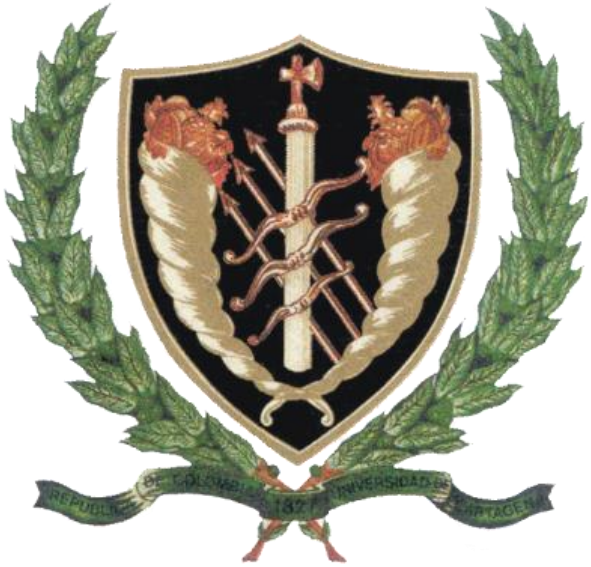
EVALUACIÓN DE HERRAMIENTAS DE INGENIERÍA INVERSA DE CÓDIGO ABIERTO

INVESTIGADOR

Jonathan Lozano Morelo



UNIVERSIDAD DE CARTAGENA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

CARTAGENA DE INDIAS, 2017

EVALUACIÓN DE HERRAMIENTAS DE INGENIERÍA INVERSA DE CÓDIGO ABIERTO

TÉSIS DE GRADO

E-Soluciones

Ingeniería De Software

INVESTIGADOR

Jonathan Lozano Morelo

Director: Martín Monroy Ríos, Msc, PhD (Universidad de Cartagena)

UNIVERSIDAD DE CARTAGENA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

CARTAGENA DE INDIAS, 2017

UNIVERSIDAD DE CARTAGENA

Tesis de Grado: EVALUACIÓN DE LAS HERRAMIENTAS DE INGENIERÍA INVERSA DE CÓDIGO ABIERTO

Autor: JONATHAN LOZANO MORELO

Director: Ing. MARTÍN MONROY RÍOS, MSc., PhD.

Nota de Aceptación

_____

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Cartagena de Indias, ____ de _____ de 2017

Contenido

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

RESUMEN

Se requiere un profundo conocimiento en arquitectura y detalles de implementación para mantener sistemas de software de manera exitosa (Redocumentation of a Legacy Banking System, 2010), habitualmente este conocimiento es registrado en forma explícita en la documentación o de manera implícita en la mente de los expertos que lo desarrollan, pero desafortunadamente, con frecuencia no se dispone de dicho conocimiento, por lo que es necesario reconstruirlos a través de un proceso de ingeniería inversa. Sin embargo, las capacidades de las herramientas existentes no satisfacen las necesidades de ingeniería inversa existentes y tienen que ser mejoradas.

Para esto es necesario identificar las limitaciones a nivel de funcionalidad que tienen las herramientas de ingeniería inversa de código abierto existentes, razón por la cual este trabajo tiene como fin evaluar las herramientas de ingeniería inversa de código abierto para conocer las limitaciones de las funcionalidades que ofrecen, aplicando la metodología DESMET.

Teniendo en cuenta esto se identificaron métodos de recuperación documental necesarios para lograr una correcta obtención de las herramientas a evaluar, realizando un comparativo de las funcionalidades encontradas en la documentación recuperada y las pruebas realizadas a cada herramienta, definiendo de esta manera los artefactos de diseño y arquitectura que estas herramientas recuperan, las funcionalidades que estas ofrecen y dejando en claro aquellas funcionalidades que no han sido implementadas en las herramientas.

Sin embargo, se concluye que las herramientas de ingeniería inversa están enfocadas en responder a una necesidad específica, por lo que estas no alcanzan a abarcar todas las funcionalidades que se establecieron como base de este estudio, pero logrando responder a la necesidad por la cual fueron creadas.

ABSTRACT

A deep knowledge of architecture and implementation details is required to maintain software systems successfully (Redocumentation of a Legacy Banking System, 2010). This knowledge is usually explicitly recorded in documentation or implicitly in the minds of experts who develop it, but unfortunately, this knowledge is often not available, so it is necessary to rebuild them through a reverse engineering process. However, the capabilities of existing tools do not meet existing reverse engineering needs and need to be improved.

Because of this, it is necessary to identify the limitations in functionality that actually have the open source reverse engineering tools, reason why this work purpose is to evaluate the open source reverse engineering tools to know the limitations of the functionalities that they offer, whereby the DESMET methodology was applied.

Considering this, were identified methods of documentary recovery necessary to achieve a correct obtainment of the tools to be evaluated and made a comparison of the functionalities found in the recovered documentation and the tests performed to each tool, defining this way the design artifacts and architecture that these tools recover, the functionalities that offer and clarifying those functionalities that have not been implemented in the tools.

However, the reverse engineering tools are focused on responding to a specific need, so they are not enough to cover all the functionalities that were established as the basis of this study, but managing to respond to the need which they were created.

1. INTRODUCCIÓN

En este capítulo se encuentra el problema por el cual este estudio fue planteado, manifestando la importancia que tiene, señalando como se verificó la documentación actual y ampliando la literatura existente sobre las herramientas de ingeniería inversa.

1.1. FORMULACIÓN DEL PROBLEMA

Actualmente la mayoría de los procesos organizacionales e industriales se soportan en sistemas software (Redocumentation of a Legacy Banking System, 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 (Object-oriented reengineering patterns - an overview, 2005).

Para reconstruir éste conocimiento se aplica la ingeniería inversa, entendida como *"el proceso de análisis de un sistema sujeto a identificar los componentes del sistema y sus interrelaciones, y crear representaciones del sistema en otra forma o en un nivel más alto de abstracción"* (Reverse Engineering and Design Recovery: A Taxonomy, 1990). Aunque las dos últimas décadas han representado grandes avances para la ingeniería inversa (Canfora et al, 2007; Tonella et al, 2007), aún existen desafíos que le impiden dar solución de manera completa a las situaciones expuestas. No obstante, 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), pero es muy limitada la capacidad para consultar la información base obtenida en los artefactos (piezas de información) generados por los analizadores (Canfora et al, 2011).

Ante esta situación, el ingeniero Martín Monroy Ríos, MSc, docente investigador de la Universidad de Cartagena, establece como objetivo de su tesis doctoral en curso "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". Para lograr dicho objetivo propone dentro de las actividades a realizar la construcción de una herramienta de ingeniería inversa de código abierto.

Con el fin de que dicha herramienta represente un aporte al estado del arte, se hace necesario identificar las limitaciones a nivel de funcionalidad tienen las herramientas de ingeniería inversa de código abierto existentes en la actualidad, razón por la cual se propone realizar una evaluación técnica de las herramientas de ingeniería inversa para identificar qué aspectos de la documentación actualmente recuperan y verificar cuál es el nivel de calidad de las funcionalidades que ofrecen. Para ello se realizó un estudio detallado de las herramientas de ingeniería inversa existentes, centrando la atención en aquellas que son de código abierto, verificando las funcionalidades que brindan.

1.2. JUSTIFICACIÓN

Las herramientas de ingeniería inversa de código abierto existentes poseen muchas diferencias y por lo tanto proveen de buenas capacidades de ingeniería inversa según el contexto de uso, todas tienen sus fortalezas y debilidades, por lo que resulta difícil declarar que una herramienta sea mejor que otra (Bellay et al, 1997). Aun así, resulta necesario identificar las funcionalidades que están siendo provistas por las herramientas de ingeniería inversa ya que al ser útiles en aspectos diferentes del desarrollo y el mantenimiento de software resultaría desventajoso aplicarlas en contextos en el que se abarquen una gran cantidad de situaciones diferentes y la solución que estas brinden serían incompletas.

Uno de los beneficios primordiales de la investigación propuesta es el aporte académico que puede brindar, ya que uno de los principales objetivos que se quiere alcanzar con esta investigación es promover la investigación y el desarrollo de herramientas que puedan suplir las deficiencias que actualmente poseen las herramientas de ingeniería inversa de

código abierto existentes, lo que hará que estas puedan ser aplicadas en contextos un poco más generales y gracias a los conocimientos adquiridos a lo largo de la formación académica del autor en el área de la ingeniería de sistemas, este dispone de las capacidades necesarias que hacen posible la realización de este proyecto, enriqueciendo y ampliando además, con conocimientos nuevos que lo ayudaron a crear un pensamiento más crítico en el área de la ingeniería de software y la ingeniería inversa.

El resultado del estudio sirve a aquellas personas que se dedican al mantenimiento de software, ya que en este proceso la comprensión del software es un procedimiento intensivo en el que se deben adquirir suficientes conocimientos acerca de los artefactos del software o del sistema entero, lo cual requiere un gran esfuerzo debido a que éste demanda una gran capacidad de análisis, lo que en algunos casos podrá verse desafiado por la falta de documentación adecuada y actualizada del software (Canfora et al, 2011). Por ello las personas que se dedican al mantenimiento de software tienden a utilizar las herramientas de ingeniería inversa para facilitar el proceso de comprensión de software, pero los requerimientos y expectativas de las personas están en un continuo cambio y crecimiento, y las herramientas de ingeniería inversa no son la excepción, por lo que este estudio es favorable porque en él se encontraron las características que permiten un cambio en las herramientas de ingeniería inversa actuales y se respondió a la pregunta de investigación: ¿Qué funcionalidades no brindan actualmente las herramientas de ingeniería inversa de código abierto?

Además, la temática de la ingeniería inversa actualmente es de gran importancia ya que permite enmendar la deficiencia existente al documentar productos software a través de código ejecutable, lo cual representa muchas ventajas a nivel de disminución de costos y tiempos de mantenimiento, lo que resulta provechoso debido a que en el ciclo de vida de software el mantenimiento y el soporte representan la mayor parte del costo, y cuya proporción es mayor al 50% del costo del software (Canfora et al, 2011).

Este estudio se realizó en la Universidad de Cartagena haciendo uso de las bases de datos digitales disponibles ofrecidas por esta (IEEExplore, ACM Digital Library, SpringerLink,

Citeseer, Wiley online Library), realizando una búsqueda sistemática para encontrar la mayor cantidad de información de las herramientas que fueron objeto de este estudio y verificando esta documentación a través de la instalación y pruebas realizadas, evidenciando las funcionalidades que estas herramientas no brindan, generando de esta manera un valor agregado a la literatura existente sobre las herramientas de ingeniería inversa de código abierto, como se demuestra en el capítulo de resultados.

1.3. OBJETIVO GENERAL

Evaluar las herramientas de ingeniería inversa de código abierto para conocer las limitaciones de las funcionalidades que ofrecen, aplicando la metodología DESMET.

1.4. OBJETIVOS ESPECÍFICOS

- Identificar las herramientas de ingeniería inversa disponibles en la actualidad para seleccionar las de código abierto que serán objeto del estudio a través de un análisis de características.
- Definir los criterios de valoración que se utilizarán para evaluar las herramientas de ingeniería inversa de código abierto
- Diseñar los escenarios de valoración de las herramientas de ingeniería inversa teniendo en cuenta los criterios definidos.
- Evaluar las herramientas de ingeniería inversa seleccionadas aplicando los escenarios de valoración diseñados a cada una de ellas.

1.5. ALCANCE

El proyecto se limita a evaluar sólo aquellas herramientas de ingeniería inversa que sean de código abierto y que permitan iniciar al proceso de ingeniería inversa desde el código fuente, además, las limitaciones de las funcionalidades que ofrecen las herramientas de ingeniería inversa de código abierto fueron evaluadas con respecto a la documentación que recuperan (artefactos de diseño y de arquitectura), ya sea con diagramas UML y sus respectivas características o con otros tipos de representaciones.

2. ESTADO DEL ARTE

2.1. INGENIERÍA INVERSA

El estándar IEEE-1219 recomienda la ingeniería inversa como una tecnología clave de soporte para manejar aquellos sistemas que tengan el código fuente como único fundamento de confianza para su representación, por lo que ha sido empleado para lidiar con una gran cantidad de problemas de ingeniería de software, como la recuperación de arquitectura, diseño de patrones, re-documentación de programas y bases de datos, renovación de interfaces de usuario, cambio de plataformas de los sistemas existentes, entre otras actividades frecuentes del mantenimiento de software. Para cubrir el amplio rango de sistemas existentes se han creado una gran cantidad de herramientas, por lo que ya cuentan con una gran cantidad de habilidades tales como la capacidad de analizar diferentes lenguajes y código no compilable, la capacidad de compilar y ejecutar programas, la capacidad de realizar un análisis de caja negra, la capacidad de clasificar y combinar información de diferentes repositorios de software, visualización apropiada de información abstracta, por mencionar algunas (Achievements and challenges in software reverse engineering, 2011).

Las herramientas de ingeniería inversa usualmente proporcionan un grupo de funciones para que el ingeniero de software pueda trabajar con respecto a diferentes puntos de vista del software, dependiendo de la tarea que desea realizar o del punto de vista específico desde el cual se desea comprender el sistema, esto pensando en la gran variedad de sistemas que pueden existir y en la cantidad de perspectivas que se pueden tener con respecto al mismo. Mientras que la variedad de actividades de la ingeniería inversa produce que se dificulte comparar las herramientas, también representa un reto para los desarrolladores de herramientas, ya que resulta difícil construir una herramienta de ingeniería inversa que sea de uso general, cuando no sólo los lenguajes de programación, sino también las operaciones que han de aplicarse para analizar el software difieren. En algunas herramientas este problema ha intentado ser resuelto, haciendo de estas más adaptables y extensibles de tal

manera que puedan acoplarse a diferentes necesidades o perspectivas. La posibilidad de personalizar y ampliar una herramienta afecta claramente a su facilidad de uso y adaptabilidad (Empirical studies in reverse engineering: state of the art and future trends, 2007).

La ingeniería inversa de software ha alcanzado un alto grado de madurez, sin embargo, existen una gran cantidad de problemas que deben ser estudiados para poder mejorar los diferentes métodos y herramientas existentes, sobre si se considera la cantidad de escenarios diferentes que puedan surgir en el futuro en el área de la ingeniería de software y más específicamente en el mantenimiento, y en los que la ingeniería inversa pueda ocupar un rol importante (Achievements and challenges in software reverse engineering, 2011).

En los últimos años, la disponibilidad de múltiples fuentes de datos ha exigido técnicas para combinarlas y la necesidad de filtrar datos en una enorme cantidad de información, que de otra manera causarían una sobrecarga para los desarrolladores. Los sistemas de recomendación son una respuesta emergente a este tipo de temas favorecidos por la disponibilidad de los entornos de desarrollo altamente personalizables, tales como Eclipse[1] y NetBeans[2], proporcionando formas de desarrollar rápidamente nuevas herramientas completamente integradas en el entorno que el desarrollador esté utilizando (Achievements and challenges in software reverse engineering, 2011).

En la actualidad hay muchos sistemas multilenguaje y multiplataforma, por lo que las herramientas de ingeniería inversa deben ser capaces de manejar múltiples lenguajes y plataformas dentro de un único marco conceptual (Achievements and challenges in software reverse engineering, 2011). Además, continúan emergiendo una gran cantidad de tipos de artefactos de software, lo que demuestra que los sistemas están cambiando frecuentemente con el tiempo, por lo que las herramientas de ingeniería inversa deben adaptarse a estos cambios continuamente.

