UNIVERSIDAD DE COSTA RICA
SISTEMA DE ESTUDIOS DE POSGRADO

# ESTIMACIÓN DE TAMAÑO Y VISUALIZACIÓN DE UN MODELO FUNCIONAL DE APLICACIONES DE SOFTWARE SIMPLIFICADO MEDIANTE GRAFOS

Trabajo final de investigación aplicada sometido a la consideración de la Comisión del Programa de Estudios de Posgrado en Computación e Informática para optar al grado y título de Maestría Profesional en Computación e informática

OSCAR LUIS HUERTAS LARA

Ciudad Universitaria Rodrigo Facio, Costa Rica

2019

## Dedicatoria

A mis padres por impulsarme en todo momento a seguir adelante.

## Agradecimientos

Al profesor Dr. Christian Quesada López por su guía y por su disposición a ayudarme en todo momento durante la realización de este proyecto.

"Este trabajo final de investigación aplicada fue aceptado por la Comisión del Programa de Estudios de Posgrado en Computación e Informática de la Universidad de Costa Rica, como requisito parcial para optar al grado y título de Maestría Profesional en Computación e Informática."

Dra. Gabriela Marín Raventós

**Representante del Decano**

**Sistema de Estudios de Posgrado**

Dr. Christian Quesada López

**Profesor Guía**

Dr. Jeisson Hidalgo Céspedes

**Representante de la Directora**

**Programa de Posgrado en Computación e Informática**

Oscar Luis Huertas Lara

**Sustentante**

# Tabla de contenido

# Resumen

Las herramientas de soporte al proceso de desarrollo de software proporcionan opciones para crear artefactos y adoptar prácticas estandarizadas de la ingeniería del software. No obstante, la ingeniería del software se encuentra en constante evolución y a medida que aparecen nuevas tecnologías también se generan nuevos retos que hacen que la toma de decisiones sea una tarea cada vez más compleja para los interesados de un proyecto.

Una representación visual de una aplicación de software desde la perspectiva del usuario final mediante modelos funcionales puede proveer apoyo para un mejor entendimiento del comportamiento de la aplicación y del impacto de una modificación, mejora o adición de la funcionalidad. Asimismo, a las organizaciones de desarrollo de software se les dificulta el manejo de los requerimientos cambiantes, y en muchos casos el único insumo actualizado para comprender la estructura y funcionalidad de una aplicación es su código fuente, por lo que la representación de modelos funcionales obtenidos a partir éste y las métricas de tamaño y complejidad pueden representar un insumo valioso para los procesos de toma de decisiones. Las representaciones mediante grafos pueden ayudar en dichos procesos ya que facilitan el análisis visual; adicionalmente, los metamodelos basados en grafos permiten la definición de métricas orientadas a objetos que son independientes del lenguaje de programación empleado, las cuales dan una noción del tamaño y la complejidad del software que se está analizando.

El objetivo general de este trabajo de investigación es desarrollar una herramienta prototipo para visualizar modelos funcionales de aplicaciones de software extraídos mediante el análisis del código fuente. Primero, se identificaron los requerimientos para la creación de la herramienta de visualización de los modelos. Segundo, se implementó la herramienta de visualización que utiliza como entrada un modelo funcional de una aplicación de software simplificado mediante grafos y que calcula un conjunto de métricas a partir de este modelo funcional para estimar el tamaño funcional de la aplicación representada. Tercero, se evaluó la efectividad de las métricas del modelo funcional para estimar el tamaño y la complejidad de la aplicación de software.

Como resultado de la investigación se desarrolló una herramienta prototipo que permite visualizar los componentes funcionales relacionados a los requerimientos de las aplicaciones de software representadas, y se determinó que las métricas de tamaño y complejidad obtenidas a partir del modelo funcional pueden estimar correctamente el tamaño funcional de las aplicaciones evaluadas.

# Lista de cuadros

# Lista de figuras

UNIVERSIDAD DE COSTA RICA

SEP Sistema de Estudios de Posgrado

**Autorización para digitalización y comunicación pública de Trabajos Finales de Graduación del Sistema de Estudios de Posgrado en el Repositorio Institucional de la Universidad de Costa Rica.**

Yo, Oscar Huertas Lara, con cédula de identidad 113240139, en mi condición de autor del TFG titulado Estimación de Tamaño y Visualización de un Modelo Funcional de Aplicaciones de Software Simplificado Mediante Grafos,

SI [✓] NO [ ] autorizo a la Universidad de Costa Rica para digitalizar y hacer divulgación pública de forma gratuita de dicho TFG a través del Repositorio Institucional u otro medio electrónico, para ser puesto a disposición del público según lo que establezca el Sistema de Estudios de Posgrado.

Este Trabajo Final de Graduación será publicado en formato PDF, o en el formato que en el momento se establezca, de tal forma que el acceso al mismo sea libre, con el fin de permitir la consulta e impresión, pero no su modificación.

Manifiesto que mi Trabajo Final de Graduación fue debidamente subido al sistema digital Kerwá y su contenido corresponde al documento original que sirvió para la obtención de mi título, y que su información no infringe ni violenta ningún derecho a terceros. El TFG además cuenta con el visto bueno de mi Director (a) de Tesis o Tutor (a) y cumplió con lo establecido en la revisión del Formato por parte del Sistema de Estudios de Posgrado.

**INFORMACIÓN DEL ESTUDIANTE:**

Nombre Completo: Oscar Luis Huertas Lara.

Número de Carné: A52807 Número de cédula: 113240139.

Correo Electrónico: oscarhl01@gmail.com.

Nombre del Director (a) de Tesis o Tutor (a): Christian Quesada López.

Fecha: 29/08/2019.

Oscar Huertas

**FIRMA ESTUDIANTE**

Nota: El presente documento constituye una declaración jurada, cuyos alcances aseguran a la Universidad, que su contenido sea tomado como cierto. Su importancia radica en que permite abreviar procedimientos administrativos, y al mismo tiempo genera una responsabilidad legal para que quien declare contrario a la verdad de lo que manifiesta, puede como consecuencia, enfrentar un proceso penal por delito de perjurio, tipificado en el artículo 318 de nuestro Código Penal. Lo anterior implica que el estudiante se vea forzado a realizar su mayor esfuerzo para que no sólo incluya información veraz en la Licencia de Publicación, sino que también realice diligentemente la gestión de subir el documento correcto en la plataforma digital Kerwá.

# Capítulo 1. Introducción

Las herramientas de soporte al proceso de desarrollo de software, conocidas como herramientas CASE[1], proporcionan opciones para crear artefactos y adoptar prácticas estandarizadas de la ingeniería del software. No obstante, la ingeniería del software se encuentra en constante evolución y a medida que aparecen nuevas tecnologías también se generan nuevos retos que hacen que la toma de decisiones sea una tarea cada vez más compleja para los interesados de un proyecto [1]. Cuando las aplicaciones de software eran simples, administrar su arquitectura era un problema de mediana complejidad, sin embargo, con aplicaciones de software más grandes, el manejo de su arquitectura se ha vuelto un problema igualmente grande [1].

Las tareas que dependen en algún grado de la comprensión del software se tornan cada vez más complejas e imprecisas en sus resultados. Por ejemplo, crear estimaciones acertadas de tamaño y complejidad en un proyecto de software es una tarea muy difícil [3]. Una manera para minimizar estos problemas es administrar (generar y almacenar) datos históricos de proyectos de forma automática que se utilicen como entradas para generar modelos que permitan una retroalimentación más simple para la gerencia [3].

Para la estimación de software se han utilizado metodologías que tradicionalmente se aplican sobre artefactos formales construidos en las etapas de análisis de requerimientos y diseño [24]. Sin embargo, en Costa Rica, gran cantidad de organizaciones utilizan metodologías ágiles para el desarrollo de software [2], las cuáles por definición se centran en la codificación antes que en la elaboración de documentos formales; limitando en algunos casos la trazabilidad de los requerimientos, diseño y código fuente, y la posibilidad de la aplicación de estas metodologías de estimación sobre artefactos de requerimientos o diseño. Por esta razón, es común que el código fuente represente el principal artefacto del cual extraer información actualizada para la toma de decisiones. En muchos casos, este artefacto es el que se mantiene actualizado durante todo el ciclo de desarrollo de las aplicaciones.

[1] *Computer Aided Software Engineering*

Por lo anterior, surge la necesidad de adoptar herramientas de análisis y visualización que permitan obtener información (por ejemplo, modelos de representación estructural y de comportamiento) que faciliten el entendimiento de una aplicación de software, partiendo precisamente del código fuente debido a la falta de documentación. La literatura científica indica que no existe gran variedad de herramientas o técnicas para la visualización de modelos de software y que la creación de este tipo de herramientas está enfocada en áreas específicas de investigación y no en su aplicación para el entendimiento del software de gran escala [4]. Por lo que se conjetura que una herramienta de visualización de modelos funcionales generados a partir del análisis del código fuente que permita visualizar los componentes funcionales y apoyar los procesos de estimación, puede apoyar a los profesionales en sus procesos de toma de decisiones.

