplimiento del cronograma o de las demás obligaciones contraídas por los estudiantes; y en cualquier caso de suspensión, la obligación de devolver en el estado en que les fueron proporcionados y de manera inmediata, los equipos de laboratorio, de cómputo y demás bienes suministrados por la Universidad para la realización del proyecto.

7.- El docente y los estudiantes se comprometen a dar crédito a la Universidad y de hacer mención del Fondo de Fomento de Investigación en caso de existir, en los informes de avance y de resultados, y en registro de éstos, cuando ha habido financiación de la Universidad o del Fondo.

8.- Cuando por razones de incumplimiento, legalmente comprobadas, de las condiciones de desarrollo planteadas en el anteproyecto y sus modificaciones, el participante deba ser excluido del proyecto, los derechos aquí establecidos concluyen para él. Además se tendrán en cuenta los principios establecidos en el reglamento del programa y el acuerdo 035 de 1992 vigente de la Universidad del Cauca en lo concerniente a la cancelación y la pérdida del derecho a continuar estudios.

9.- El documento del anteproyecto y las actas de modificaciones si las hubiere, forman parte integral de la presente acta.

10.- Los aspectos no contemplados en la presente acta serán definidos en los términos del acuerdo 008 del 23 de febrero de 1999 expedido por el Consejo Superior de la Universidad del Cauca, del cual los participantes del acuerdo aseguran tener pleno conocimiento.

Director ----------

Estudiante----------

Decano Facultad----------