[1] http://www.eclipse.org/
[2] https://netbeans.org/

2.2. EVALUACIÓN DE HERRAMIENTAS DE INGENIERÍA INVERSA

El continuo proceso de adaptación de las herramientas de ingeniería inversa a los permanentes cambios y problemas que surgen ha dado paso a numerosos estudios de estas mismas, de las necesidades que representan estos cambios y de cómo adaptar estas herramientas con respecto a esos cambios. Hay una gran cantidad de campos diferentes en los cuales se han hecho investigaciones, experimentos, comparaciones, entre otros métodos de evaluación; algunas de las evaluaciones que se han realizado para el mejoramiento de las herramientas de ingeniería inversa demuestran la importancia y la necesidad que existe de adaptar las herramientas de ingeniería inversa a las nuevas tecnologías y las necesidades actuales, algunos casos son los siguientes:

En 1996 fue ejecutado un estudio piloto realizado en la Universidad de *Victoria and Simon Fraser University* para evaluar interfaces alternativas sobre la herramienta *Rigi* y mejorar su rendimiento (On Designing an Experiment to Evaluate a Reverse Engineering Tool, 1996), se pudo hacer comparación de las funcionalidades de estas interfaces de usuario y cómo interactuaban con la herramienta, viendo de esta manera las ventajas y desventajas que representa la implementación de diferentes tecnologías bajo un mismo contexto, además de que se pueden evidenciar las ventajas que pueden conllevar la realización de evaluaciones en el proceso de mejoramiento del rendimiento de las herramientas de las herramientas ingeniería inversa con la adaptación de estas con las nuevas tecnologías.

En una conferencia realizada en Ámsterdam (A Comparison of four Reverse Engineering Tools, 1997) fue presentada una comparación realizada sobre 4 herramientas de ingeniería inversa mostrando los beneficios y deficiencias de las herramientas en términos de aplicabilidad para software integrado, usabilidad y extensibilidad, y el cual fue enfocado principalmente en su capacidad de crear reportes gráficos tales como árboles de llamadas, datos y gráficos de control de flujo. En este estudio se explica que el estado de una evaluación tecnológica depende de cómo la tecnología evaluada difiere de otras tecnologías y cómo estas diferencias atienden las necesidades de los contextos de uso específicos,

mostrando de esta manera otra ventaja que pueden ser obtenidas con la realización de evaluaciones sobre este tipo de herramientas.

En la WCRE (Working Conference on Reverse Engineering) realizada en Stuttgart, Alemania, fue presentada una demostración estructurada creativa de herramientas de ingeniería inversa, la cual consiste en una técnica de evaluación de herramientas hibrida que combina elementos de experimentos, casos de estudio, demostraciones tecnológicas y evaluación comparativa, que buscaba facilitar el aprendizaje sobre herramientas de ingeniería de software usando un grupo de tareas comunes. En esta experiencia se discutió sobre tecnologías complementarias que podrían ser aplicadas para la solución de problemas de ingeniería inversa de la vida real, en la que aplicaron sus propias herramientas para este problema, concluyendo que era necesario aprender mucho más sobre sus herramientas y como pueden ser aplicadas en un proceso de ingeniería inversa o reingeniería (A Collaborative Demonstration of Reverse Engineering tools, 2001).

En el 2003 fue realizada una comparación entre cinco herramientas de visualización dinámica, las cuales consisten en modelar el comportamiento de los sistemas de software, por lo que tienen una gran variedad de usos en las herramientas de ingeniería inversa y el proceso de comprensión de software. En el estudio fueron identificadas tres áreas de investigación para el análisis dinámico: formas efectivas de presentar los resultados, integración con la ingeniería avanzada, y la aplicación de la integración y la migración, concluyendo que las herramientas no eran lo suficientemente poderosas, y requieren combinar información estadística y dinámica para tener un buen desempeño en todas las tareas, demostrando claramente que ninguna herramienta de visualización dinámica puede responder todas las preguntas que son típicas en la comprensión de software o de los esfuerzos de ingeniería inversa. Algunas tareas eran peor manejadas que otras y algunas capacidades estaban más allá de las capacidades de las herramientas (A Comparative Evaluation of Dynamic Visualisation Tools, 2003).

En Cartagena de indias, Colombia, fue realizado un trabajo de caracterización de herramientas de ingeniería inversa (Caracterización de Herramientas de Ingeniería Inversa,

2012) en el cual se definen los criterios de evaluación de este tipo de herramientas, para facilitar su análisis y entendimiento, y permitir la valoración de sus componentes. Además, facilita el establecimiento de comparaciones de herramientas y apoya la toma de decisiones al decidir cuál es la más apropiada, lo que ayudará a definir aquellos aspectos de estas herramientas que necesitan ser mejorados.

2.3. METODOLOGÍAS DE EVALUACIÓN DE HERRAMIENTAS DE SOFTWARE

En el proceso de evaluación de software es muy común cometer errores debido a la mala práctica de las técnicas de evaluación de la industria del software, las metodologías de evaluación de software han mejorado a través del tiempo, creándose metodologías cada vez más robustas que podrían abarcar una gran cantidad de técnicas de apoyo de decisiones y que puedan adaptarse a muchos contextos. Las metodologías de evaluación pueden ser tanto cuantitativas como cualitativas; las evaluaciones cuantitativas se basan en la asunción de que se pueden identificar propiedades medibles del producto software que se esperan cambiar como resultado de usar una metodología y las evaluaciones cualitativas se basan en la identificación de necesidades que tiene un usuario para realizar una tarea/actividad en particular y la asignación de requisitos a características que una método o instrumento debe poseer para apoyar esa tarea/actividad (DESMET: A method for evaluating Software Engineering methods and tools, 1997).

Las metodologías de evaluación cuantitativa pueden ser casos de estudios, experimentos formales y encuestas; los casos de estudio involucran que la tarea o actividad haya sido utilizada en un proyecto de software de la vida real, esto asegura aplicabilidad en casos concretos; las encuestas pueden ser utilizadas cuando varios métodos o instrumentos han sido usados y se puede obtener información de la experiencia que se obtuvo de su utilización a través de preguntas.

Las metodologías de evaluación cualitativa pueden ser: experimentos cualitativos en los que se está enfocado al análisis de las características pero organizando el proceso de evaluación como un experimento; los casos de estudio cualitativos que se refiere a la evaluación de las características a través del uso de los métodos/instrumentos en el mundo

real; y por último, las encuestas cualitativas que están basadas en la recolección de información de los métodos/instrumentos luego de que han sido utilizados en la práctica o en un proyecto de la vida real.

Además, se encuentran las metodologías de evaluación híbridas, que combinan elementos de las metodologías cualitativas y cuantitativas, entre los que se encuentran las evaluaciones comparativas y el análisis a partir de los efectos cuantitativos. Además, existe una metodología de evaluación híbrida llamada DESMET, esta separa los ejercicios de evaluación en cuantitativos y cualitativos, contando con un criterio de identificación que ayuda al evaluador a escoger el método más apropiado en una circunstancia específica, la metodología DESMET identifica criterios que pueden afectar la selección del método de evaluación (DESMET: A method for evaluating Software Engineering methods and tools, 1997) lo cual reduce el riesgo de que la evaluación resulte inválida o incorrecta. Los niveles de evaluación fueron identificados en 3 niveles sucesivos: elemental, el cual se refiere a la descripción completa, entendible, utilizable, consistente, etc. de los componentes; el nivel de uso, el cual se refiere al grado de utilidad de los componentes a través de aspectos como cuando es utilizado, si alcanza su objetivo, produce los resultados esperados, produce resultados utilizables y relevantes, se comporta como se espera y si requiere de la asistencia de expertos; por último está el nivel de ganancias, en el que se evalúa si el producto es mejor que anteriores, a través de aspectos como el valor agregado, el apoyo a alguna hipótesis o satisfacción de criterios específicos de ganancias. Además, identifica actividades de evaluación como la revisión, casos de estudios simulados y casos de estudio.

3. MARCO TEÓRICO

3.1. INGENIERÍA DE SOFTWARE:

3.1.1. CONCEPTUALIZACIÓN:

Según la IEEE la ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de ingeniería al software.

La Ingeniería de Software es una disciplina o área de la Informática o Ciencias de la Computación, que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo. Hoy día es cada vez más frecuente la consideración de la Ingeniería de Software como una nueva área de la ingeniería, y el ingeniero de software comienza a ser una profesión implantada en el mundo laboral internacional, con derechos, deberes y responsabilidades que cumplir, junto a una, ya, reconocida consideración social en el mundo empresarial y, por suerte, para esas personas con brillante futuro (Pressman, 2002). Esta disciplina aborda todas las fases del ciclo de vida del desarrollo de cualquier tipo de sistemas de información, aplicables a una infinidad de áreas.

El ciclo de vida del desarrollo de software debe contar con varias actividades como planificación, desarrollo, pruebas, documentación, despliegue y mantenimiento, para lo cual se han desarrollado varios modelos que perfilan el proceso de desarrollo como el modelo de cascada, el modelo de espiral, el modelo RUP, entre otros, los cuales cuentan con muchos beneficios y debilidades lo que les hace apropiados para necesidades específicas.

3.1.2. PROCESO DE DESARROLLO DE SOFTWARE:

El estándar internacional ISO/IEC 12207 define 2 grandes subdivisiones del proceso, un contexto del sistema para lidiar con productos o servicios de software autónomos o sistemas de software y los procesos específicos de software para usar o implementar productos o servicios de software que hacen parte de un sistema mayor.

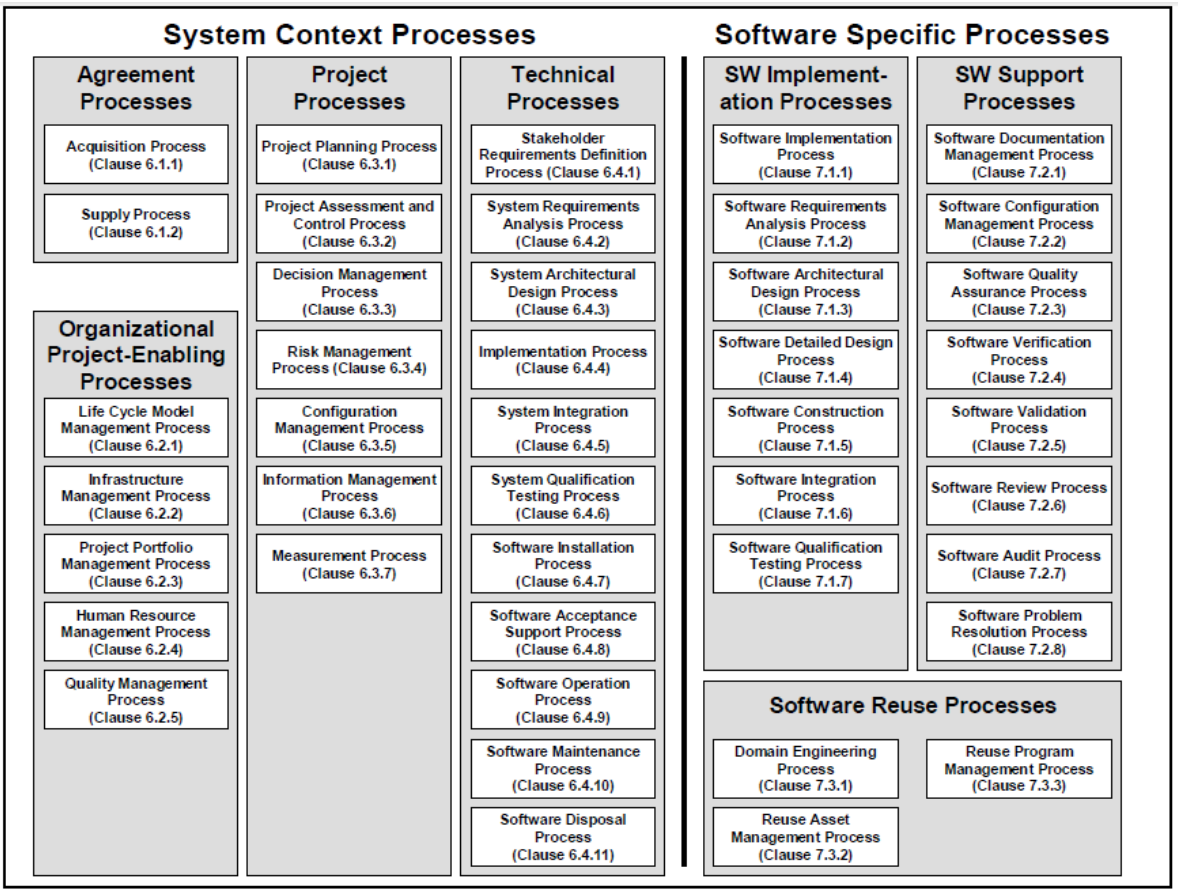


Ilustración 1 Procesos del ciclo de vida. Fuente: Estándar ISO/IEC 12207

A su vez, el proceso del contexto del sistema se divide en 4 subprocesos, según se observa en la Ilustración 1 y se explica a continuación**:**

- **Procesos de acuerdo:** estos procesos definen las actividades necesarias para establecer un acuerdo entre 2 organizaciones.
- **Procesos de habilitación de proyectos organizacionales:** gestionan las capacidades de la organización de adquirir y suministrar productos o servicios a través de la iniciación, soporte y control de proyectos. Proveen de los recursos e

infraestructura necesaria para apoyar proyectos y asegurar el cumplimiento de los objetivos organizacionales y los acuerdos establecidos. Este está conformado por: procesos del ciclo de vida modelo de gestión, proceso de infraestructura de gestión, proceso de gestión de proyectos, gestión de procesos de recursos humanos y proceso de gestión de la calidad.

- **Procesos del proyecto:** en este estándar, el proyecto ha sido escogido como el contexto para describir procesos referidos a la planeación, la evaluación y el control. Existen 2 categorías de procesos de proyecto: los procesos de soporte de proyectos que son utilizados para planear, ejecutar, evaluar y controlar el progreso de un proyecto, conformado por el proceso de planificación de proyectos, y el proceso de valoración y control de proyectos; y el proceso de soporte de proyectos que provee un conjunto de tareas específicamente enfocado en la realización de un objetivo de gestión especializado, y está compuesto por el proceso de gestión de decisiones, el proceso de gestión de riesgos, el proceso de gestión de la configuración, el proceso de gestión de la información y el proceso de medición.
- **Procesos técnicos:** son utilizados para definir los requerimientos para un sistema, transformar los requerimientos en un producto efectivo, permitir la reproducción consistente del producto si es necesario, usar el producto, proveer los servicios requeridos, mantener el suministro de esos servicios y disponer del producto cuando es retirado del servicio.

Y los procesos específicos de software se dividen en tres, según se observa en la Ilustración 1 y se explica a continuación:

- **Procesos de implementación de software:** son utilizados para producir elementos específicos del sistema implementados en software. Estos procesos transforman restricciones de comportamientos, interfaces e implementaciones específicos en acciones de implementación, resultando en un elemento del sistema que satisface los requerimientos derivados de los requerimientos del sistema. Estos procesos son: el análisis de requerimientos, diseño arquitectónico

de software, diseño detallado de software, construcción de software, integración de software y pruebas de calificación de software.