Cuando se aplican metodologías de medición del tamaño del software, estas generan resultados numéricos tabulados que buscan dar al analista una noción del tamaño y la complejidad de un proyecto en desarrollo. Sin embargo, el alto volumen de información resultante requiere de herramientas de soporte para su correcta administración e interpretación [4]. Wim De Pauw y otros afirman que "un modelo visual es mejor para el entendimiento del software, por encima del análisis de datos textuales" [7]. Así mismo, el *Standish Group* destaca la necesidad de procesos de medición más eficientes en cuanto al tiempo de elaboración y a sus resultados, así como un "adecuado seguimiento del progreso en forma visual" [3]. Del mismo modo, asegura que la mayoría de los proyectos de pequeño y mediano tamaño catalogados como "exitosos" cuentan con tales procesos de medición y seguimiento del progreso. Por lo que una herramienta de visualización de modelos funcionales que estime el tamaño funcional de los procesos y componentes funcionales representados puede ser de gran ayuda para los tomadores de decisiones durante el desarrollo de las aplicaciones de software.

Para realizar una representación funcional desde la perspectiva del usuario final a partir de los artefactos técnicos del software (tales como los del código fuente) se requiere una transformación del dominio técnico hacia el dominio del problema [20]. Para obtener una representación funcional a partir de estos artefactos manteniendo la trazabilidad se requieren procesos de reingeniería complejos. Con la identificación de los artefactos y piezas de código responsables de implementar cada una de las funcionalidades de una

aplicación, una herramienta de este tipo puede ser de gran ayuda para los equipos de desarrollo [5]. Además, un modelo visual puede proveer un mejor entendimiento del impacto de una modificación a un elemento existente o la adición de una nueva funcionalidad, ya que el manejo de un conjunto de requerimientos cambiante es complejo, en muchos es difícil justificar el esfuerzo, y por lo tanto el costo de una modificación [5]. Finalmente, mediante la representación visual de los diferentes elementos del software es posible identificar patrones y comportamientos en la estructura de la aplicación bajo análisis, mejorando así su administración [1].

Algunos ejemplos de herramientas de visualización han sido propuestas en la literatura como medio para facilitar el entendimiento de las aplicaciones de software. Por ejemplo, la herramienta "*Program Explorer*" [6] permite al usuario explorar la ejecución de un sistema de software basándose en las invocaciones de sus métodos y otros indicadores como la interacción entre los objetos. "*Jinsight*" [7] ofrece la capacidad de generar una representación visual de la ejecución de aplicaciones desarrolladas en lenguaje Java. Las facilidades que ofrecen herramientas de esta clase para el entendimiento del comportamiento de una aplicación de software sugieren que el análisis basado en un modelo de visualización es más rico e intuitivo que el análisis de datos en texto plano. Es aceptado que la visualización es trascendental para el entendimiento y manejo de la complejidad de los sistemas [7].

Con la utilización de herramientas de visualización, los administradores y líderes de proyectos pueden contar con información detallada y visual del producto de software durante el proceso de desarrollo, con datos para apoyar la estimación del tamaño y el esfuerzo que pueda demandar cada tarea específica. Los modelos de visualización pueden jugar un rol fundamental para los profesionales en las labores de estimación, ya que los elementos visuales complementados con las métricas numéricas son más fáciles de interpretar al permitir detectar fortalezas y debilidades de un proceso, sin la necesidad de contar con un conocimiento técnico muy detallado [1]. Por ejemplo, si un interesado del proyecto, que no conoce a fondo el lenguaje de programación que se utiliza, observa una tabla en la que se indican los objetos y clases con sus respectivas métricas de tamaño y complejidad, difícilmente podrá identificar las funcionalidades del sistema asociadas a estas mediciones. Sin embargo, al visualizar un grafo en el que se muestren los

componentes asociados a una funcionalidad, probablemente le será más fácil identificar las partes del sistema que demandan una mayor cantidad de esfuerzo y tomar decisiones informadas. Aspectos clave en el desarrollo de software como los atributos de calidad, automatización de procesos y métricas, pueden ser expresados mediante la visualización con grafos, ya que son ideales para representar la interacción e integración entre elementos de una aplicación de software [1].

Este proyecto de investigación presenta una propuesta de herramienta de estimación y visualización de modelos funcionales obtenidos a partir del análisis del código fuente mediante grafos. De esta forma se propone una estrategia para manejar la trazabilidad de los componentes funcionales (elementos de datos y los elementos transaccionales), generando una representación visual y funcional del código fuente desde una perspectiva funcional del usuario final, para apoyar las tareas que requieran de la comprensión de una aplicación de software [14] y la estimación de su tamaño y complejidad. Esta investigación analiza la utilidad de modelos funcionales basados en grafos, aprovechando la propiedad de los grafos que permite definir métricas no ambiguas orientadas a objetos, de forma independiente al lenguaje de programación utilizado para desarrollar el software en análisis [1]. Se desarrolla una herramienta prototipo para la visualización del modelo funcional suministrado como entrada, con el cual se puede representar el código fuente de una aplicación en desarrollo sin importar el nivel de abstracción o el tipo de elementos que quieran representarse, tales como objetos, clases, módulos, funciones, flujo de ejecución, utilización de recursos, entre otros. La independencia del nivel de abstracción es otra propiedad clave que sugiere la utilización de grafos como medio de representación, según Dąbrowski et al. [1].

## Objetivos

El objetivo general de este trabajo final de investigación aplicada (TFIA) es *desarrollar una herramienta prototipo para visualizar modelos funcionales de aplicaciones de software extraídos a partir del análisis del código fuente*. Los objetivos específicos son los siguientes:

1. Determinar los requerimientos para la visualización de modelos funcionales de aplicaciones de software, donde los modelos son obtenidos a partir del análisis del código fuente.
2. Implementar una herramienta prototipo para la construcción y visualización de modelos funcionales de aplicaciones de software.
3. Evaluar la efectividad del modelo funcional obtenido para determinar el tamaño y la complejidad de la aplicación de software representada.

Esta investigación se enmarca en los proyectos de investigación "Medición automatizada del tamaño funcional de aplicaciones transaccionales", No. 834-B5-A18, y "Evaluación empírica de una metodología para la automatización de la medición del tamaño funcional del software", No. 834-B8-A27, de la Universidad de Costa Rica. En dichos proyectos se desarrolló una metodología que contiene varios módulos de medición, verificación y estimación, donde la herramienta prototipo desarrollada en esta investigación se integra específicamente al módulo de medición [20]. En el Capítulo 5 se explica dicha integración en detalle.

La herramienta prototipo desarrollada obtiene como entrada un archivo de interfaz XML producto del análisis estático y dinámico del código fuente (generado por el módulo de medición [20] detallado en el Capítulo 5), calcula las métricas de tamaño y complejidad mediante grafos, y elabora una representación visual de dichos grafos, tal como se muestra en la Figura 1:


*Figura 1. Descripción del proceso.*

- **Entrada**: Archivo de interfaz XML resultante del análisis del código fuente de la aplicación de software que se desea visualizar. Este contiene la información de la estructura y comportamiento de la aplicación que se debe interpretar y visualizar.
- **Procedimiento**: Validación de estructura y sintaxis, interpretación, representación del grafo y cálculo de métricas de tamaño y complejidad.
- **Salida**: Visualización de grafo y presentación de métricas para la estimación del tamaño funcional.

El usuario meta de esta herramienta prototipo son los miembros de los equipos de desarrollo de software, de manera que tengan una perspectiva visual del sistema que están desarrollando o dando mantenimiento, y de las métricas asociadas de tamaño. Se espera que esta herramienta le permita apoyar sus labores de administración y llevar una mejor trazabilidad de las funcionalidades del sistema desarrollado.

Este documento está organizado de la siguiente manera. El Capítulo 2 describe los fundamentos teóricos utilizados en esta investigación. El Capítulo 3 describe la metodología utilizada para el desarrollo de esta investigación y para el cumplimiento del objetivo general y los objetivos específicos. El Capítulo 4 identifica los requerimientos de la herramienta de visualización. El Capítulo 5 describe el desarrollo del prototipo de la herramienta. El Capítulo 6 describe la evaluación empírica aplicada a las métricas obtenidas por la herramienta desarrollada para la estimación del tamaño funcional. Finalmente, el Capítulo 8 describe las conclusiones y el trabajo futuro.

# Capítulo 2. Marco teórico

Este capítulo describe los principales conceptos relacionados con el tema de la visualización del software, cuyo objetivo principal es facilitar la comprensión de las aplicaciones de software. A continuación se explican los conceptos relacionados con la representación del software mediante modelos de comportamiento, entendiéndose éstos como representaciones abstractas de los flujos generados por un sistema que a su vez facilitan el análisis de las aplicaciones de software [10]. Seguidamente se explica brevemente la teoría sobre la utilización de grafos [27], y finalmente se presenta la metodología IFPUG FPA [20] utilizada para medir el tamaño funcional de una aplicación de software. Este último tema es relevante ya que dicha metodología se utiliza como punto de comparación para los resultados obtenidos en la evaluación desarrollada en el Capítulo 6.

## 2.1. Visualización de los sistemas de software

En el contexto de la computación, el término "visualización" se refiere al proceso de transformar información en una forma visual, permitiendo a los usuarios observar dicha información [12]. La visualización hace uso de metáforas, las cuales pueden definirse como mapeos sistemáticos entre entidades de dos dominios conceptuales distintos [26], y cuya esencia radica en el entendimiento o representación de un tipo de objeto en términos de otro [13]. Debido a que la ciencia de la computación posee una terminología abundante en metáforas [12], la visualización tiene un amplio impacto y utilidad sobre dicha ciencia.

En el campo del software propiamente, la visualización puede entenderse como el arte y la ciencia de generar representaciones visuales de aspectos del software y de su proceso de desarrollo [12]. El objetivo principal de la visualización del software es reducir la complejidad del análisis del software [13], para ayudar a comprender las aplicaciones de software y mejorar la productividad en el proceso de desarrollo del software [12]. Según Diehl [12] el software se representa desde tres perspectivas:

(1) desde el punto de vista de la *estructura* es posible analizar la parte estática del software, lo que incluye todo aquello que puede ser inferido sin ejecutar el sistema (por ejemplo el código fuente y las estructuras de datos)

(2) desde el punto de vista del *comportamiento*, que hace referencia a la información obtenida mediante la ejecución del sistema (por ejemplo, llamados a funciones), y
(3) desde la perspectiva de la *evolución* del software como toda aquella tarea relacionada al proceso de desarrollo de éste, partiendo del hecho de que software cambia a lo largo del tiempo.

Esta investigación se enfoca en la perspectiva del comportamiento, desarrollando este tema en la Sección 2.2; sin embargo, también aprovecha aspectos del análisis estático y puede ser utilizado para analizar la evolución.

Según Gračanin y otros [13], un sistema de visualización efectivo debe considerar los siguientes atributos para clasificar visualizaciones:

(1) **Alcance de la representación**: Identificación de las características que se desean analizar en el sistema, principalmente al visualizar aplicaciones complejas que puedan generar mucha información visual. Por ejemplo, se debe especificar si se quiere visualizar información estática o dinámica.
(2) **Medio de representación**: Definido según el tipo de información a visualizar y el nivel de detalle requerido. Por ejemplo, un grafo simple en dos dimensiones o una representación más compleja en tres dimensiones.
(3) **Metáfora visual**: Las metáforas deben tener consistencia (una figura representativa debe mapearse a un único artefacto dentro del dominio), además de una adecuada riqueza semántica (proveer suficientes figuras representativas para mapear todos los posibles artefactos del dominio).
(4) **Abstracción**: Capacidad de permitir al usuario enfocarse en ciertas partes de la representación, según el objetivo del análisis. Muy comúnmente se utilizan funcionalidades como *zoom in* (acercamiento) y *zoom out* (alejamiento) para enfocarse en secciones deseadas.
(5) **Facilidad de navegación e interacción**: Capacidad de la herramienta para ayudar al usuario a entender el nivel de abstracción en el que se encuentra, permitiéndole por ejemplo cambiar fácilmente de una vista a otra. Entre otras funcionalidades importantes (según el medio de representación) se pueden mencionar la capacidad de mostrar y ocultar objetos de interés, y visualizar la información desde diferentes ángulos.

(6) **Nivel de automatización**: Se refiere al grado en el que el sistema de visualización es construido de manera automática. Un sistema efectivo requiere de una completa automatización.

Los seis atributos anteriores fueron considerados como parte de los requerimientos de la herramienta de visualización desarrollada en este proyecto (descritos en la Sección 4.7).

## 2.2. Modelos de comportamiento para representar el software

En el contexto del software muchas veces no se cuenta con la información necesaria para un adecuado entendimiento de su funcionalidad. La ***ingeniería inversa de software*** es el proceso mediante el cual se obtienen artefactos de una etapa de desarrollo previa a partir de los artefactos de una etapa de desarrollo posterior. Esto permite que cuando no se cuenta con un insumo más que el código fuente, este proceso se convierte en una alternativa para la construcción de modelos que permitan tener una visión clara y, por ende, una mejor comprensión del software. Mediante la aplicación de la ingeniería inversa a los artefactos de software es posible obtener distintas perspectivas del alcance y las formas de operación de una aplicación de software.

Por ejemplo, en el caso del software se puede analizar el código fuente escrito en un lenguaje de programación particular, con el fin de identificar sus componentes y crear representaciones de éste con un determinado nivel de abstracción [11]. Esta práctica toma gran relevancia durante las tareas de análisis de información con distintos propósitos, en entornos en los que los datos no son precisos, son incompletos o incluso no existen del todo. Por esta razón, la ingeniería inversa ayuda a facilitar la comprensión del software [11], y además juega un rol importante durante las actividades de mantenimiento y evolución del software, debido a que muchas veces la única información confiable de un programa se encuentra empotrada dentro de su código fuente al momento de dar mantenimiento o añadir funcionalidades [11].

Como resultado de la ingeniería inversa se obtienen los ***modelos***, que pueden definirse como la simplificación de un sistema con una meta específica [11]. Un ejemplo de modelo es un diagrama que muestra solamente los módulos de un programa y sus dependencias con el fin de entender la estructura básica del sistema.

En el contexto del software, un *comportamiento* se describe en términos de secuencias de acciones que un sistema puede ejecutar [10]. Del mismo modo, una acción se define como cualquier evento atómico del sistema que causa un cambio en el estado del programa [10].

Al combinar las definiciones de los términos *modelo* y *comportamiento*, se obtiene que un *modelo de comportamiento* es toda aquella representación o abstracción construida a partir de un conjunto de acciones que puede ejecutar un sistema de software. Un modelo se crea a partir de una implementación existente mediante el proceso de *extracción del modelo* [9], y es un requerimiento esencial del proceso que el modelo resultante sea una representación fiel del comportamiento del sistema, ya que cualquier análisis basado en un modelo incorrecto llevaría a un entendimiento erróneo del sistema [10]. Los modelos de comportamiento pueden ser extraídos complementando el análisis estático del código fuente con el análisis dinámico durante su ejecución.

Para extraer un modelo se utiliza un *meta-modelo* que especifica exactamente cuales elementos e interacciones deben ser extraídos [9]. En síntesis, el metamodelo define qué y cómo debe extraerse un modelo con un propósito específico. Además de utilizar un metamodelo o esquema [9], el proceso de extracción del modelo puede adoptar diferentes enfoques, según el nivel de abstracción del modelo [9]. El *enfoque estático* crea el modelo directamente partiendo del código fuente del sistema, mientras que el *enfoque dinámico* utiliza trazas de ejecución del sistema como entrada [9].

El "Lenguaje Unificado de Modelado" (*UML* por sus singlas en inglés) presenta dos categorías de diagramas: los diagramas de estructura y los diagramas de comportamiento [16]. Los diagramas de grafos utilizados como forma de representación visual para la herramienta desarrollada en esta investigación se encuentran dentro de la categoría de los diagramas de comportamiento, ya que genera la visualización de la comunicación entre objetos o componentes utilizando mensajes secuenciados [16].

Por tanto, la ingeniería inversa hace un gran aporte al tema del mejoramiento de la comprensión del software. Sin embargo, en gran parte de la literatura estudiada se hace énfasis en la necesidad de reducir el exceso de información en los resultados generados con el fin de evitar una saturación visual. Por ejemplo, la herramienta *Program Explorer* hace uso de la técnica *Pruning* (poda) [6] que se encarga de remover objetos no deseados

para enfocarse en ciertas secciones bajo un criterio específico. Al ser un tema recurrente entre los diferentes modelos estudiados, se identifica este aspecto como una limitación de los modelos creados a partir de la ingeniería inversa, y debe ser considerado al momento de crear una herramienta de visualización de software.

## 2.3. Grafos

Es importante el conocimiento de los conceptos básicos relacionados a grafos citados a continuación, para el entendimiento del proceso de cálculo de las métricas de la herramienta prototipo desarrollada en esta investigación, las cuales se basan en conteos de vértices con conexiones específicas a otros vértices (este proceso se detalla en el Capítulo 5).