- ➢ **Procesos de reutilización de software:** este grupo consiste en el soporte de las habilidades organizacionales para reutilizar elementos de software a través de los límites del proyecto. Estos procesos son únicos debido a que, por naturaleza, operan fuera de los límites de cualquier proyecto en particular. Estos procesos son: la ingeniería de dominio, gestión de reutilización de activos y gestión de la reutilización de programas.
- ➢ **Procesos de modelo de referencia:** este modelo es aplicable para cualquier organización que valore sus procesos para determinar la capacidad de estos procesos. El propósito de los resultados son una declaración de los objetivos de rendimiento de cada proceso. Esta declaración de objetivos permite la valoración de la efectividad del proceso por medios diferentes a la simple evaluación de conformidad.

3.1.3. ARTEFACTOS DE SOFTWARE:

El manual de referencia UML define artefacto como "la especificación de una pieza física de información que se utiliza o se produce en un proceso de desarrollo de software, como un documento externo o un producto del trabajo, o mediante el desarrollo y manipulación de un sistema. Los artefactos pueden ser un modelo, una descripción o un software" (Rumbaugh, y otros, 2007). Los artefactos cuentan con la siguiente estructura:

- ➢ **Despliegue:** los artefactos pueden ser desplegado en un nodo, por lo que las instancias de un artefacto pueden ser desplegadas en las instancias de un nodo.
- ➢ **Artefactos anidados:** un artefacto puede contener otros artefactos, por lo que los artefactos anidados se desplegarán cuando se instancie el nodo contenedor.
- ➢ **Manifestación:** un artefacto puede ser originado de un conjunto de elementos del modelo e implementarlos.

3.1.4. *Licenciamiento de software:*

Una licencia es el contrato entre el desarrollador de un software sometido a propiedad intelectual y a derechos de autor y el usuario, en el cual se definen con precisión los derechos y deberes de ambas partes. Es el desarrollador, o aquél a quien éste haya cedido los derechos de explotación, quien elige la licencia según la cual distribuye el software.

- **Código fuente:** Se refiere al conjunto de líneas de texto escrito en un lenguaje de programación específico que especifican las instrucciones que debe seguir la computadora para ejecutar un programa, aunque las computadoras no realizan este proceso directamente, sino a través de compiladores o intérpretes que traducen estas líneas de código a un lenguaje comprensible por la máquina.

Los tipos de licencias de software de mayor importancia para el estudio con:

- **Software libre:** Según la Free Software Foundation, «Software libre» significa que el software respeta la libertad de los usuarios y la comunidad. En términos generales, los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el software. Con estas libertades, los usuarios (tanto individualmente como en forma colectiva) controlan el programa y lo que hace.

  Cuando los usuarios no controlan el programa, el programa controla a los usuarios. El programador controla el programa y, a través del programa, controla a los usuarios. Un programa que no es libre, llamado «privativo», es por lo tanto un instrumento de poder injusto.

  Un programa es software libre si los usuarios tienen las cuatro libertades esenciales (Free Software Foundation, 2013):

  - La libertad de ejecutar el programa para cualquier propósito (libertad 0).
  - La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
  - La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).

- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Un programa es software libre si los usuarios tienen todas esas libertades.

➢ **Código abierto:** Se refiere al software que puede ser libremente utilizado, modificado y compartido por cualquiera. Código abierto no significa solamente acceso al código fuente, los términos de distribución de los software de código abierto deben cumplir con los siguientes criterios (Open Source Initiative, 2013):

- **Distribución libre:** la licencia no debe restringir a un tercero el vender o entregar el software como parte de una distribución mayor que contenga programas de fuentes diferentes. La licencia no debe requerir regalía u otras comisiones por dicha venta.
- **Código fuente:** el programa debe incluir el código fuente y debe permitir tanto la distribución del código como de su forma compilada. Si el producto no es distribuido con el código fuente, debe haber un medio bien publicitado para obtener el código fuente por no más que un costo de reproducción razonable, descargándolo a través de internet sin costo alguno. El código fuente debe estar en la forma en la que un programador pueda modificarla mejor.
- **Trabajos derivados:** la licencia puede permitir modificaciones y trabajos derivados, y debe permitir que sean distribuidos bajo los mismos términos de licencia que el software original.
- **Integridad del código fuente del autor:** la licencia puede restringir que el código fuente sea distribuido en una forma modificada solo si la licencia permite la distribución de archivos de revisión con el código fuente con el fin de modificar el programa en tiempo de compilación. La licencia debe permitir explícitamente la distribución de software

integrado con código fuente modificado. Debe requerir que los trabajos derivados tengan un nombre diferente o un número de versión distinto al del software original.

- **No discriminar a ninguna persona o grupo:** la licencia no debe discriminar a ninguna persona o grupo de personas.
- **No discriminar ningún campo de trabajo:** la licencia no debe restringir a nadie de usar el programa en un campo de trabajo específico.
- **Distribución de la licencia:** los derechos de uso del programa deben aplicarse a todos aquellos a quienes se le redistribuya el programa, sin necesidad de pedir una licencia adicional para estos.
- **La licencia no debe ser específica de un producto:** los derechos de uso del programa no deben depender de que el programa sea parte de una distribución de software particular. Si el programa es extraído de esa distribución y utilizado o distribuido dentro de los términos de licencia del programa, cualquiera a quien sea redistribuido el programa debe tener los mismos derechos que son garantizados al formar parte de la distribución de software original.
- **La licencia no debe restringir otro software:** la licencia no debe colocar restricciones a los sistemas de software que son distribuidos con el software licenciado.
- **La licencia debe ser tecnológicamente neutral:** ninguna disposición de la licencia puede basarse en una tecnología individual o estilo de interfaz.

➢ **Estándar abierto:** según Bruce Perens es aquel basado en principios de disponibilidad de lectura e implementación, maximización de las opciones del usuario final, libertad de implementación, no discriminación al implementador, libertad de extensión y restricción, y prevención de prácticas predatorias por parte de fabricantes dominantes (Perens, 2013).

3.1.5. UML:

El lenguaje unificado de modelado o UML por sus siglas en inglés (Unified Modeling Language) es el proceso unificado define los componentes que se utilizarán para construir el sistema y las interfaces que conectarán los componentes. Utilizando una combinación del desarrollo incremental e iterativo, el proceso unificado define la función del sistema aplicando un enfoque basado en escenarios (desde el punto de vista de1 usuario). Entonces acopla la función con un marco de trabajo arquitectónico que identifica la forma que tomará el software (Pressman, 2002).

Para lograr la representación del sistema se requiere de 3 bloques de construcción básicos: los elementos, que son abstracciones en el modelo y se clasifican en estructurales, comportamiento, agrupación y anotación; las relaciones, que establecen el modo de interacción entre los elementos y pueden ser de dependencia, asociación, generalización y realización; y por último están los diagramas, que son representaciones gráficas del conjunto de elementos del modelo, UML 2.0 define 13 tipos de diagramas, divididos en 3 categorías: 6 diagramas para la representación de la estructura de la aplicación estática, 3 para las representaciones generales de comportamiento y 4 representando los diferentes aspectos de la interacción (OMG, 2013).

- ➢ Los diagramas estructurales incluyen el diagrama de clases, diagrama de objetos, diagrama de componentes, diagrama de estructura compuesta, diagrama de paquetes y diagrama de despliegue.
- ➢ Los diagramas de comportamiento incluyen el diagrama de casos de uso, diagrama de actividades y diagramas de estados.
- ➢ Los diagramas de interacción incluyen el diagrama de secuencia, diagrama de comunicación diagramas de tiempos y diagrama global de interacciones.

3.1.6. Calidad de software:

La calidad del producto software se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor. Son precisamente estos requisitos (funcionalidad, rendimiento, seguridad, mantenibilidad, etc.) los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y subcaracterísticas. La familia de normas ISO/IEC 25000 especifica un conjunto de características de calidad de software para su evaluación. Esta familia de normas se encuentra compuesta por 5 divisiones:

- **ISO/IEC 2500n – División de Gestión de Calidad:** estas normas definen todos los modelos, términos y definiciones comunes referenciados por todas las otras normas de la familia 25000.
- **ISO/IEC 2501n – División de Modelo de Calidad:** las normas de este apartado presentan modelos de calidad detallados incluyendo características para calidad interna, externa y en uso del producto software. El modelo de calidad del producto definido por la ISO/IEC 25010 se encuentra compuesto por las ocho características de calidad que se muestran en la Ilustración 2 Calidad en el producto software.:

Ilustración 2 Calidad en el producto software.

- ➢ **ISO/IEC 2502n – División de Medición de Calidad:** estas normas incluyen un modelo de referencia de la medición de la calidad del producto, definiciones de medidas de calidad (interna, externa y en uso) y guías prácticas para su aplicación.
- ➢ **ISO/IEC 2503n – División de Requisitos de Calidad:** las normas que forman este apartado ayudan a especificar requisitos de calidad que pueden ser utilizados en el proceso de elicitación de requisitos de calidad del producto software a desarrollar o como entrada del proceso de evaluación.
- ➢ **ISO/IEC 2504n – División de Evaluación de Calidad:** este apartado incluye normas que proporcionan requisitos, recomendaciones y guías para llevar a cabo el proceso de evaluación del producto software. ISO/IEC 25040 define el proceso para llevar a cabo la evaluación del producto software. Dicho proceso de evaluación consta de un total de cinco actividades:
    - **Actividad 1: Establecer los requisitos de la evaluación:** el primer paso del proceso de evaluación consiste en establecer los requisitos de la evaluación.
        - ✓ Tarea 1.1: Establecer el propósito de la evaluación.
        - ✓ Tarea 1.2: Obtener los requisitos de calidad del producto.
        - ✓ Tarea 1.3: Identificar las partes del producto que se deben evaluar.
        - ✓ Tarea 1.4: Definir el rigor de la evaluación
    - **Actividad 2: Especificar la evaluación:** en esta actividad se especifican los módulos de evaluación (compuestos por las métricas, herramientas y técnicas de medición) y los criterios de decisión que se aplicarán en la evaluación.
        - ✓ Tarea 2.1: Seleccionar los módulos de evaluación
        - ✓ Tarea 2.2: Definir los criterios de decisión para las métricas
    - **Actividad 3: Diseñar la evaluación:** en esta actividad se define el plan con las actividades de evaluación que se deben realizar.

- ✓ Tarea 3.1: Planificar las actividades de la evaluación

- **Actividad 4: Ejecutar la evaluación:** en esta actividad se ejecutan las actividades de evaluación obteniendo las métricas de calidad y aplicando los criterios de evaluación.
  - ✓ Tarea 4.1: Realizar las mediciones
  - ✓ Tarea 4.2: Aplicar los criterios de decisión para las métricas
  - ✓ Tarea 4.3: Aplicar los criterios de decisión de la evaluación
- **Actividad 5: Concluir la evaluación:** en esta actividad se concluye la evaluación de la calidad del producto software, realizando el informe de resultados que se entregará al cliente y revisando con éste los resultados obtenidos.
  - ✓ Tarea 5.1: Revisar los resultados de la evaluación
  - ✓ Tarea 5.2: Crear el informe de evaluación
  - ✓ Tarea 5.3: Revisar la calidad de la evaluación y obtener feedback
  - ✓ Tarea 5.4: Tratar los datos de la evaluación

3.2. INGENIERÍA INVERSA:

3.2.1. CONCEPTUALIZACIÓN:

La función de la ingeniería inversa es construir especificaciones de un mayor nivel de abstracción, a partir de un documento accesible públicamente. Más formalmente puede definirse así:

"*La ingeniería inversa consiste en analizar un sistema para identificar sus componentes y relaciones entre ellos, así como para crear representaciones del mismo en alguna que no exista, generalmente a un nivel de abstracción más elevado"* (Sánchez, y otros, 2012)**.**

La definición más amplia y aceptada de ingeniería inversa es la que dieron Chikofsky y Cross II (1990): *"La ingeniería inversa es el proceso de análisis de un sistema sujeto a identificar los componentes del sistema y sus interrelaciones, y*

crear representaciones del sistema en otra forma o en un nivel más alto de abstracción" (Reverse Engineering and Design Recovery: A Taxonomy, 1990).

3.2.2. Ingeniería inversa de software:

La ingeniería inversa de software es un término muy amplio que abarca una gran cantidad de métodos y herramientas para obtener información y conocimiento de los artefactos que software existentes y aprovecharlos en el proceso de ingeniería de software, dando solución a muchos problemas que pueden darse en este proceso. En un artículo seminal Chikofsky y Cross definen la ingeniería inversa de software como: "El proceso de analizar un sistema sujeto a la identificación de los componentes del sistema y sus inter-relaciones, y crear representaciones del sistema en otra forma o en un nivel superior de abstracción", de lo que se puede inferir que la ingeniería inversa de software está centrada en el proceso de examinación, ya que esta se basa en comprender los artefactos de software disponibles y realizar representaciones comprensibles por los humanos (Canfora, Di Penta, & Cerulo, 2011).

3.2.3. Herramienta de ingeniería inversa de software:

Son aquellas herramientas de soporte que son utilizadas para facilitar la obtención de información y resultados que se producen en el proceso de ingeniería inversa a través de artefactos de software (Canfora, Di Penta, & Cerulo, 2011).

La caracterización de las herramientas de ingeniería inversa se hace teniendo en cuenta dos criterios: el primero corresponde al aspecto estructural, centrando la atención en los elementos básicos que conforman la arquitectura genérica de este tipo de herramientas: analizador e interfaz, mientras que el segundo se centra en las propiedades comunes entre ellas. Para caracterizar las herramientas de ingeniería inversa, tomando como principal referente su arquitectura genérica, se establece la siguiente estructura: elemento, función, aspecto y característica, como se puede apreciar en la Ilustración 3.

Para la función de entrada del analizador sólo se incluyen dos características nuevas asociadas a la fuente. La primera hace referencia al tipo de fuente, asumiendo que en la actualidad el proceso de ingeniería inversa incluye, además del código fuente, colecciones de datos y documentación, y la segunda característica corresponde al tipo de aplicación y se utiliza para definir la naturaleza de la tecnología visual según la experiencia de usuario y la tecnología con la que está implementada la aplicación objeto de análisis para la herramienta de ingeniería inversa, con el ánimo de facilitar la identificación del uso de patrones arquitectónicos (Caracterización de Herramientas de Ingeniería Inversa, 2012).