Según la teoría, muchas situaciones de mundo real pueden convenientemente ser descritas mediante un diagrama que consta de un conjunto de puntos acompañados de líneas, las cuales unen ciertos pares de dichos puntos. Estas situaciones son comunes cuando lo que se quiere es establecer algún tipo de relación o comunicación entre dos o más elementos. La abstracción matemática de esas situaciones descritas anteriormente es la que da cabida al concepto de "grafo" [27]. De esta forma, un grafo G se define como un conjunto ordenado (V(G), E(G), $\psi_G$) en el cual:

- V(G) = conjunto de vértices o nodos.
- E(G) = conjunto de aristas.
- $\psi_G$ = Función de incidencia que asocia cada arista a un par de nodos (no necesariamente distintos).

La palabra "grafo" se origina en el hecho de que puede ser representado gráficamente, y es esa representación gráfica la que ayuda al entendimiento de sus propiedades. Así es posible identificar "caminos" dentro de un grafo, los cuales hacen referencia a secuencias fintas, no nulas, de vértices y aristas alternantes que poseen un vértice origen y un vértice término. Es correcto afirmar que dos nodos están conectados en un grafo si existe al menos un camino entre éstos [27].

En un grafo pueden identificarse múltiples conjuntos de nodos conectados entre sí, los cuales se denominan "subgrafos". Un grafo H es subgrafo de G (denotado $H \subseteq G$) si $V(H) \subseteq V(G)$ y $E(H) \subseteq E(G)$. Esto significa que todos los nodos y aristas del subgrafo H deben estar contenidos dentro del conjunto de nodos y aristas del grafo G [27].

## 2.4. Metodología IFPUG FPA

Esta metodología se utiliza para medir el tamaño funcional del software, independientemente de la tecnología implementada para su desarrollo, y fue creada originalmente por Allan Albrecht en 1979. El conteo de puntos de función es una de las principales métricas de tamaño funcional en la industria. Se definen los siguientes tipos de componentes funcionales utilizados para realizar dicho conteo [20]:

- Funciones de datos (DF)
    - ILF (*Internal Logical Files*): se refieren a cualquier archivo o tabla interna de la base de datos de la aplicación a medir.
    - EIF (*External Interface FIles*): grupo de datos lógicamente relacionados, ubicados fuera de los límites de la aplicación.
- Funciones transaccionales (TF)
    - EI (*External Inputs*): entrada por pantallas que provee datos a la aplicación.
    - EO (*External Outputs*): cualquier salida en pantalla o reporte. Incluye algún tipo de cálculo como parte del procesamiento de datos
    - EQ (*External Inquiries*): cualquier salida en pantalla o reporte que no incluye ningún tipo de cálculo.

A continuación se enumeran los pasos a seguir para realizar el conteo de puntos de función [20]:

1. Determinar el tipo de conteo y reunir la documentación disponible.
2. Identificar el alcance del conteo y los límites de la aplicación, estableciendo la funcionalidad a considerar para tal efecto.
3. Identificar y contar las funciones de datos.

4. Identificar y contar las funciones transaccionales.
5. Realizar el conteo bruto de puntos de función y calcular el tamaño funcional sin ajustar.
6. Determinar el factor de ajuste de acuerdo con las características generales del sistema.
7. Calcular el conteo del tamaño funcional ajustado.

Este capítulo cubrió los principales conceptos que sustentan la base teórica de esta investigación. Cabe mencionar que, ante la falta de teoría relacionada a la aplicación de grafos para la generación de métricas de tamaño funcional del software específicamente, la metodología aplicada en esta investigación (detallada en el Capítulo 3) busca probar si los grafos pueden utilizarse para realizar dichas estimaciones de forma correcta.

# Capítulo 3. Metodología de investigación

La metodología aplicada para esta investigación se resume en la Figura 2, y es adaptada de la metodología "Ciencia del diseño para sistemas de información e ingeniería del software" [25]. La primera columna de la figura hace referencia a los objetivos de investigación, y para cada objetivo se mapearon los métodos para cumplirlo en la segunda columna. Asimismo, se detallan los productos de los resultados obtenidos en la tercera columna para cada una de las actividades realizadas. Finalmente se especifica el número del capítulo de este documento en el que se ubica el detalle de cada uno de los productos.

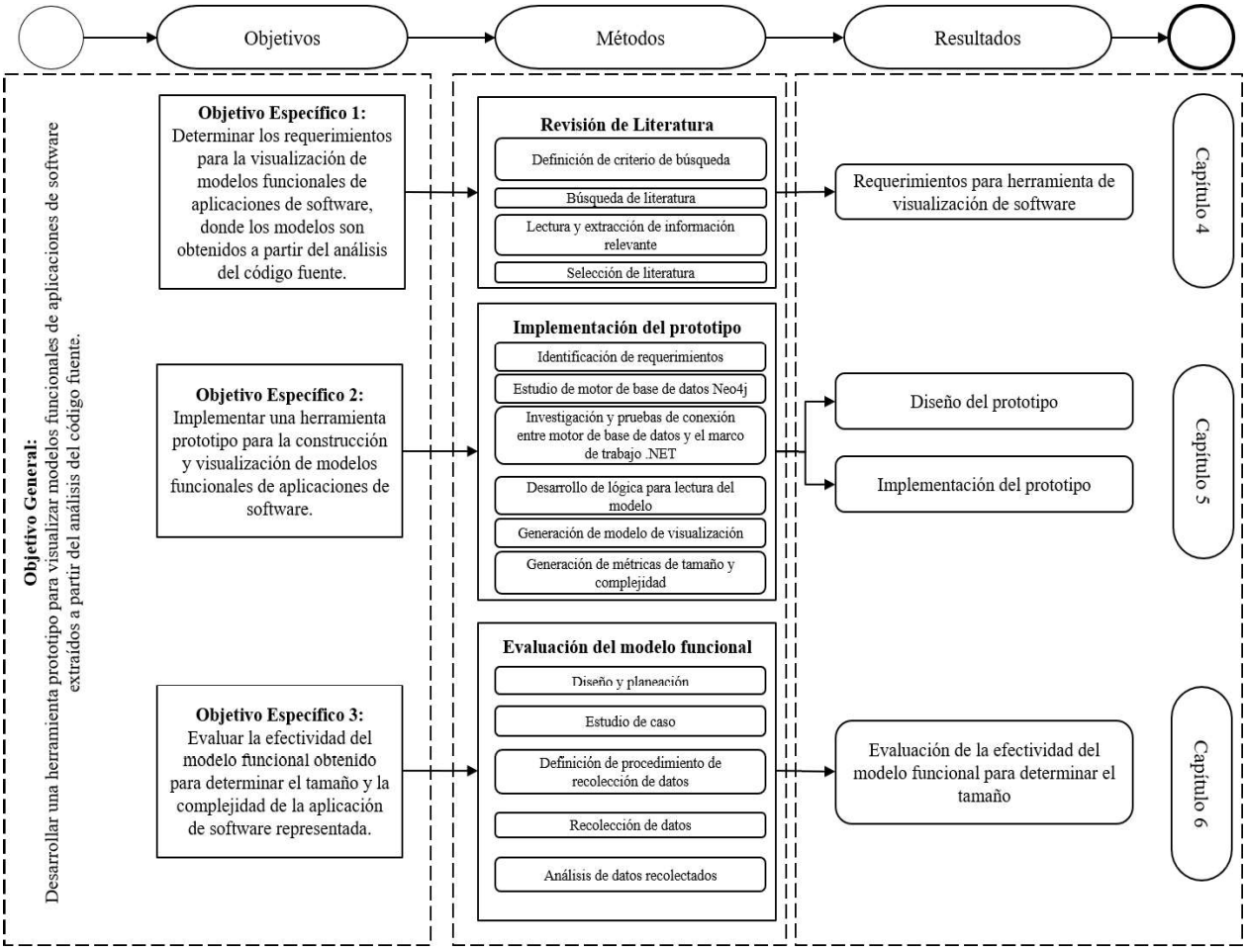



*Figura 2. Resumen de metodología de investigación.*

A continuación se detallan las etapas para cumplir con cada uno de los objetivos específicos planteados:

## 3.1. Revisión de literatura

En esta etapa se realiza una revisión de literatura para identificar modelos de visualización de software. Para realizar el proceso de identificación de artículos relevantes se siguen algunos de los principios simplificados de la metodología de revisión sistemática de literatura presentada por Wohlin et al [8], que consta de los siguientes pasos:

1. Definición de los criterios de búsqueda de la literatura a analizar.
2. Búsqueda de literatura que se ajustara a los criterios definidos.
3. Lectura y extracción de información relevante de la literatura.
4. Selección de la literatura a utilizar.

Para identificar la literatura a considerar en la investigación, primeramente se incluyeron en la búsqueda investigaciones en las que se planteara la visualización como una alternativa viable para mejorar el entendimiento del software. Bajo esta premisa se analizaron documentos en los que se presentara algún tipo de visualización de software, sin importar el ámbito o naturaleza de la información de entrada y salida. Dentro de este subconjunto de documentos, se filtraron artículos en los que se describiera el proceso del desarrollo de alguna herramienta de visualización de software que fuera capaz de analizar un sistema de manera dinámica o estática [9]. Este filtro facilitaría la obtención de prácticas recomendadas para la especificación de requerimientos necesarios para crear una herramienta de visualización. Además, como último criterio de selección se consideraron aquellas investigaciones en las que se utilizan los principios de los modelos de comportamiento [10] en el estudio, y por consiguiente en la herramienta de visualización presentada en las mismas. Con este criterio se buscó obtener información sobre tipos de métricas utilizadas.

El propósito de esta revisión es la identificación de herramientas de visualización que permitan la extracción de sus características principales que serán consideradas en el desarrollo de la herramienta prototipo propuesta en esta investigación. Se desea identificar para cada uno de los artículos seleccionados la siguiente información relevante:

- El proceso de construcción del modelo.

- Los artefactos base del modelo (requerimientos, entradas, salidas, flujos de datos).
- La metodología de evaluación.
- Las métricas utilizadas para evaluar el modelo.

## 3.2. Implementación del prototipo

En esta etapa se llevó a cabo el desarrollo de la herramienta prototipo basado en los requerimientos de visualización obtenidos a partir del primer objetivo. Se utilizó una metodología de desarrollo ágil en la que se identificaron las etapas del desarrollo y los artefactos resultantes esperados para cada una. Los métodos más importantes según su orden cronológico fueron:

1. Identificación de requerimientos para visualización del modelo a partir del análisis resultante del primer objetivo.
2. Estudio de motor de base de datos Neo4j para manejo de grafos (descrito en el Capítulo 5 de este documento).
3. Investigación y pruebas de conexión entre motor de base de datos y el marco de trabajo .NET utilizado para desarrollar la herramienta.
4. Desarrollo de lógica para lectura del modelo.
5. Generación de modelo de visualización (grafo).
6. Generación de métricas de completitud

## 3.3. Evaluación del modelo funcional

Se realizó una evaluación empírica de las métricas obtenidas por la herramienta a partir del modelo funcional, con el fin de determinar la efectividad de estas métricas para la estimación del tamaño funcional y la complejidad de un conjunto de aplicaciones de software bajo análisis. Para efectuar dicho estudio se compararon las mediciones funcionales de distintas aplicaciones de referencia, con las estimaciones obtenidas con las métricas obtenidas mediante el prototipo. Se utilizó la metodología de casos de estudio propuesta por [8], la cual consta de las siguientes etapas con sus respectivas tareas:

- Descripción.
    - Definición del problema

    - Objetivo de investigación
    - Preguntas de investigación
- Diseño.
    - Selección de tipo y objeto de investigación
    - Procedimiento de recolección de datos
    - Procedimiento de análisis de datos
    - Amenazas a la validez
- Resultados.

En los siguientes capítulos se detallan los productos y resultados para cada una de las etapas de la metodología. El Capítulo 4 corresponde a la identificación de los requerimientos a partir de los trabajos relacionados identificados, el Capítulo 5 detalla el proceso de implementación de la herramienta prototipo, y el Capítulo 6 presenta la evaluación de la efectividad de las métricas del modelo para la estimación del tamaño y complejidad.

# Capítulo 4. Trabajo relacionado

Esta sección presenta la literatura relacionada con las herramientas de visualización de software. Se extraen los aspectos más relevantes de cada uno de los enfoques expuestos en la literatura revisada, los cuales conformaron la base para la construcción del prototipo propuesto en este trabajo. Los artículos de visualización están ordenados según su fecha de publicación de forma ascendente. Finalmente, con base en las características de las herramientas propuestas en la literatura se determinan los requerimientos para desarrollar una herramienta de visualización de software.

## 4.1. Herramientas de visualización de software

Los artículos de investigación analizados se listan en el siguiente cuadro, donde se especifica el año de la publicación, el título, y la definición de la visualización.

*Cuadro 1. Herramientas de Visualización.*

| Año | Título | Herramienta | Definición |
|---|---|---|---|
| 1997 | *Object-Oriented Program Tracing and Visualization [6]* | *Program Explorer* | Permite al usuario explorar en detalle el flujo de ejecución de un sistema de software orientado a objetos, utilizando como base las invocaciones de sus métodos y el comportamiento de sus instancias de clase. |
| 2002 | *Visualizing the Execution of Java Programs [7]* | *Jinsight* | Representa visualmente la ejecución de un sistema desarrollado en lenguaje Java. Utiliza técnicas como la extracción de patrones, navegación interactiva, y trazas orientadas a tareas, con el objetivo de minimizar la sobrecarga de información usualmente inherente a las herramientas de análisis de rendimiento |
| 2015 | *Comparing Trace Visualizations for Program Comprehension through Controlled Experiments [14]* | *Extravis y ExplorViz* | Extravis: se enfoca en la visualización de trazas de ejecución considerablemente grandes, para lo cual utiliza dos vistas enlazadas e interactivas ("*Circular Bundle"* y "*Massive Sequence"*).<br>ExplorViz: visualiza la comunicación entre un conjunto de sistemas de software, mostrando al usuario detalles de la interacción de las aplicaciones bajo demanda. Del mismo modo la herramienta posee una perspectiva a nivel de aplicación que muestra la interacción entre los componentes de un único sistema, en lugar de múltiples sistemas. Esta última modalidad fue la utilizada por los autores de la publicación para el experimento controlado. |

| Año | Título | Herramienta | Definición |
|---|---|---|---|
| 2017 | *GETAVIZ: Generating Structural, Behavioral, and Evolutionary Views of Software Systems for Empirical Evaluation [17]* | *GETAVIZ* | Conjunto de herramientas integradas que brindan la funcionalidad necesaria para diseñar, modelar y adaptar diferentes tipos de visualizaciones de software, de manera que puedan ser evaluadas bajo las mismas condiciones de usabilidad |
| 2019 | *GoCity: Code City for Go [18]* | *GoCity* | Implementa la "metáfora de la ciudad" [15] para visualizar proyectos desarrollados en lenguaje *Go* como si fueran ciudades, donde cada componente es representado mediante un edificio. |

Las secciones siguientes detallan cada una de las herramientas de visualización del Cuadro 1.

## 4.2. Program Explorer

Danny y Yuichi crearon *Program Explorer* (Figura 3) que permite al usuario visualizar la ejecución de un sistema de software basándose en las invocaciones de sus métodos y otros indicadores como la interacción entre los objetos [6]. A continuación se cita una breve descripción del modelo.

- **Requerimientos**:
    - Crear una herramienta útil para apoyar tareas propias de la ingeniería de software tales como la reutilización de código, el análisis de las funcionalidades de un sistema para su mantenimiento, y la depuración visual del código fuente.
    - Facilitar el análisis cualitativo de un sistema mediante la visualización de componentes, como una alternativa a los análisis cuantitativos que se obtienen mediante información numérica o tabular.
    - Utilizar como datos de entrada no solamente información estática (definición de clases, objetos, métodos) sino también información extraída en tiempo de ejecución para generar el modelo de visualización.
- **Definición**: Herramienta que permite al usuario explorar en detalle el flujo de ejecución de un sistema de software orientado a objetos, utilizando como base las invocaciones de sus métodos y el comportamiento de sus instancias de clase.

- **Descripción**: Mediante la implementación de métodos de depuración, la aplicación monitorea el ciclo de vida de los objetos del sistema en análisis (desde su creación hasta su destrucción), al mismo tiempo que captura el flujo de activación y desactivación de las funciones en las que son partícipes. Como resultado muestra al usuario una interfaz gráfica con la posibilidad de seleccionar un subconjunto de objetos a analizar entre el total de nodos, permitiendo enfocarse en la información detallada de los elementos de interés. Por ejemplo, la Figura 3 muestra la interacción entre los objetos contenidos en un grupo previamente seleccionado por el usuario. Se observa como los objetos de diferentes clases (representados por los nodos en el grafo) invocan métodos de otros objetos (representados por las flechas).