Elemento	Función	Aspecto	Características
Analizador	Entrada	Supuestos (+)	Estructurales, Semánticos
		Fuente	Tipo, Modelo (+), Representación (+), Origen de datos, Tipos de datos(+) , Tipos de aplicación
		Precisión (+)	Semántica, Sintáctica
		Automatización (+)	Automática, Semiautomática, Manual
		Versiones (+)	Múltiples, Singular
	Proceso	Técnica (+)	Adaptabilidad, Automatización, Complejidad, Determinismo, Explicación, Tratamiento de aspectos indefinidos, Nivel de detalle, Incremental, Iterativo, Independiente del lenguaje, Madurez, Método, Modelo, Escalabilidad, Semántica
		Tipo de analizador	Dinámicos, Estáticos, Híbridos, Históricos
		Aspectos generales	Reconocimiento de patrones, Artefactos que recupera, Análisis incremental (*), Reparsing (*), Preprocesador de comandos configurable (*), Soporte para conmutadores de compilador(*), Capacidad para analizar Funciones/Variables externas (*), Capacidad para trabajar con información incompleta, Capacidad para trabajar con caja negra, Identificación de clonación, Ingeniería inversa de Documentos WSDL en UML
	Salida	Tipo	Modelo, Representación
		Reporte (+)	Legibilidad, Nivel de detalle, Modelo, Calidad, Representación, Tipo de datos
Interfaz	Visualización	Trazabilidad	Horizontal, Vertical
		Tipo (*)	Estática, dinámica
		Tipo de Navegación (*)	Capas, Ojo de pescado, SHriMP: Simple Hierarchical Multi Perspective view
		Calidad (+)	Sintáctica, Semántica
		Niveles de abstracción	Código, Diseño, Arquitectura, Requerimientos, Modelo del negocio
		Análisis de los resultados	Capacidades de hipertexto (*), Capacidad para analizar diferencias, Mecanismos de consulta, Capacidad para agrupar cambios relacionados, Capacidad para clasificar y combinar información desde repositorios de software diferentes y heterogéneos
	Edición	Adaptabilidad del reporte generado	Anotaciones de ayuda (*), Permite organizar el informe (*), Capacidad para mejorar el diseño del reporte de salida (*), Facilidades de edición
		Capacidad de salida (*)	Generación automática de documentación, Formatos de exportación, Impresión de informes
		Generales	Definición de proyectos, Función de búsqueda, Capacidad para seleccionar entradas representativas

Ilustración 3Caracterización de Herramientas de Ingeniería Inversa según su arquitectura (Monroy et al, 2012).

3.3. EVALUACIÓN DE SOFTWARE:

3.3.1. DESMET:

DESMET es un método multi-componente que comprende pautas para la realización de una cantidad de actividades de evaluación diferentes (selección del método de evaluación, los casos de estudio cuantitativos, los experimentos cuantitativos y análisis de características). DESMET está destinado a ayudar a un evaluador en una determinada organización para planificar y ejecutar un ejercicio de evaluación que es imparcial y fiable.

Un ejercicio de evaluación de DESMET es comparativo, debido a que se asume que existen varias maneras posibles de cumplir una tarea de ingeniería de software y se quiere identificar cuál de las alternativas es mejor en determinadas circunstancias.
Los objetos de evaluación de DESMET son:
Métodos genéricos. Un paradigma genérico para algún aspecto de desarrollo de software.
Métodos. Un enfoque específico dentro de un paradigma genérico.
Herramientas. Una aplicación de software que es compatible con actividades bien definidas.
Este separa los ejercicios de evaluación en dos tipos principales:

- Evaluaciones dirigidas establecer los efectos medibles de uso de un método o herramienta.
- Evaluaciones dirigidas a establecer un método o herramienta apropiado, es decir, qué tan bueno es un método o herramienta, y si se adapta a las necesidades y la cultura de una organización.

Los efectos medibles generalmente se basan en la reducción de la producción, el restablecimiento o mantenimiento de tiempo o costos. DESMET se refiere a este tipo de evaluación como una evaluación cuantitativa u objetivo. Las evaluaciones cuantitativas se basan en la identificación de los beneficios que se pueden esperar

de un nuevo método o herramienta para ofrecer en términos medibles y recolección de datos para determinar si los beneficios esperados son realmente entregados.

La idoneidad de un método o herramienta generalmente se evalúa en términos de las características que este proporciona, las características de su proveedor y su necesidad de formación. Las propiedades y características específicas incluidas en la evaluación se basan en las necesidades de los usuarios y los estándares de contratación organizacionales. Los evaluadores evalúan el grado en que la herramienta o método proporciona las características necesarias de una manera útil y eficaz basada (por lo general) en una opinión personal. DESMET se refiere a este tipo de evaluación como análisis de características y una evaluación identifica como una evaluación cualitativa o subjetiva.

Algunos métodos incluyen tanto elementos subjetivos como objetivos, los cuales son llamados, métodos híbridos.

Además de la separación entre evaluaciones cuantitativas, cualitativas e híbridas, hay otra separación entre evaluaciones a partir de la forma en que se organiza la evaluación. DESMET ha identificado tres formas muy diferentes de organizar un ejercicio de evaluación:

- Como un *experimento formal*, en el que en muchos casos se pide realizar una tarea (o variedad de tareas) utilizando los diferentes métodos o herramientas que se investigan. Los sujetos son asignados a cada herramienta o método de tal manera que los resultados son imparciales y pueden ser analizados usando técnicas estadísticas estándar.
- Como un *estudio de casos* en el que cada método o herramienta, objeto de la investigación, es probada en un proyecto real utilizando los procedimientos estándares de desarrollo de proyectos.
- Como una *encuesta* que se pide a un ejecutivo u organizaciones que han utilizado métodos o herramientas específicas en proyectos que

proporcionen información sobre estos. Información obtenida de los usuarios se puede analizar utilizando técnicas estadísticas estándar.

3.4. SOFTWARE:

El término software se suele atribuir a John W. Tukey quien, en un artículo publicado en 1957 en la revista American Mathematical Monthly, introdujo por primera vez el término. La idea de software en los años 50 era prácticamente un sinónimo del término ‹‹programa de computadora››, es decir, un artefacto que proporciona las instrucciones necesarias para que una computadora lleve a cabo una tarea, sin embargo en la actualidad esta definición es demasiado específica (Sánchez, y otros, 2012).

Una definición más amplia es la que proporciona el diccionario Merriam-Webster, traducida en el libro "Ingeniería de Software un enfoque desde la guía" como de la siguiente manera: *"Software es el conjunto de programas, procedimientos y documentación relacionada que se asocia con un sistema, y especialmente con un sistema de computadora. En un sentido especifico, software son los programas de computadora"* (Sánchez, y otros, 2012).

La IEEE se refiere a software como *"el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación"*. Visto de esta manera el software es el soporte lógico de la computadora; programas que sirven para interactuar con el sistema y que permite que esta pueda dirigir sus componentes físicos a través de instrucciones.

4. METODOLOGÍA

4.1. TIPO DE INVESTIGACIÓN

Basándose en las características de este proyecto se clasifica como una investigación aplicada, ya que se basa en la adquisición de un conocimiento para su posterior aplicación y utilización, obteniendo consecuencias prácticas a las que pueda llevar el mismo (Zorrilla, 1993); teniendo en cuenta como criterio el lugar y los recursos donde se obtiene la información requerida se clasifica como documental, ya que esta investigación está basada en la obtención de conocimiento a través de libros, artículos, búsquedas en internet, etc. en una primera instancia y en la prueba directa de las herramientas que serán objeto del estudio; según el lugar en el que se desarrollará la investigación se clasifica como una metodología de laboratorio debido a que este se llevará a cabo en un ambiente artificial.

Como técnica de recolección de información se utilizó la revisión documental de cada una de las herramientas objeto de estudio, además se recolectó la información que se obtuvo al aplicar los escenarios de valoración y los resultados obtenidos serán objeto de análisis estadístico.

4.2. MÉTODO

Para el logro de los objetivos específicos del proyecto que se encuentran orientados al cumplimiento del objetivo general, se aplicó el método híbrido de evaluación, según la metodología DESMET, aplicando las siguientes actividades:

Actividad 1. Revisión documental con el fin de identificar las herramientas de ingeniería inversa existentes, utilizando los motores de búsqueda disponibles en la web y la literatura científica relacionada con tema.

Actividad 2. Selección de las herramientas de ingeniería inversa de código abierto a partir del listado obtenido en la actividad anterior**.**

Actividad 3. Realización de una revisión bibliográfica sobre la caracterización de herramientas de ingeniería inversa, para establecer los criterios de valoración.

Actividad 4. A partir de los criterios de valoración identificados, diseñar los escenarios de valoración.

Actividad 5. Descarga e instalación de las herramientas de ingeniería inversa seleccionadas.

Actividad 6. Aprender a utilizar las herramientas seleccionadas. Para la correcta ejecución del estudio, es necesario que los investigadores manejen las herramientas seleccionadas y puedan sacar provecho de cada una de las características que estas ofrecen.

Actividad 7. Valorar las herramientas identificadas. Se aplicaron los escenarios de valoración a cada una de las herramientas seleccionadas, tomando registro de los resultados obtenidos.

Actividad 8. Aplicar análisis estadístico estableciendo un comparativo entre los resultados obtenidos para cada herramienta.

Actividad 9. Identificar las funcionalidades que aún no han sido implementadas en las herramientas de ingeniería inversa de código abierto existentes.

Actividad 10. Publicación de resultados, mediante la elaboración el informe final del proyecto y la presentación de una ponencia.

5. RESULTADOS

5.1. Identificación de herramientas de ingeniería inversa

Para cumplir con el primer objetivo se realizó una búsqueda sistemática de literatura en las que se hiciera mención de herramientas de ingeniería inversa, para luego seleccionar las de código abierto a través de los siguientes criterios:

i. Primeramente se realizó una búsqueda de literatura en 5 bases de datos (IEEExplore, ACM Digital Library, SpringerLink, Citeseer, Wiley online Library), en las que se consultaron las siguientes cadenas de búsqueda: “Reverse engineering tools”, “design recovery tools”, “architecture recovery tools”, obteniendo los resultados que se muestran en tabla 4 (Ver ANEXO I).

ii. Para incluirlos en la revisión bibliográfica se tomó en cuenta que estos fueran:
   - Documentos cuyo tema principal sea la ingeniería inversa en el contexto de la ingeniería de software.
   - Artículos de revistas o actas de conferencia, capítulos de libros, reportes técnicos y literatura gris publicados en cualquier fecha que presenten resultados de estudios empíricos

iii. Luego se realizó un compilado de la bibliografía tomando en cuenta características como: título, citaciones, año, publicación, medio de publicación, editor y autores (Ver ANEXO II).

iv. Al tener el compilado se descartaron las literaturas duplicadas y se verificaron solo aquellas que hablaran realmente de herramientas de ingeniería inversa.

v. Del listado de literaturas resultante se detalló información importante de cada una de las herramientas que eran mencionadas con el fin de identificar aquellas que fueran de código abierto (Ver ANEXO **II)**, teniendo en cuenta que los valores de tipo de licencia son: NE = No Especifica, CA = Código Abierto, P = Privativa

Tabla 1. Resultados de la búsqueda de literatura

Base de datos / Cadena de búsqueda	IEEEXPORE		ACM		SpringerLink		CiteSeer		Wiley	
	Encontrados	Seleccionados	Encontrados	Seleccionados	Encontrados	Seleccionados	Encontrados	Seleccionados	Encontrados	Seleccionados
reverse engineering tool	52	50	32	31	77	53	56	28	56	28
design recovery tool	5	5	7	7	3	3	9	9	0	0
architecture recovery tool	10	10	3	2	6	6	2	2	1	1
Totales	67	65	42	30	83	62	67	39	57	29
			Documentos cuyo tema principal es la ingeniería inversa en el contexto de la ingeniería de software							225
			Repetidos							178
			Herramientas							66

vi. Luego de tener el listado de herramientas de código abierto se pasó a escoger las herramientas (Ver ANEXO II) que fueron evaluadas en este trabajo teniendo en cuenta los siguientes criterios:

- Año de primera versión.
- Lenguaje de programación.
- Documentación.
- Manuales de instalación.
- Manual del usuario.
- Año de última versión.
- Disponibilidad de la herramienta para hacer las respectivas pruebas

Dando como resultado las siguientes herramientas:

- Reclipse
- MARPLE
- Doxygen
- Q-ImPrESS

5.2. Criterios de valoración de las herramientas seleccionadas

Para cumplir con el segundo objetivo se realizó un análisis de las características que generalizan las herramientas de ingeniería inversa (Ilustración 3), teniendo en cuenta aquellas que representan un valor agregado a la investigación según lo definido en el alcance de este trabajo.

- **Tipos de analizador:** Canfora et al (2011), distingue para las herramientas cuatro tipos de analizadores: Estático: extrae la estructura del sistema; dinámico: extrae el comportamiento del sistema; híbrido: extrae tanto la estructura como el comportamiento del sistema. Adicionalmente está el analizador histórico que extrae información que permite establecer la evolución de un producto software.

- **Reconocimiento de patrones:** indica si las herramientas seleccionadas reconocen o no la existencia del uso de patrones a partir de la entrada que se le ha suministrado (Monroy et al, 2012).
- **Artefactos recuperados:** corresponde a las piezas de información que recupera el analizador de las herramientas seleccionadas y está directamente relacionado con su tipo, el modelo y la representación de la salida.
- **Modelo:** corresponde a la técnica utilizada por cada una de las herramientas para representar un sistema.
- **Representación:** indica si los elementos son interpretados para la correcta visualización de las salidas que arroja cada una de las herramientas.
- **Niveles de abstracción:** capacidad que tienen las herramientas seleccionadas para permitir visualizar los resultados que arroja, desde el código fuente hasta el modelo del negocio, pasando por el diseño, las vistas arquitectónicas y los requerimientos del sistema (Monroy et al, 2012).
- **Capacidades de hipertexto:** capacidad que tienen las herramientas de reconocer y organizar entradas representativas, y vincularlas de manera dinámica a través de fragmentos de texto o gráficos.
- **Mecanismos de consulta:** indica si las herramientas seleccionadas ofrecen mecanismos para la búsqueda de información dentro de la salida e incluso en el código fuente.
- **Capacidad de organizar cambios relacionados:** indica si las herramientas seleccionadas agrupan la información que comparte características comunes.
- **Anotaciones de ayuda:** anotaciones de las herramientas que permiten guiar al usuario por la aplicación, generando sugerencias y haciendo más interactivo el análisis de los resultados.
- **Permite organizar el informe:** capacidad que tienen las herramientas de modificar el orden en el que se encuentran las salidas generadas.
- **Capacidad para mejorar el diseño del reporte de salida:** indica el grado de acceso a la edición de los reportes generados por las herramientas seleccionadas.

- **Formatos de exportación:** corresponde a los estándares con los que se almacenan los archivos arrojados como salida por las herramientas.
- **Capacidad de seleccionar entradas representativas:** capacidad que tienen las herramientas seleccionadas para definir las entradas más importantes.

5.3. Diseño de los escenarios de valoración

A partir de lo propuesto en la metodología DESMET se identificaron los elementos importantes que se deben tener en cuenta para terminar la planeación y realizar un diseño apropiado de los estudios de caso, ya que en esta metodología se propone seguir los lineamientos definidos por Glass (1997), en el cual se plantean las siguientes fases:

5.3.1. PLANEACIÓN

Se evaluaron las herramientas de ingeniería inversa seleccionadas con el fin de conocer las funcionalidades que estas ofrecen, enfocados principalmente en los artefactos de diseño y arquitectura que recuperan para lo cual se verificaron los siguientes factores claves a través de un estudio de casos, teniendo en cuenta la propuesta de caracterización de herramientas de ingeniería inversa establecida por Monroy et al (2012):

- Tipo de analizador: dinámico, estático, híbrido e histórico.
- Aspectos generales: reconocimiento de patrones, artefactos que recupera.
- Tipos de salidas: modelo, representación.
- Niveles de abstracción: código, diseño, arquitectura, requerimientos, modelo del negocio.
- Capacidad para analizar los resultados: capacidades de hipertexto, mecanismos de consulta, capacidad para agrupar cambios relacionados.
- Adaptabilidad de los reportes generados: anotaciones de ayuda, permite organizar el informe, capacidad para mejorar el diseño del reporte de salida.
- Capacidad de las salidas: formatos de exportación.