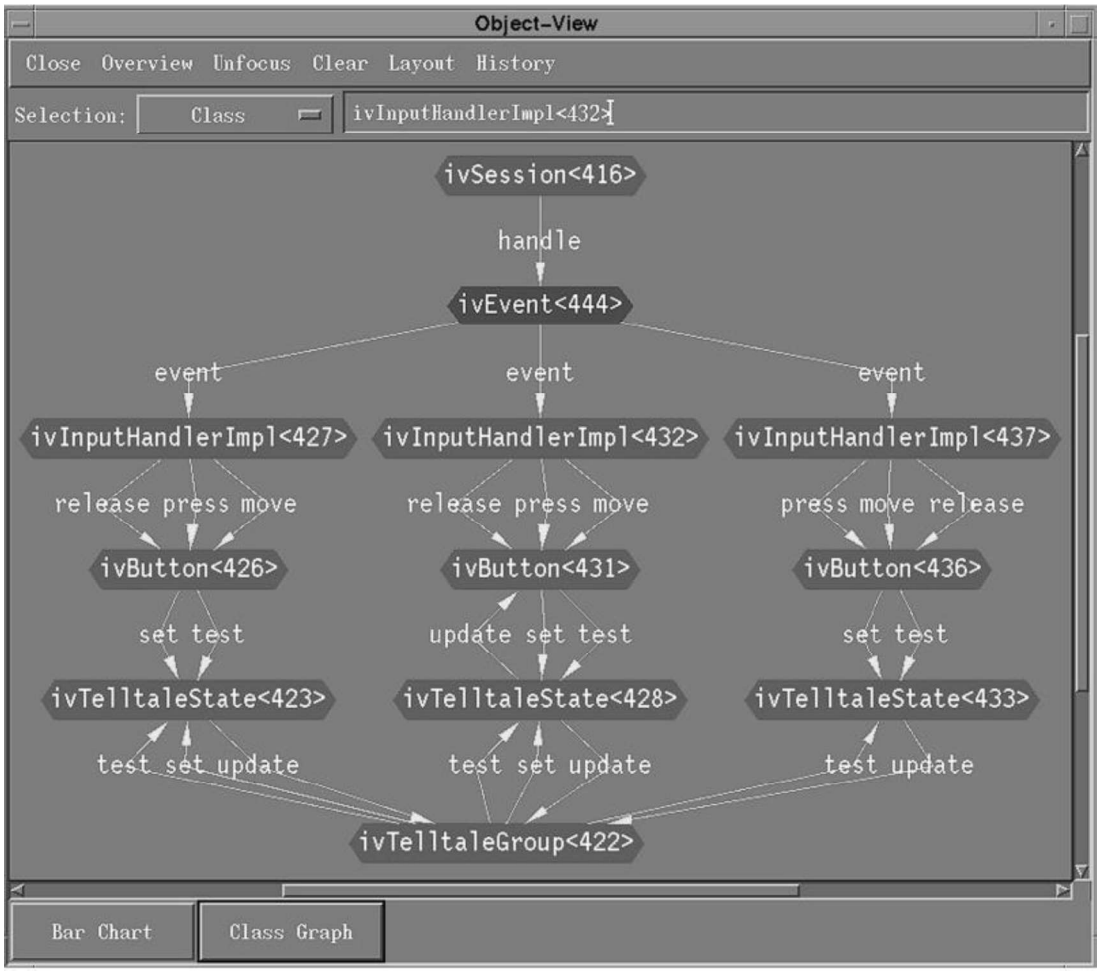



*Figura 3. Captura de pantalla de la herramienta Program Explorer que muestra la interacción entre los objetos mediante sus invocaciones [6]*

Para reducir la cantidad de información resultante o "ruido" *Program Explorer* aplica tres técnicas:

- **Combinación (*Merging*)**: Se agrupan los objetos de la misma clase, tipificando su comportamiento y resumiendo la información de naturaleza similar que se mostrará en el resultado.
- **Poda (Pruning)**: Se descartan ciertos elementos según un criterio específico definido, por ejemplo, según la jerarquía de herencia o la clase a la que pertenecen.
- **Recorte (*Slicing*)**: Se generan los resultados tomando como partida un subconjunto o sección del modelo, evitando así la saturación visual.

- **Métricas**: Dada la naturaleza de los resultados visuales de la herramienta, es posible la identificación de patrones de diseño de software existentes en la estructura del sistema en análisis; sin embargo, los autores no exponen ningún tipo de métrica cuantificable.
- **Validación**: La herramienta se ejecutó sobre un programa real fuera de la compañía IBM, la cual fue su lugar de desarrollo. Además, fue implementado también para analizar un sistema interno de IBM llamado "Interviews", en el cual se analiza la interacción de un botón específico de la interfaz con algunos métodos y se identifican los patrones de diseño *Observer, Command* y *Mediator.*

## 4.3. Jinsight

Wim De Pauw et al plantean que un modelo visual es una mejor alternativa para el entendimiento de un sistema de software, por encima del análisis de datos textuales [7]. Presentan Jinsight para facilitar el entendimiento de un determinado comportamiento complejo mediante la construcción de un modelo visual adecuado, y permitir al usuario la exploración del comportamiento del programa en ejecución de manera visual, brindando mayor facilidad en tareas como el análisis de rendimiento, la depuración y la optimización.

- **Definición**: *Jinsight* que representa visualmente la ejecución de un sistema desarrollado en lenguaje Java. Utiliza técnicas como la extracción de patrones, navegación interactiva, y trazas orientadas a tareas, con el objetivo de minimizar la

sobrecarga de información usualmente inherente a las herramientas de análisis de rendimiento.

- **Descripción**: *Jinsight* [7] muestra el flujo de ejecución en tiempo real de un sistema de software (Figura 4). Expone elementos presentes en la ejecución tales como la interacción entre hilos, cuellos de botella y la actividad del recolector de basura. Esto es posible gracias a los diferentes tipos de vistas que ofrece, cada una orientada a aspectos específicos del comportamiento de un programa desarrollado en lenguaje Java, dando al usuario la posibilidad de analizar la información desde distintas perspectivas, y representando así una fuente de datos más diversa y completa que si se analizaran datos bajo un único criterio definido. *Jinsight* aplica ciertas técnicas de simplificación de resultados como por ejemplo la extracción de patrones, la cual consiste en no desplegar cada instancia, sino una consolidación por clase que muestra las interacciones entre las mismas como se ilustra en la vista denominada "Patrón de Referencia" ilustrada en la Figura 4.

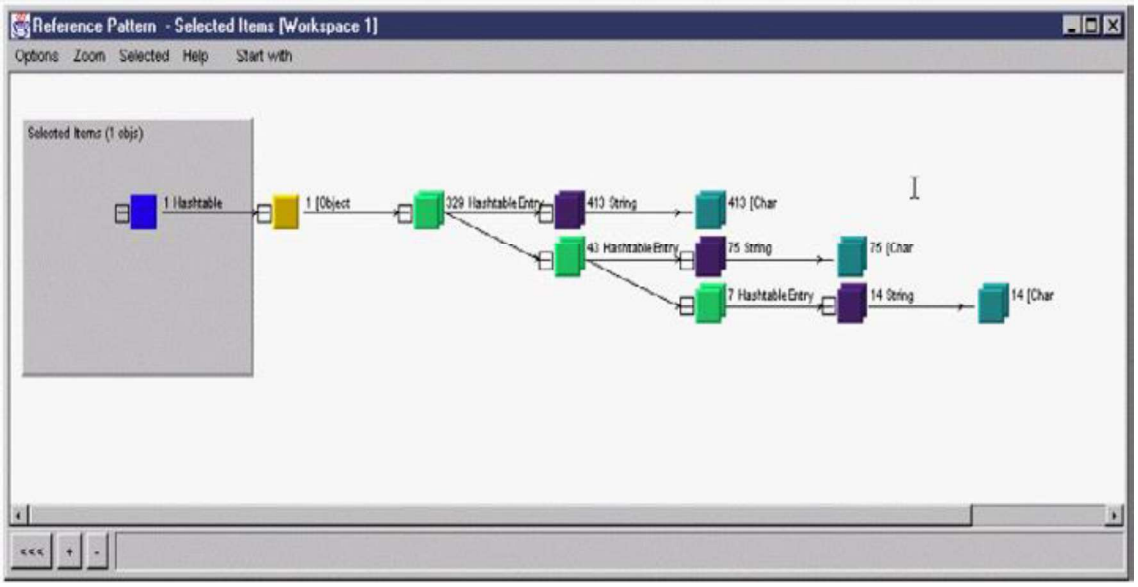



*Figura 4. Vista "Patrón de Referencia" de la herramienta Jinsight [7]*

Otra de las posibles vistas a seleccionar es la "Vista de Histograma", la cual muestra las instancias de cada clase creadas a lo largo de la traza de ejecución, así como la interacción entre las mismas. En la Figura 5 se observa las invocaciones a métodos de objetos de la clase "java/Lang/Integer" como líneas verdes sobre los rectángulos

azules; así mismo se puede observar el momento en que una instancia es recogida por el recolector de basura (representada mediante un rectángulo vacío).

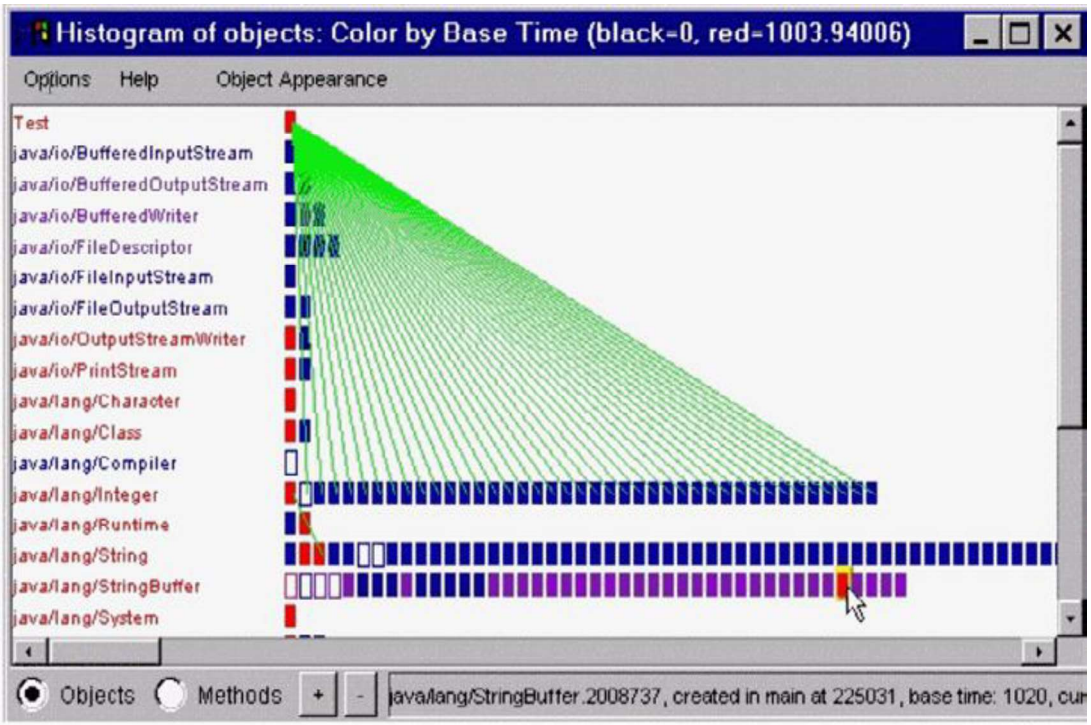



*Figura 5. Vista "Histograma" de la herramienta Jinsight [7]*

Cabe destacar que los autores consideran que la vista mostrada en la Figura 5 puede hacer que cualquier programa más grande que "*Hello World*" sea difícil de examinar debido a la naturaleza combinatoria de las relaciones entre los objetos [7].

- **Métricas**: Este modelo arroja resultados por tipo de clase como el tiempo invertido por sus métodos ejecutados, el número de invocaciones, la cantidad de memoria consumida y el número de hilos en los cuales participan sus instancias.
- **Validación**: La herramienta fue desarrollada en un ambiente de investigación industrial, lo que permitió a los creadores tener acceso a clientes reales que aportaron los prototipos de sus aplicaciones para su análisis. El ciclo de desarrollo abarcó tres etapas, sumando en total 12 meses. Durante este tiempo *Jinsight* [7] fue probado utilizando como entrada cada uno de los prototipos con casos de uso de máxima exigencia. Seguidamente se lanzó un prototipo dentro de la red interna de IBM con el objetivo de obtener retroalimentación de los compañeros de los desarrolladores.

Finalmente se liberó una versión definitiva en el sitio web www.alphaworks.ibm.com para su consumo general.

### 4.4. Extravis y ExploreViz

Los autores de la publicación aseguran que el contar con una herramienta de visualización de trazas de ejecución puede proveer beneficios con respecto al tiempo y la correctitud, por encima de realizar solamente un análisis estático en tareas típicas de comprensión del software [14]. Por tal motivo, realizan un experimento controlado en el que buscan evidenciar tales beneficios, y además enumerar los criterios necesarios para determinar si una herramienta de visualización es mejor que otra; en este caso comparando las dos herramientas descritas más adelante en esta sección.

- **Objetivos**:
  - Contribuir con un diseño reutilizable de dos experimentos controlados para comparar las herramientas de visualización de trazas de ejecución *Extravis* y *ExplorViz*.
  - Comprobar si la herramienta *Extravis* ofrece una solución más eficiente y efectiva en tareas relacionadas a la comprensión de un sistema de software, en comparación con otras.
  - Identificar los retos comunes al momento de comprar herramientas de visualización de trazas de ejecución.
- **Definición de Extravis**: Herramienta que se enfoca en la visualización de trazas de ejecución considerablemente grandes, para lo cual utiliza dos vistas enlazadas e interactivas (“*Circular Bundle”* y “*Massive Sequence”*). La Figura 6 muestra la primera vista.

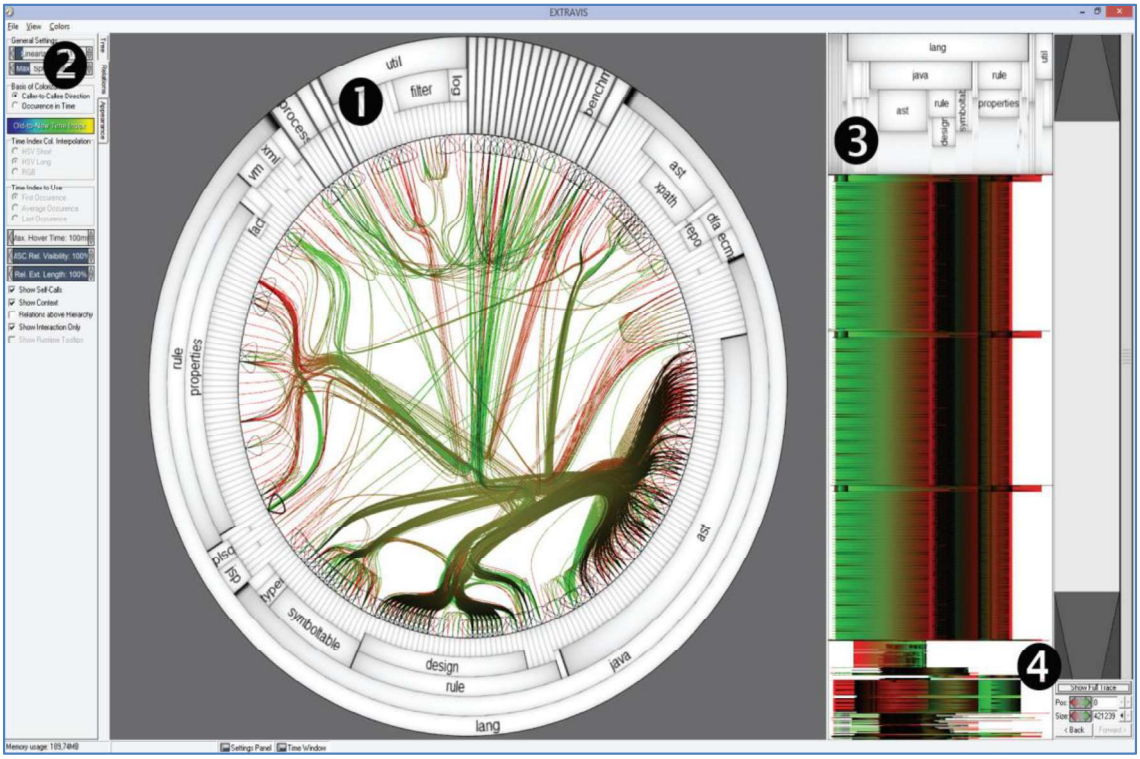



*Figura 6. Visualización de la herramienta Extravis [14]*

- **Descripción de Extravis**: Como se mencionó anteriormente, este modelo consta de dos vistas principales para mostrar al usuario la información requerida. Vista "*Circular Bundle*": la Figura 6 muestra las clases representadas mediante ① en el círculo más interno. Debido al número de clases, sus nombres solamente son visibles mediante *tooltip* al señalarlas con el cursor (por cuestiones de espacio). Los círculos externos representan los paquetes, y las líneas salientes de cada clase corresponden a los llamados de los métodos (el nombre de cada método es visible mediante *tooltip*s de forma similar a los nombres de las clases). Los colores de las líneas representan la dirección de la comunicación entre los métodos, siendo el extremo verde la clase de donde sale el llamado al método y el extremo rojo la clase que recibe el llamado. La interfaz permite ocultar paquetes de manera que las clases contenidas son eliminadas de la vista; además es posible filtrar la comunicación entre dos clases señalando ambas en el círculo interno, lo que genera que se resalten los llamados a sus métodos en la vista "*Massive Sequence*" (detallada a continuación y rotulada en la Figura 6 con ③). Finalmente, la herramienta permite visualizar el orden cronológico de los llamados a

los métodos cambiando el esquema de representación de los colores (punto ②). Dicha acción genera un cambio en el color de las líneas, mostrando las invocaciones más antiguas de color azul oscuro, y las más recientes de amarillo.