- Aspectos generales de la edición: capacidad de seleccionar entradas representativas.

De las cuales, la única variable que necesita ser controlada es el código fuente (origen de los datos), debido a que la precisión del estudio fue mayor debido a que todas las evaluaciones a las herramientas se realizaron recuperando la arquitectura de un mismo código fuente, esto brindó mayor seguridad al analizar cada factor clave y al comparar las herramientas estudiadas. El código fuente escogido para la evaluación fue ArgoUML 0.34 (Ver Anexo IV) ya que este se encuentra escrito en Java, al ser de código abierto se tiene acceso al código fuente y se consideró lo suficientemente robusto para evidenciar las diferencias en las características evaluadas en este estudio.

5.3.2. DISEÑO

5.3.2.1. DEFINICIÓN DE TAREAS

- Seleccionar el sistema objeto de estudio.
- Configuración de las herramientas.
- Realizar el proceso de ingeniería inversa registrando los valores de las variables a evaluar.
- Documentar y explicar todas las decisiones tomadas.
- Comparar los resultados obtenidos.
- Identificar funcionalidades que aún no han sido implementadas.

5.3.2.2. DISEÑO DEL ESTUDIO DE CASOS

Para la evaluación se preparó un plan de ejecución en el que se identifiquen todos los problemas posibles para que esta se ejecute sin problemas. Incluyendo las medidas necesarias en el análisis y procedimientos de recolección de datos (Kitchenham, 1996), los aspectos a verificar y de los factores a considerar son descritos en detalle por Glass (1997), el cual se tomó como referencia principal

para el diseño del estudio de casos, en el que se identificaron las características a evaluar, sus aspectos, tipos de función y elemento al cual pertenecen. Esto se realizó utilizando la ficha de caracterización elaborada a partir de la propuesta de Monroy et al. (2012) como se puede ver en la tabla 2 (Ver ANEXO III**)**.

Tabla 2 Ficha de caracterización de herramientas

Elemento	Función	Aspecto	Característica
Analizador	Proceso	Tipo de Analizador	Dinámico
			Estático
			Híbrido
			Histórico
		Aspectos Generales	Reconocimiento de Patrones
			Artefactos que recupera
	Salida	Tipo	Modelo
			Representación
Interfaz	Visualización	Niveles de Abstracción	Código
			Diseño
			Arquitectura
			Requerimientos
			Modelo del Negocio
		Análisis de los Resultados	Capacidades de Hipertexto
			Mecanismos de consulta
			Capacidad para agrupar cambios relacionados
	Edición	Adaptabilidad del reporte generado	Anotaciones de ayuda
			Permite organizar el informe
			Capacidad para mejorar el diseño del reporte de salida
		Capacidad de salida	Formatos de exportación
		Generales	Capacidad para seleccionar entradas representativas

5.3.3. GESTIÓN

En esta sección se presenta el resultado de aplicar las herramientas de ingeniería inversa sobre el código fuente seleccionado para el estudio, las diferentes salidas que estas arrojan y los artefactos que estas generan.

5.3.3.1. DOXYGEN

Es la herramienta estándar para generar documentación en fuentes C++ comentadas, pero también soporta otros lenguajes de programación populares como C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl y hasta cierto punto D.

La instalación de Doxygen es muy sencilla ya que cuenta con un asistente de instalación que simplifica el trabajo de instalación a solo dar clic en el botón siguiente hasta que se haya finalizado, sin embargo para la generación de una mejor salida este cuenta con la opción de usar GraphViz para la generación de diagramas más detallados, y dependiendo de los tipos de salida que se quiera se tendrán que instalar ciertos programas extras para la generación de estos: Microsoft HTML help workshop para la generación de la salida en HTML, Qt Software para la generación de archivos comprimidos Qt, y LaTeX y Ghostscript para la generación de PDF o uso de fórmulas científicas.

Al abrir el doxywizard de doxygen se configuraron los datos principales del nuevo proyecto a crear, como son el nombre del proyecto, el directorio en el cual se encuentra el código fuente y el directorio en el cual se almacenarán los resultados (Ilustración 4. Interfaz principal Doxygen).

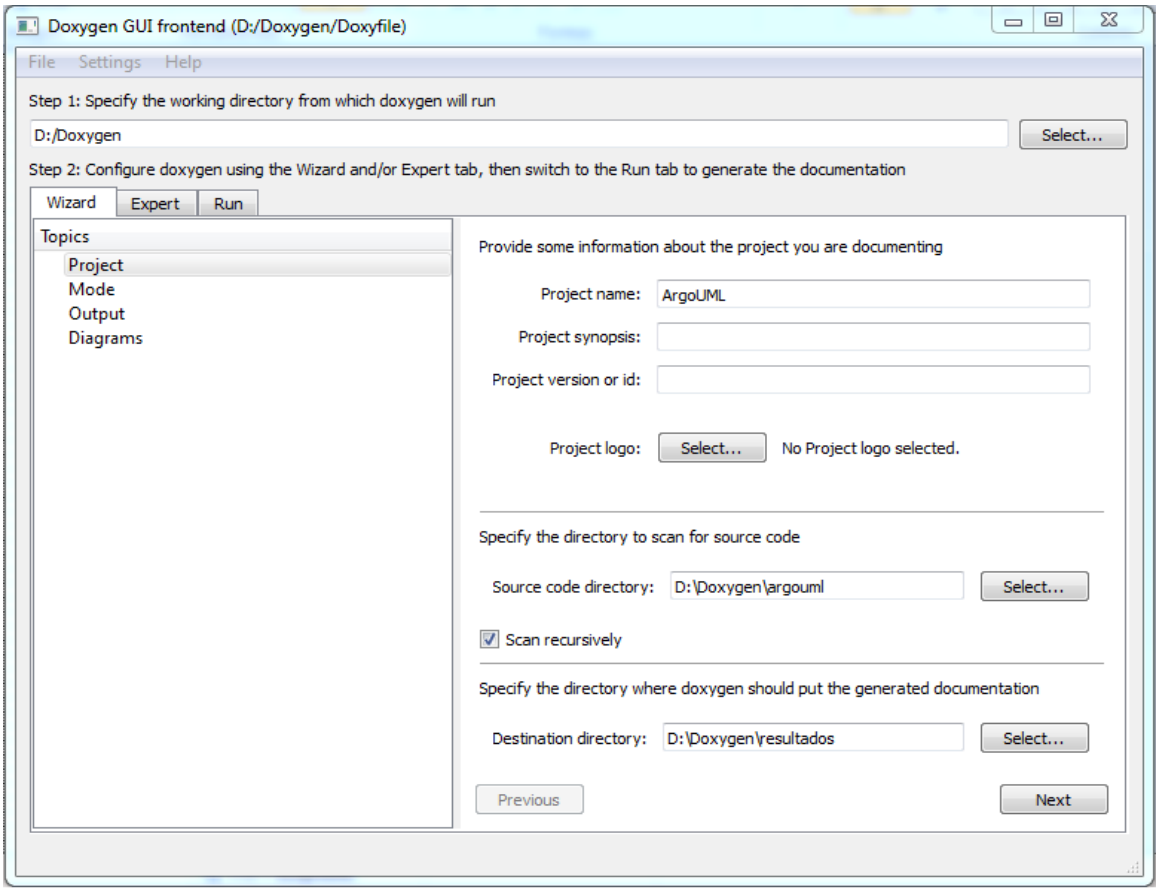


Ilustración 4. Interfaz principal Doxygen

Luego se indicó el modo de extracción, ya sea solo para las entidades documentadas o para todas las entidades (para que los resultados fueran más completos para el estudio se aplicó para todas las entidades incluyendo las referencias cruzadas en los resultados), y se selecciona el lenguaje que se va a optimizar, el cual en nuestro caso es Java, como se muestra en la Ilustración 5.

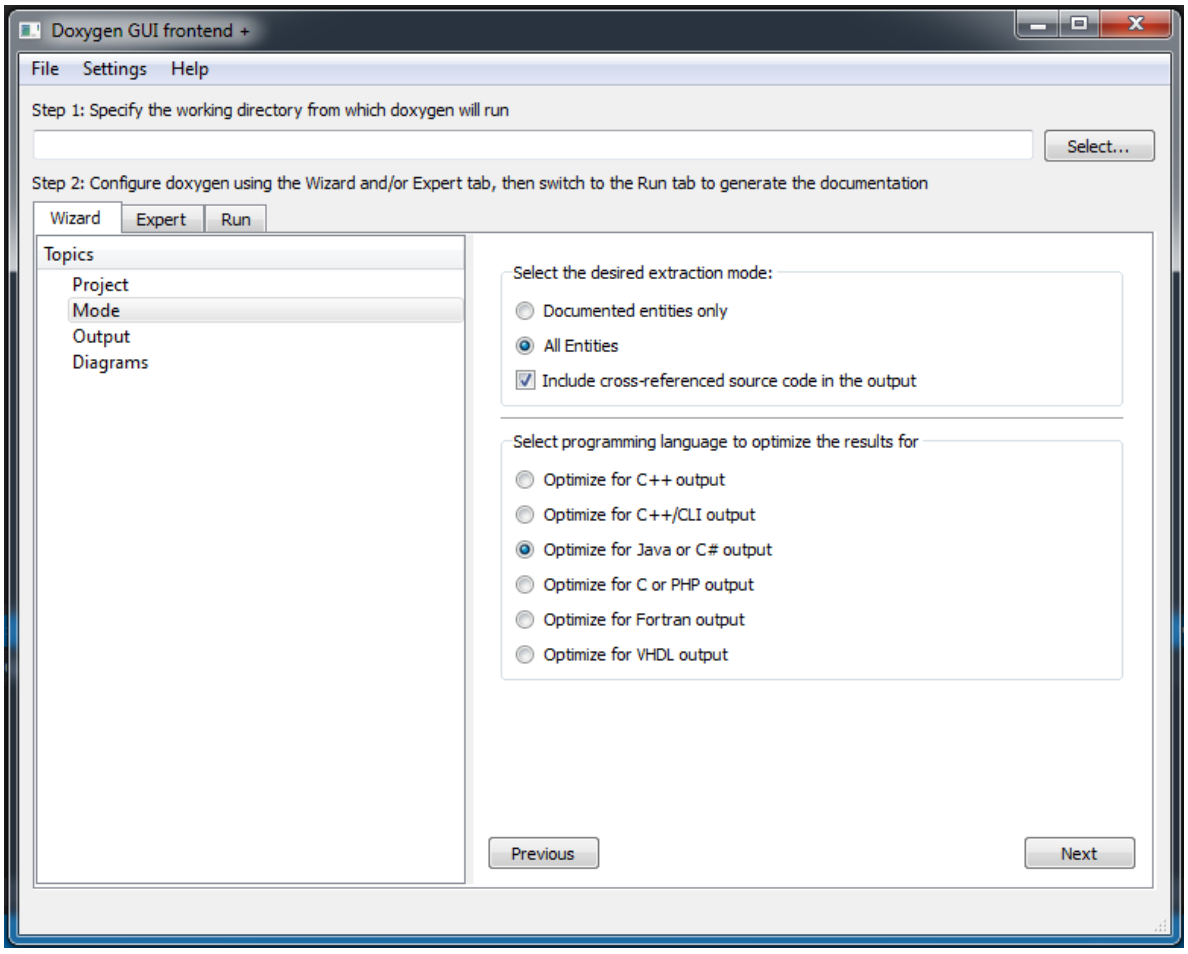


Ilustración 5. Selección del lenguaje de entrada

Posteriormente se seleccionaron los formatos de salida a generar en el cual se incluyen html con panel de navegación y búsqueda, LaTeX como intermediario para enlaces a PDF, páginas MAN, RTF y XML (Ilustración 6)

Y finalmente se seleccionan los diagramas a generar ya sea con el generador de diagramas de clase por defecto o utilizando el programa externo GraphViz para generar otros tipos de diagramas como se puede observar en la Ilustración 7 y luego se ejecuta Doxygen..

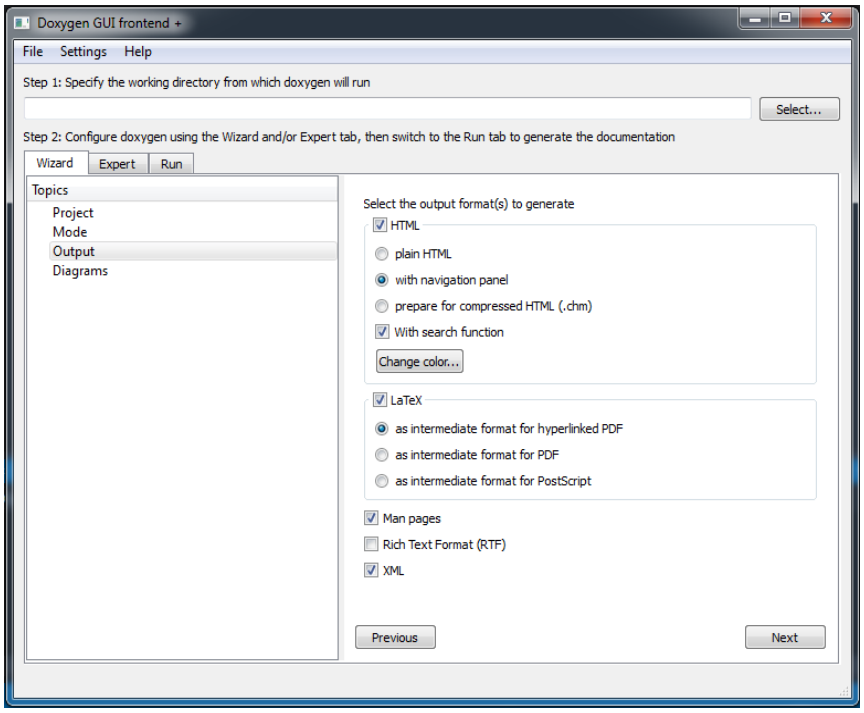


Ilustración 6. Formatos a generar

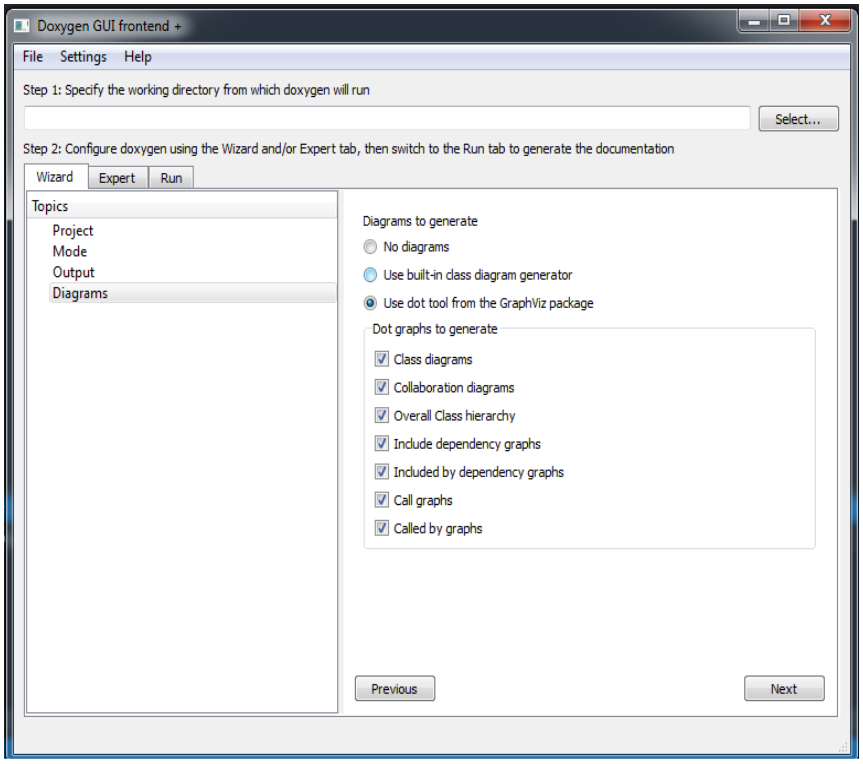


Ilustración 7. Selección de diagramas a generar

Con esto configurado, se habilita la opción de ver la salida en HTML abriéndose en el navegador y mostrando una ventana principal como se puede observar en la Ilustración 8.

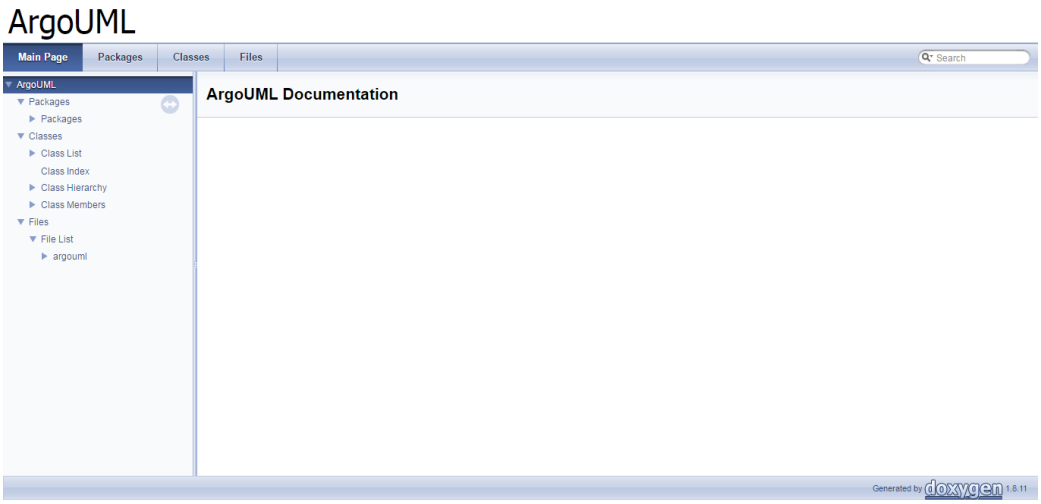


Ilustración 8. Vista Html del resultado generado por Doxygen

Mostrando una navegación con los paquetes del proyecto, el listado de clases, los índices, la jerarquía de clases, los miembros y el listado de archivos encontrados, ofreciendo una descripción detallada de las clases, funciones, métodos, parámetros y excepciones encontrados, y mostrando los diagramas generados.

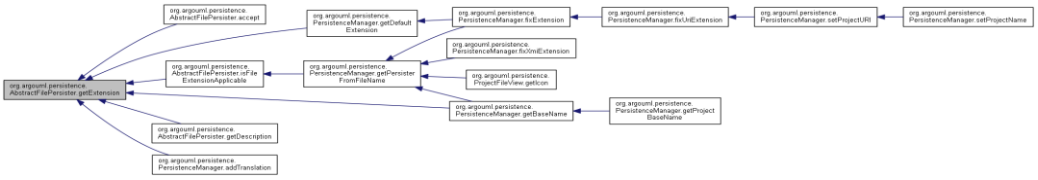


Ilustración 9. Ejemplo de Diagrama de clases generado con Doxygen con GraphViz

5.3.3.2. RECLIPSE

Reclipse es una herramienta de detección automática de patrones en un código fuente. Provee una detección de patrones estáticos y dinámicos, el análisis estático usa un lenguaje de especificación de patrones basado en diagramas de objetos

llamado patrones estructurales, también provee un diagrama de clases que muestra las posibles implementaciones de patrones de diseño llamado patrones candidatos.

Para crear un catálogo de patrones se hace clic derecho en nuestro proyecto base -> New -> Other -> Reclipse -> Seleccionando Pattern Specification Catalog y haciendo clic en el botón siguiente, se le asigna un nombre el archivo del modelo y luego se da clic en el botón siguiente, posteriormente se nombra el archivo de diagramas y haciendo clic en el botón siguiente se nombra el catalogo, seleccionando posteriormente el metamodelo SourceCodeDecorator y se finaliza.

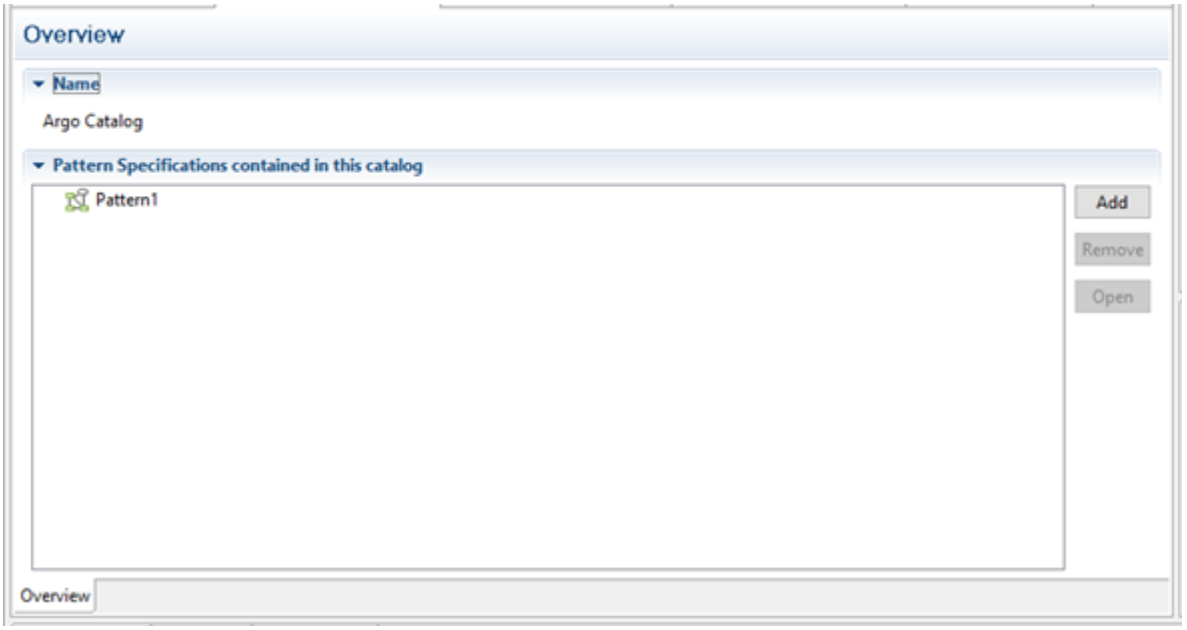


Ilustración 10. Reclipse Overview

Al abrir el archivo de diagramas, se añade un nuevo patrón y lo se le da el nombre 'FindPrimitiveComponents'; se le da doble clic en el para abrir en una nueva pestaña y crear el patrón (Ilustración 11).

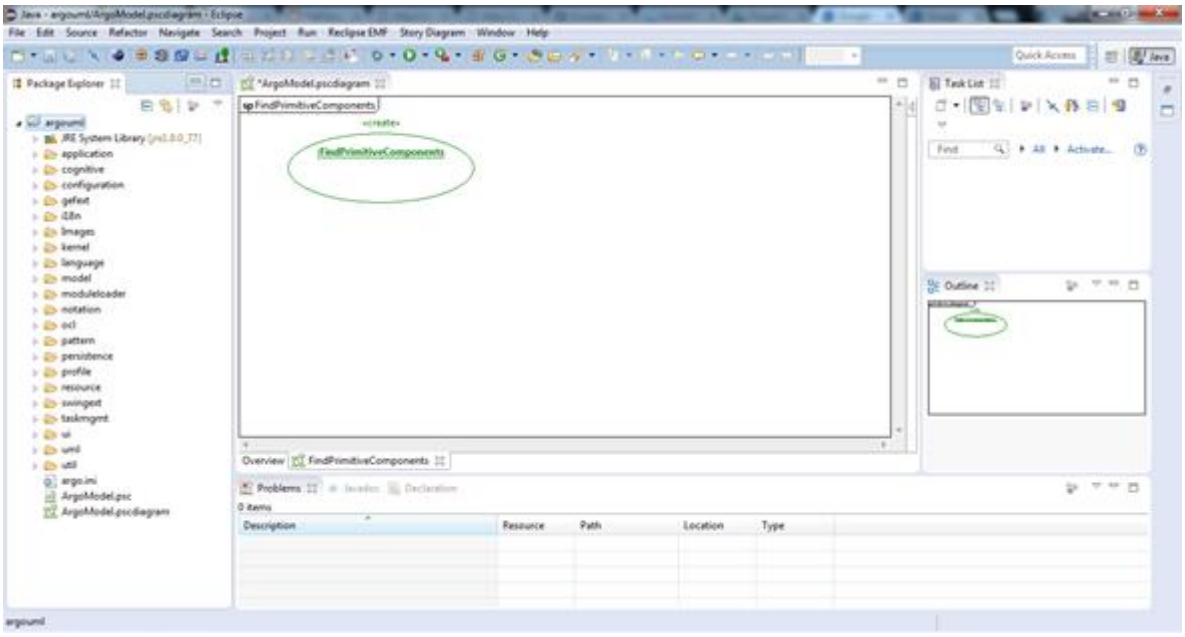


Ilustración 11. Reclipse FindPrimitiveComponents view

Este patrón encuentra todos los componentes primitivos en el modelo de entrada, mostrándolos posteriormente en la pestaña presentada en la Ilustración 11. Reclipse FindPrimitiveComponents view.

Siguiente, se añade el segundo patrón ‘AcquireReleasePair’ desde la pestaña overview (Ilustración 12).

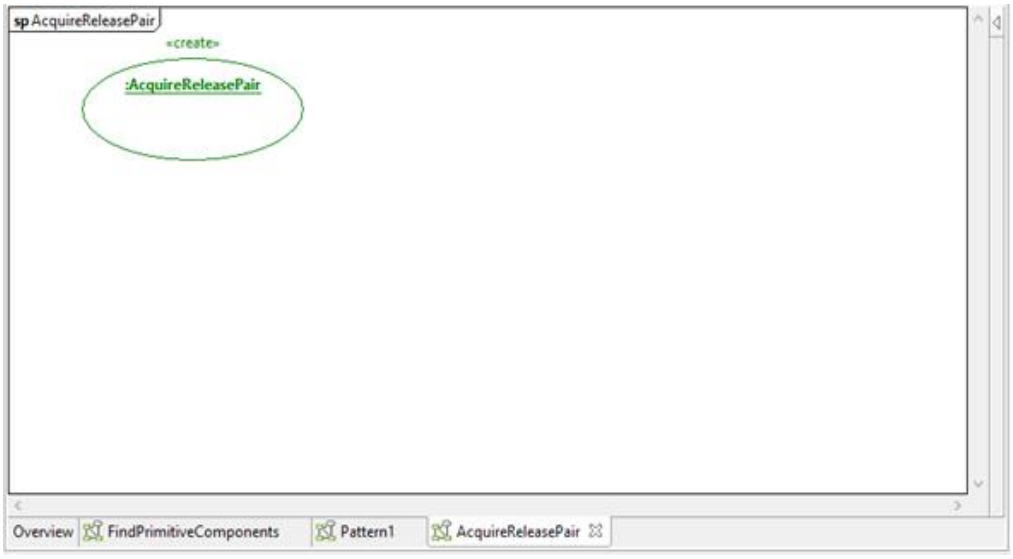


Ilustración 12. Reclipse AcquireReleasePair overview

Este patrón muestra los bloques sincronizados en nuestro modelo de entrada. Lo cual arroja el resultado que se ve en la Ilustración 12. Reclipse AcquireReleasePair overview

Para detectar patrones en el código de prueba con Reclipse se ejecutaron los siguientes pasos:

Convertir el código en un árbol de sintaxis abstracta con Modisco, seleccionando org.eclipse.modisco.java.composition.discoverer.fromProject en las configuraciones de arranque en la opción de *"select a discoverer"*, seleccionar el código fuente como *"source element"*, chequear las opciones Deep_analysis y Serialize_target en la opción "discover parameters" y luego correr (Run).

Crear un modelo SourceCodeDecorator para el código, utilizando Somox, creando una nueva instancia de Somox en las configuraciones de arranque seleccionando el código fuente de ArgoUML como proyecto y el archivo java2kdm.xmi creado en el paso anterior como archivo de entrada, y correr (Run).

5.3.3.3. MARPLE

Marple (Metrics and Architecture Recognition Plug-in for Eclipse) es una herramienta desarrollada para el reconocimiento de arquitecturas de software y patrones de diseño de programas Java.

Marple tiene las siguientes dependencias para su instalación:

- EMF (Eclipse Modeling Framework) SDK
- GEF (Graphical Editing Framework) SDK
- EMF Model Query SDK
- Zest

En la versión de modelado de eclipse (Eclipse Modeling Tools) cuenta con los plug-ins EMF y GEF pre-instalados, por lo que solo se necesitó instalar el plug-in Zest (Ilustraión 13).

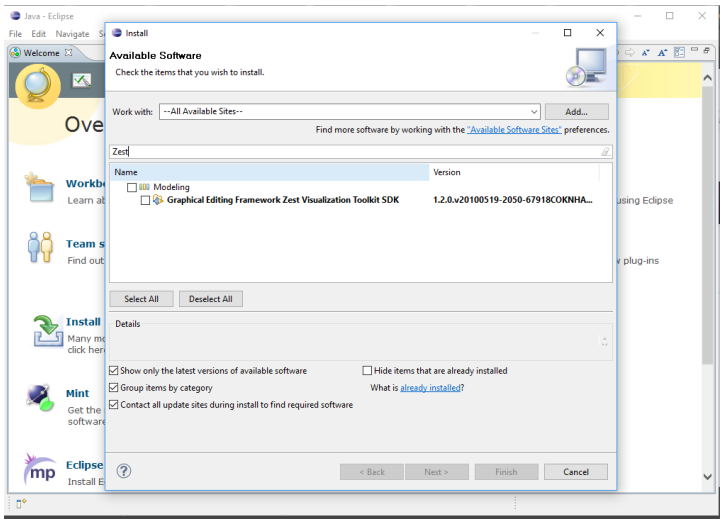


Ilustración 13. Instalación del plugin Zest

En el siguiente paso, se copia el archivo .jar de Marple dentro de la carpeta "plugins" en la instalación de Eclipse mientras se encuentra cerrado. Si se tiene problemas con la herramienta debe copiada en la carpeta "dropins" que se encuentra en la instalación de Eclipse en vez de la carpeta "plugins" (Ilustración 14).

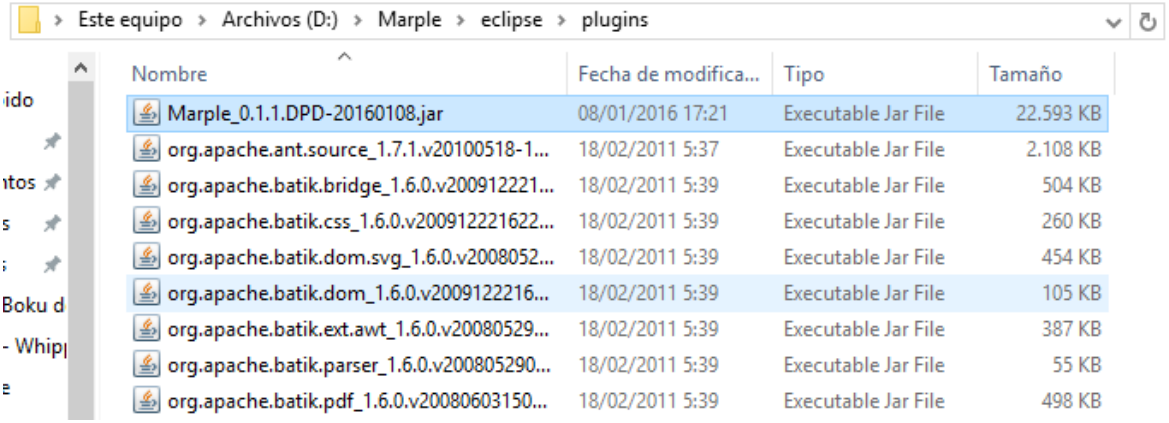


Ilustración 14. Librería Marple

Se seleccionó la opción Run > Run Configurations, al abrirse la ventana de configuración se creó una nueva "Eclipse Aplication", en "location" se colocó la

ubicación del espacio de trabajo donde se encuentra nuestro proyecto a analizar (diferente al espacio de trabajo por defecto de Eclipse) y se seleccionó la opción “Run”.

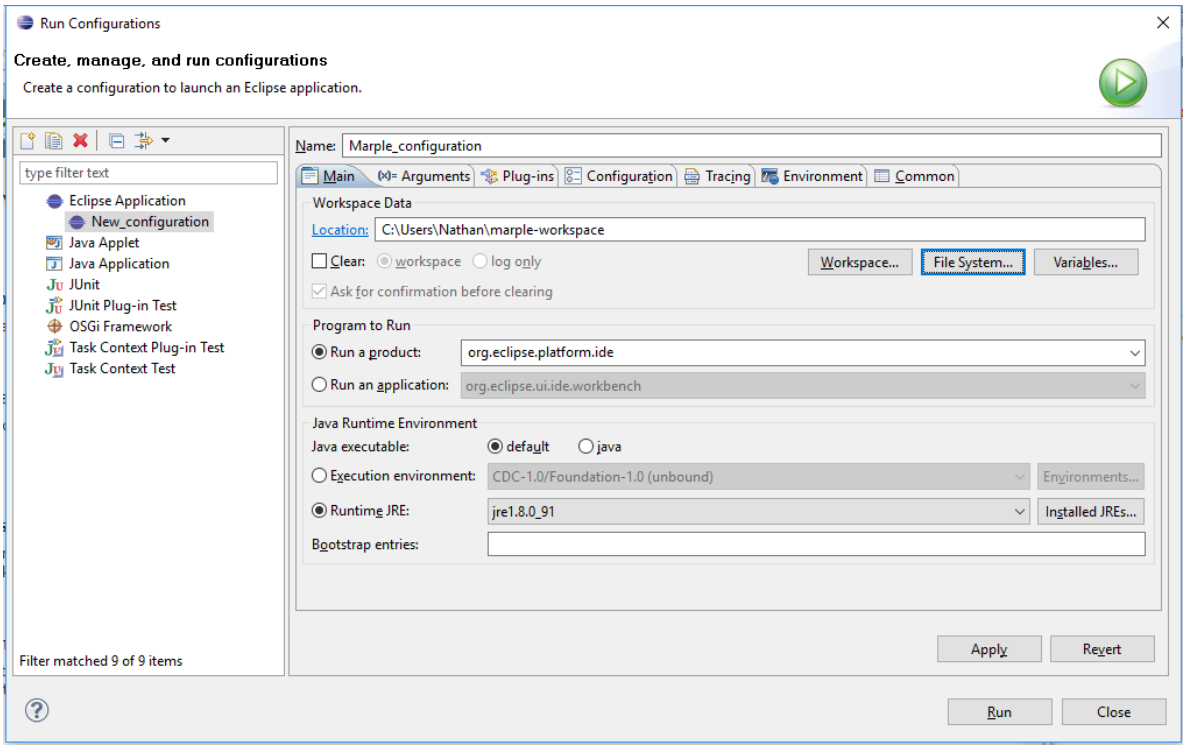


Ilustración 15. Configuración de ejecución de Marple

En la nueva ventana de eclipse se crea un nuevo proyecto Marple Developer File, luego, se hace clic derecho en el proyecto y se selecciona la opción “Other” del menú “New”. En la nueva ventana se abre el folder llamado Marple y se selecciona la opción “Marple Developer File” dando clic en el botón siguiente.

Luego se hace clic derecho en la imagen del semáforo y se selecciona la opción “Run”. Cuando el análisis esté completo, la luz verde encenderá y entonces al dar clic derecho en el semáforo nuevamente y seleccionando la opción “Show results”, se carga una vista como en la Ilustración 16. Resultado de la detección de Marple

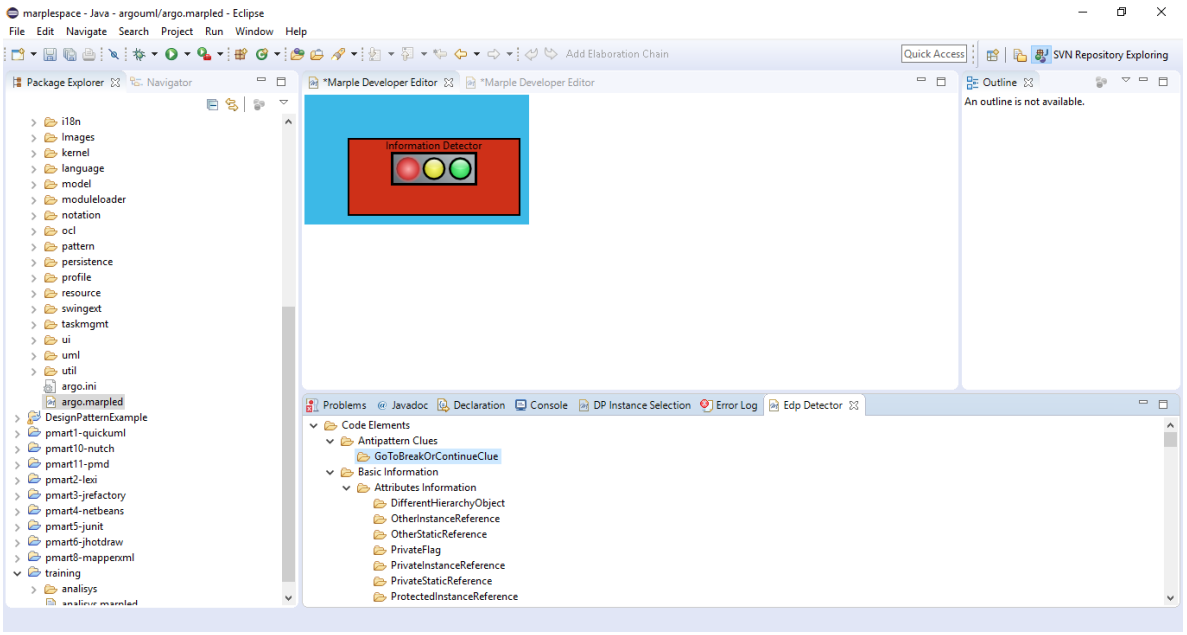


Ilustración 16. Resultado de la detección de Marple

5.3.3.4. Q-IMPRESS

Es una plataforma que les provee a los ingenieros de software una evaluación de calidad de las múltiples alternativas de evolución de un sistema, prediciendo una cantidad diferente de atributos de calidad antes de que la implementación tome lugar, lo cual conlleva a un menor tiempo de producción mientras se asegura la calidad del sistema.

Para la instalación de Q-Impress se siguieron las instrucciones que se encontraron en la documentación "**D6.1 Annex:** Guidelines and Tool Manuals" indicado en el punto 4.1 "Q-ImPrESS IDE installation" intentar instalar los plugins del sitio de descarga de Q-Impress estos no se instalaron debido a la gran cantidad de dependencias obsoletas que estos tienen, esto se intentó en diferentes versiones de Eclipse incluyendo Galileo que es la versión recomendada en el manual de instalación, sin embargo, se presentaron algunos errores en las diferentes versiones, que se relacionan a continuación, por lo tanto se decidió que para efectos del estudio

se tomaría la información encontrada en la documentación para evaluar esta herramienta.

- Cannot complete the install because one or more required items could not be found.

  Software being installed: Q-ImPrESS Trade-off Analysis Feature 1.0.0.201101311330NGT (eu.qimpress.ide.tradeoff.feature.feature.group 1.0.0.201101311330NGT)

  Missing requirement: Resultmodel Model 1.0.0.201101311330NGT (eu.qimpress.resultmodel 1.0.0.201101311330NGT) requires 'bundle de.uka.ipd.sdq.probfunction 0.0.0' but it could not be found

  Missing requirement: Resultmodel Model 1.0.0.201201131421NGT (eu.qimpress.resultmodel 1.0.0.201201131421NGT) requires 'bundle de.uka.ipd.sdq.probfunction 0.0.0' but it could not be found

  Cannot satisfy dependency:

  From: Q-ImPrESS Trade-off Analysis Feature 1.0.0.201101311330NGT (eu.qimpress.ide.tradeoff.feature.feature.group 1.0.0.201101311330NGT)

  To: eu.qimpress.resultmodel 0.0.0

- Cannot complete the install because one or more required items could not be found.

  Software being installed: Q-ImPrESS Trade-off Analysis Feature 1.0.0.201101311330NGT (eu.qimpress.ide.tradeoff.feature.feature.group 1.0.0.201101311330NGT)

  Missing requirement: Resultmodel Model 1.0.0.201101311330NGT (eu.qimpress.resultmodel 1.0.0.201101311330NGT) requires 'bundle de.uka.ipd.sdq.probfunction 0.0.0' but it could not be found

  Missing requirement: Resultmodel Model 1.0.0.201201131421NGT (eu.qimpress.resultmodel 1.0.0.201201131421NGT) requires 'bundle de.uka.ipd.sdq.probfunction 0.0.0' but it could not be found

  Cannot satisfy dependency:

From: Q-ImPrESS Trade-off Analysis Feature 1.0.0.201101311330NGT (eu.qimpress.ide.tradeoff.feature.feature.group 1.0.0.201101311330NGT)

To: eu.qimpress.resultmodel 0.0.0

- Cannot complete the install because one or more required items could not be found.

  Software being installed: SISSy Feature 1.1.0.201201171344NGT (de.fzi.sissy.bundle.feature.feature.group 1.1.0.201201171344NGT)

  Missing requirement: SISSy Feature 1.1.0.201201171344NGT (de.fzi.sissy.bundle.feature.feature.group 1.1.0.201201171344NGT) requires 'org.junit4 4.3.1' but it could not be found

- Cannot complete the install because one or more required items could not be found. Software being installed: Q-ImPrESS SAMM Model Importers Feature 1.0.0.201101311330NGT (eu.qimpress.transformations.importers.feature.feature.group 1.0.0.201101311330NGT) Missing requirement: RPG2SAM 1.4702.0 (RPG2SAM 1.4702.0) requires 'bundle de.uka.ipd.sdq.stoex 2.0.0' but it could not be found

  Cannot satisfy dependency:

  From: Q-ImPrESS SAMM Model Importers Feature 1.0.0.201101311330NGT

  (eu.qimpress.transformations.importers.feature.feature.group 1.0.0.201101311330NGT)

  To: RPG2SAM [1.4702.0]

5.4. Análisis de los resultados

Al obtener los resultados del uso de cada herramienta, se realizó un comparativo de sus funcionalidades y de esta manera se verificó las que estas no brindan y para dar cumplimiento al objetivo general de esta investigación como se muestra en la Tabla 3. Comparativo de las

funcionalidades evaluadas donde **NO:** No cumple, **SI:** cumple y **NPV:** No pudo ser verificada, la cual es una muestra de los resultados evidenciados en el ANEXO 3.

Tabla 3. Comparativo de las funcionalidades evaluadas

Herramienta / Característica	**Doxygen**	**MARPLE**	**Reclipse**	**Q-Impress**
Tipo de Analizador Histórico	NO	NO	NO	NO
Reconocimiento de Patrones	Singleton, puente, dependencia, interface, Factoría abstracta	herencia, bridge, factoria abstracta, abstract cycle, singleton, herencia múltiple, producto abstracto, prototipo	Patrones estructurales	NPV
Requerimientos	NO	NO	NO	NPV
Modelo del Negocio	NO	NO	NO	NPV
Capacidades de Hipertexto	SI	NO	NO	NPV
Mecanismos de Consulta	SI	NO	NO	NPV
Capacidad para Agrupar Cambios Relacionados	SI	SI	SI	NPV
Permite Organizar el Informe	NO	NO	NO	NPV
Capacidad para	SI	NO	NO	SI

Mejorar el Diseño del reporte de salida				
Formatos de Exportación	HTML, Qt Compressed file, LaTex, PDF, Man pages, RTF, XML, PostScript, compressed HTML	Dati	psc, pscdiagram, xmi, java	SI
Capacidad para Seleccionar Entradas Representativas	SI	NO	NO	SI

De esta manera, se encontraron tanto funcionalidades en la documentación que no pudieron ser verificadas, como funcionalidades que fueron encontradas en los escenarios de prueba y que no aparecen en la documentación de las herramientas, sin embargo aún hay funcionalidades que estas herramientas no brindan y que deben ser implementadas. De estas funcionalidades se identificaron las siguientes: tipo de analizador histórico, requerimientos, modelo del negocio, la capacidad para organizar el informe de salida y aunque estas reconocen patrones de diseños y patrones estructurales, carecen de reconocimiento de patrones de comportamiento, funcionalidades que le brindarían gran valor agregado a las herramientas de ingeniería inversa de código abierto actuales.

6. CONCLUSIONES Y RECOMENDACIONES

Como resultado de la investigación se identificó que las herramientas de ingeniería inversa de código abierto no brindan funcionalidades que darían gran valor agregado a estas, como lo son: análisis histórico, recuperación de patrones de comportamiento, recuperación de requerimientos, del modelo del negocio y la capacidad para organizar el informe de salida; dando respuesta a la pregunta de investigación planteada a través del cumplimiento del objetivo general.

Todo esto se logró gracias a que se implementaron los mecanismos necesarios para la obtención de información a través del protocolo sistemático de literatura utilizado, el cual permitió conseguir la documentación precisa para realizar un análisis documental de los aspectos seleccionados en la caracterización para cada herramienta, en los cuales se buscó detalladamente cada una de las características definidas para compararlas posteriormente con las salidas arrojadas en el uso de estas, esto, no con el fin de desmentir la información documental, porque esta proviene de revistas de las cuales resultaría muy difícil dudar, sino de comprobar que se haya realizado una instalación correcta y se hayan obtenido los conocimientos necesarios para poder utilizar las herramientas seleccionadas de manera correcta, dando así mayor validez al estudio realizado y por lo tanto dando robustez a las conclusiones llegadas en este documento.

Este estudio se realizó con el fin de conocer las capacidades que ofrecen las herramientas de ingeniería inversa de código abierto, por lo cual se obtuvo toda la información posible sobre las mismas, tanto los patrones que estas recuperan como los diagramas y artefactos que arrojan, e incluso el tipo de archivos que estas utilizan para la representación de sus salidas utilizando los lineamientos definidos por Glass (1997) y Kitchenham et al (1997) lo cual vinculado a lo definido por Monroy et al (2012) permitió definir unos criterios de valoración y diseñar los escenarios en los cuales se realizó la valoración de las herramientas, no sin antes definir de manera detallada un método de selección de estas a través de un protocolo sistemático de literatura (Ver ANEXO I).

A medida que se realizaba el estudio se encontraron resultados inesperados, como la cantidad escaza de documentación de las funcionalidades y resolución de errores de cada herramienta, lo cual dificultó en gran manera la instalación de estas y la solución a los diferentes inconvenientes encontrados en las pruebas realizadas, como lo fue el aprendizaje de las variables de configuración de Doxygen, principalmente la configuración para que este generara sus reportes de salida con la herramienta GraphViz; otro de los inconvenientes presentados fueron las múltiples dependencias necesarias para instalar las herramientas, generando problemas al momento de instalarlas e incluso usarlas; en muchos casos, la escases de documentación de la correcta utilización de las herramientas afectó la utilización de las mismas.

Sin embargo, aun con los resultados inesperados encontrados a medida que se realizaba el estudio, se lograron identificar los aspectos que fueron necesarios para llegar a la conclusión de este documento, definiendo los artefactos de diseño y arquitectura que estas herramientas recuperan, y ver para cada caso, las características de las que cada herramienta seleccionada carece (Ver ANEXO III), dando cumplimiento de esta manera a los objetivos planteados como fin de este estudio, en el que se evidencia que, aunque bien, la cantidad de artefactos que recupera cada herramienta no alcanza a barrer por completo los artefactos definidos por la ingeniería de software, estos si llegan a alcanzar el objetivo planteado por cada herramienta y logran dar respuesta a un problema específico.

El estudio se limitó a herramientas de ingeniería inversa de código abierto por razones presupuestales, por eso, para ampliar el estudio se propone que en futuras investigaciones se realice un análisis documental de las herramientas de ingeniería inversa de licencias privativas identificadas en el estudio, para establecer un paralelo entre las similitudes y diferencias relacionadas con las funcionalidades que ofrecen.

Ante los resultados inesperados mencionados previamente, se recomienda como trabajo futuro la elaboración de un catálogo de herramientas de ingeniería inversa, que las organice y clasifique teniendo en cuenta los criterios utilizados en esta investigación, lo cual facilitará la evaluación de este tipo de herramientas.

7. BIBLIOGRAFÍA

A Collaborative Demonstration of Reverse Engineering tools. **Storey, Margaret-anne D., Sim, Susan Elliott and Wong, Kenny. 2001.** Stuttgart : s.n., 2001. The Working Conference on Reverse Engineering. pp. 18-25.

A Comparative Evaluation of Dynamic Visualisation Tools. **Pacione, Michael J., Roper, Marc and Wood, Murray. 2003.** Victoria, Canada : s.n., 2003. 10th Working Conference on Reverse Engineering. pp. 80-89.

A comparative study on the re-documentation of existing software: Code annotations vs. drawing editors. **Torchiano, Marco, Ricca, Flippo and Tonella, Paolo.**

A Comparison of four Reverse Engineering Tools. **Bellay, Berndt and Gall, Harald. 1997.** Amsterdam : s.n., 1997. 4th Working Conference on Reverse Engineering (WCRE '97). pp. 2 - 11.

—. **Berndt, Bellay and Harald, Gall. 1997.** 1997, IEEE - Universidad Técnica de Viena.

A Framework for Classifying and Comparing Software Reverse Engineering and Design Recovery Techniques. **Gannod, Gerarld and H.C. Cheng, Betty. 1999.** 1999 йил, IEEE.

A method for evaluating Software Engineering methods and tools. **Kitchenham, Barbara. 1996.** 1996, Department of Computer Science U.K.

Achievements and challenges in software reverse engineering. **Canfora, Gerardo, Di Penta, Massimiliano and Cerulo, Luigi. 2011.** 2011, Communications of the ACM, pp. 142-151.

Archimetrix: Improved Software Architecture Recovery in the Presence of Design Deficiencies. **Platenius, Marie Christin, von Detten, Markus and Becker, Steffen. 2012.** Genova, Italia : s.n., 2012. 16th European Conference on Software Maintenance and Reengineering. pp. 255-264.

Assessing Software Quality by Program Clustering and Defect Prediction. **Xi, Tang, et al.**

Booch, Grady, Rumbaugh, James and Jacobson, Ivar. 2006. *El Lenguaje Unificado de Modelado.* Madrid : Pearson Educación S.A., 2006.

Caldas, Universidad Distrital Francisco José de. 2013. udistrital. *udistrital.* [Online] 2013. http://www.udistrital.edu.co:8080/documents/276352/356568/Cap6GestionRequerimientosRequisitos.pdf.

Caracterización de Herramientas de Ingeniería Inversa. **Monroy, Martín E., Arciniegas, José L. and Rodríguez, Julio C. 2012.** 2012, Información tecnológica, pp. 31-42.

CHANGE IMPACT ANALYSIS OF OBJECT-ORIENTED SOFTWARE. **Lee, Michelle. 1998.** 1998, George Mason University.

Code Extraction Algorithms which Unify Slicing and Concept Assignment. **Harman, Mark, et al. 2002.** 2002, IEEE.

DESMET: A method for evaluating Software Engineering methods and tools. **Kitchenham, Barbara, Linkman, Stephen and Law, David. 1997.** 1997, Computing & Control Engineering Journal, pp. 120 - 126.

Do Software Engineers Benefit from Source Code Navigation with Traceability? **Mäder, Patrick and Egyed, Alexander. 2011.** 2011, IEEE.

DRAE, Real Academia Española. 2001. *Diccionario de la Real Academia Española de la Lengua.* Madrid : Real Academia Española, 2001.

Empirical Studies in Reverse Engineering: State of the Art and Future Trends. **Tonella, Paolo and Torchiano, Marco. 2007.** 2007, Springer, p. 5.

Empirical studies in reverse engineering: state of the art and future trends. **Tonella, Paolo, et al. 2007.** 2007, Springer Science + Business Media.

Enhancing Source-Based Clone Detection Using Intermediate Representation. **Selim, G.M.K., King, Chun Foo and Ying, Zou. 2010.** 2010, IEEE.

Enriching Reverse Engineering with Semantic Clustering. **Kuhn, Adrian and Ducasse, Stéphane.**

Free Software Foundation. 2013. El sistema operativo GNU. *Licencias - Proyecto GNU - Free Software Foundation.* [Online] 11 16, 2013. [Cited: 11 17, 2013.] http://www.gnu.org/.

Galáz, Solange. 2008. Monografías. *Monografías.* [Online] 2008. [Cited: Mayo 6, 2011.] http://www.monografias.com.

Greg, Hoglund and Gary, McGraw. 2004. *Exploiting Software How to Break Code.* Boston : Addison Wesley, 2004.

Highly Configurable And Extensible Code Clone Detection. **Biegel, Benjamin and Diehl, Stephan. 2010.** 2010, 17th Working Conference on Reverse Engineering, WCRE 2010.

IEEE Standard 610.12-1990. **IEEE. 1993.** 1993, IEEE: Standars Collection: Software Engineering.

Inc, Janalta Interactive. 2010. TECHOPEDIA. *TECHOPEDIA.* [Online] 2010. [Cited: 02 29, 2012.] http://www.techopedia.com/.

Incremental Code Clone Detection: A PDG-based Approach. **Higo, Y., et al. 2011.** 2011, IEEE.

Ingeniería Inversa y Reingeniería aplicadas a Proyectos de Software. **Juarez-Ramirez, Reyes, Licea, Guillermo and Cristobal Salas, Alfredo. 2009.** 2009, Universidad Autónoma de Baja California.

Is Reverse Engineering Always Legal? **Cristina, Cifuentes and Anne, Fitzgerald. 1999.** 1999, IEEE.

ISO/IEC 12207. **ISO. 2008.** 2008, International Standar.

Kawaguchi, Shinji, et al.

Kitchenham, Barbara. 1996. *DESMET: A method for evaluating Software Engineering methods and tools.* Keele : s.n., 1996.

Larman, Craig. 2003. *UML y Patrones, Una Introducción al Análisis y Diseño Orientado a Objetos y al PRoceso Unificado.* Madrid : LAVEL S.A., 2003.

Ley 1273 de 2009. **República, Congreso de la. 2009.** 2009, Congreso de la República.

Media, Roy Rosening Center For History And New. 2012. Zotero. *Zotero.* [Online] 01 2012. http://www.zotero.org/.

Mozilla. 2012. Firefox Mozilla. *Firefox Mozilla.* [Online] 2012. http://www.mozilla.org.

New Frontiers of Reverse Engineering. **Canfora, Gerardo and Di Penta, Massimiliano. 2007.** 2007, IEEE Computer Society, pp. 326-341.

New Frontiers of Reverse Engineering, Future of Software Engineering. **Canfora, Gerardo and Di Penta, Massimiliano. 2007.** 2007, IEEE.

Object-oriented reengineering patterns - an overview. **Nierstrasz, Oscar, Ducasse, Stéphane and Demeyer, Serge. 2005.** Tallinn : s.n., 2005. 4th international conference on Generative Programming and Component Engineering. pp. 1–9.

OMG. 2013. OMG Homepage. *Object Management Group - UML.* [Online] Septiembre 13, 2013. [Cited: Octubre 7, 2013.] http://uml.org/.

On Designing an Experiment to Evaluate a Reverse Engineering Tool. **Storey, M. -A.D., et al. 1996.** Monterey, CA : s.n., 1996. Reverse Engineering, 1996., Proceedings of the Third Working Conference on. pp. 31 - 40.

Open Source Initiative. 2013. The Open Source Initiative. *The Open Source Initiative.* [Online] 10 22, 2013. [Cited: 11 17, 2013.] http://opensource.org/.

Perens, Bruce. 2013. Bruce Perens. *Bruce Perens.* [Online] 2013. [Cited: 11 17, 2013.] http://perens.com/.

Pilot Studies: What, Why, and How. **GLASS, Robert. 1997.** 1997, Journal of Systems and Software, pp. 85-97.

Pressman, Roger S. 2002. *Ingeniería del Software Un Enfoque Práctico.* Madrid : Mc Graw Hill, 2002.

—. 2002. *INGENIERÍA DEL SOFTWARE. Un enfoque práctico. (5: edición).* s.l. : McGRAW-HILL/INTERAMERICANA DE ESPANA, S. A. U., 2002.

Program and Interface Slicing for Reverse Engineering. **Beck, Jon and Eichmann, David. 1993.** 1993, IEE.

Redocumentation of a Legacy Banking System. **Van Geet, Joris, Ebraert, Peter and Demeyer, Serge. 2010.** 2010, International Workshop on Principles of Software Evolution, pp. 33-41.

Reverse Engineering and Design Recovery: A Taxonomy. **Chikofsky, Elliot and Cross, James. 1990.** 1990, IEEE Software, Vol 7, pp. 13–17.

Reverse Engineering and Design Recovery: a Taxonomy. **Chikofsky, Elliot J. and Cross II, James H. 1990.** 1990, IEEE.

Reverse Engineering Co-maintenance Relationships Using Conceptual Analysis of Source Code. **Grant, Scott, Cordy, James R. and Skillicorn, David B. 2011.** 2011, Working Conference on Reverse Engineering.

Reverse Engineering of Software Threads: A Design Recovery Technique for Large Multi-Process Systems. **Wilde, Norman, et al. 1997.** 1997, University of West Florida.

Rumbaugh, James, Jacobson, Ivar and Booch, Grady. 2007. *El lenguaje unificado de modelado - Manual de referencia.* Madrid : Pearson Educación, 2007.

Sánchez, Salvador, Sicilia, Miguel Ángel and Rodríguez, Daniel. 2012. *Ingenieria del Software - Un enfoque desde la guía.* Madrid : Alfaomega, 2012.

Semantic Navigation of Large Code Bases in Higher-Order, Dynamically Typed Languages. **Spoon, Alexander and Shivers, Olin.** Georgia Institute of Technology.

SHINOBI: A Real-Time Code Clone Detection Tool for Software Maintenance . **Yamashin, Takanobu, et al.** Nara Institute of Science and Technology.

SHINOBI: A Real-Time Code Clone Detection Tool for Software Maintenance. **Yamashin, Takanobu, et al. 2008.** 2008, Nara Institute of Science and Technology.

Software maintenance and evolution: a roadmap. **Rajlich, Vaclav and Bennett, Keith. 2000.** New York : s.n., 2000. ICSE '00: Proceedings of the Conference on The Future of Software Engineering. pp. 73-87.

Software Visualisation using C++ Lenses. **Westland Cain, James and Jane McCrindle, Rachel.**

Software Visualization. **Kerren, Andreas. 2008.** 2008.

StarUML. 2005. StarUML. *staruml.sourceforge.* [Online] 2005 йил. http://www.staruml.sourceforge.net.

Static Techniques for Concept Location in Object-Oriented Code. **Marcus, Andrian, et al. 2005.** 2005.

Studio, Rad. 2010. Embarcadero® RAD Studio XE2. *Embarcadero® RAD Studio XE2.* [Online] 2010. [Cited: 12 28, 2011.] http://www.embarcadero.com/products/rad-studio.

Systems, Sparx. 2000-2012. Sparx Systems. *Enterprise Architect.* [Online] 2000-2012 йил. http://www.sparxsystems.com.au/.

Tracing All Around in Reingeneering. **Ebner, Gerald and Kaindl, Hermann. 2002.** 2002, IEEE SOFTWARE.

Using Program Slicing in Software Maintenance. **Gallagher, Keith Brian and Lyle, James. 1991.** 1991, IEEE.

VP-UML. 2011. Visual Paradigm. *Visual Paradigm.* [Online] 10 2011. [Cited: 12 29, 2011.] http://www.visual-paradigm.com/product/vpuml.

Why Software Requirements Traceability Remains a Challenge. **Kannenberg, Andrew and Saiedian, Hossein. 2009.** 2009, CrossTalk The Journal of Defense Software Engineering.

8. ANEXOS

8.1. ANEXO I: PROTOCOLO DEL MAPEO SISTEMÁTICO DE LITERATURA

Este anexo se encuentra en formato digital en la carpeta Anexos de la entrega del proyecto de grado.

8.2. ANEXO II: PROTOCOLO DE RECUPERACIÓN SISTEMÁTICA DE LITERATURA

Este anexo se encuentra en formato digital en la carpeta Anexos de la entrega del proyecto de grado.

8.3. ANEXO III: FICHA DE CARACTERIZACIÓN DE HERRAMIENTAS

Este anexo se encuentra en formato digital en la carpeta Anexos de la entrega del proyecto de grado.

8.4. ANEXO IV: CÓDIGO FUENTE

Este anexo se encuentra en formato digital en la carpeta Anexos de la entrega del proyecto de grado.