Vista "*Massive Sequence*": ilustra los llamados a los métodos con respecto al tiempo en que ocurrieron, similar a un diagrama de secuencia UML comprimido. Permite filtrar los llamados que se dieron en una ventana de tiempo específica dentro de la traza representada, mostrando en la sección señalada en la Figura 6 los métodos invocados durante ese tiempo en las clases y paquetes enumerados en la parte superior de la vista. Es posible además obtener el historial de los fragmentos de tiempo previamente visualizados en la Figura 6.

- **Definición de ExplorViz:** Aplicativo desarrollado para visualizar la comunicación entre un conjunto de sistemas de software, que muestra al usuario detalles de la interacción de las aplicaciones bajo demanda (Figura 7). Adicionalmente la herramienta posee una perspectiva que muestra la interacción entre los componentes de una única aplicación, en lugar de múltiples aplicaciones. Esta última modalidad fue la utilizada por los autores de la publicación para el experimento controlado.

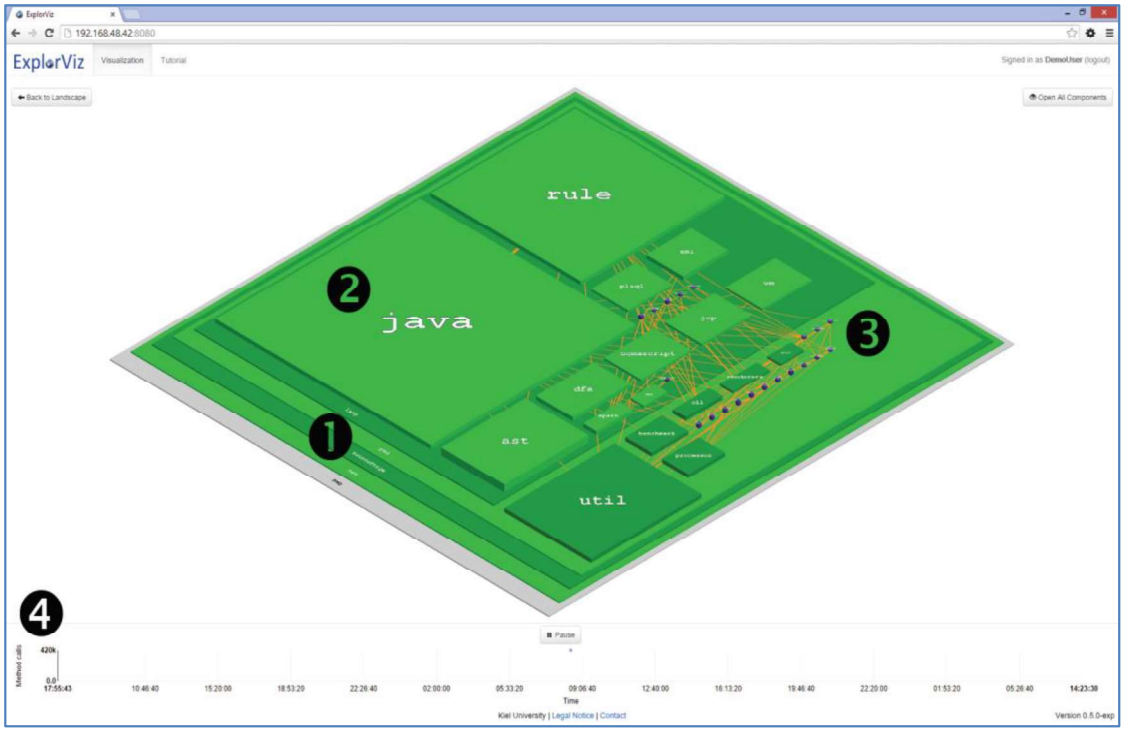



*Figura 7. Visualización de la herramienta ExplorViz.[14]*

- **Descripción de ExplorViz:** Este modelo utiliza la "metáfora de ciudad" [15] para la visualización de los componentes del software en análisis. Las cajas planas verdes señaladas con ① en la Figura 7 representan los paquetes que muestran los elementos contenidos dentro de ellos. Las cajas en la capa superior (② de la Figura 7) son paquetes cuyos elementos contenido están ocultos. Los paquetes pueden abrirse o cerrarse interactivamente. Las cajas moradas representan las clases y las líneas naranja ilustran la comunicación entre componentes. La altura de una clase es proporcional al número de instancias activas de la misma. La herramienta permite al usuario explorar las relaciones entre clases específicas al seleccionarlas; de forma homóloga a la funcionalidad de la herramienta *Extravis* explicada anteriormente.

  Es importante mencionar que existen varias funcionalidades de *ExplorViz* que fueron deshabilitadas por los autores de la publicación, ya que no existen funcionalidades homólogas en *Extravis* para su comparación. Algunas de éstas son la capacidad de visualizar código fuente de una clase al hacer clic sobre la misma; y la posibilidad de moverse a través del tiempo a lo largo de la traza (④ en la Figura 7).
- **Métricas**: El objetivo principal de la publicación consistió en realizar una comparación entre las dos herramientas mencionadas, las cuáles por sí mismas no extraen ningún dato cuantificable o métrica, pero fueron sujetas de un experimento cuyo resultado se basó en dos métricas principales: el **tiempo** tomado por las personas sujetas al experimento para completar un problema relacionado con la comprensión del software en cuestión, y el nivel de **correctitud** de las soluciones propuestas por las personas. Los autores consideraron para su estudio que ambas métricas eran buenos indicadores para dimensionar el aporte de las herramientas de visualización a la comprensión de las aplicaciones de software en análisis.
- **Evaluación**: En esencia, el experimento controlado consistió en correr las herramientas *Extravis* y *ExplorViz* para visualizar el mismo sistema de software llamado *PMD* [14]. Ambas herramientas fueron evaluadas por varias personas distribuidas aleatoriamente en dos grupos, uno grupo para cada herramienta de visualización. Individualmente, cada persona utilizó la herramienta asignada a su grupo mientras resolvió un cuestionario previamente definido con problemas que

exigieron un nivel considerable de comprensión del sistema *PMD* en análisis. De esta forma se determinó cuál de las dos herramientas ayudó al usuario a entender mejor el sistema *PMD*, tomando como indicadores el tiempo que se tardó el usuario en responder cada una de las preguntas, y el nivel de correctitud de las respuestas brindadas por el usuario. El diseño experimental se basó en estudios de diversos tipos realizados por otros autores. Se consideraron elementos como el perfil y la cantidad de las personas participantes en el experimento, la distribución de éstas entre los grupos, el tipo de preguntas a incluir en los cuestionarios, los criterios de correctitud de las respuestas, el medio por el cual se aplicarían los cuestionarios, la cantidad de estaciones de trabajo, la forma de capturar el tiempo tomado para responder las respuestas por parte de los usuarios, los tutoriales sobre cómo utilizar *Extravis* y *ExplorViz*, y demás condiciones.

El experimento se aplicó primeramente utilizando *PMD* como software de entrada, y luego se replicó utilizando una aplicación de menor tamaño llamada *Babsi* [14]. Tras extraer y promediar los datos de todas las personas, se determinó que para un sistema de mayor tamaño como *PMD*, con el modelo de visualización de *ExplorViz* las personas registraron un menor tiempo para resolver cada respuesta, y un mayor nivel de correctitud en comparación con el modelo de *Extravis*. Sin embargo, para un sistema de menor tamaño como *Babsi,* el tiempo de respuesta promedio fue similar con ambas herramientas de visualización, aunque con *ExplorViz* se obtuvo un mayor porcentaje de correctitud en las respuestas dadas por los usuarios.

## 4.5. GETAVIZ

GETAVIZ fue construido para apoyar las actividades de la ingeniería de software, considera que la naturaleza de cada tarea determina los aspectos del software que necesitan ser visualizados (estructurales, de comportamiento o de evolución) [17]. Sus autores destacan la importancia de evaluar correctamente las herramientas de visualización a utilizar según el dominio del problema, y proponen el conjunto de herramientas seguidamente detallado [17].

- **Objetivos**: Permitir a los *stakeholders* evaluar diferentes metáforas de visualización y sus variantes, para facilitar la decisión sobre el modelo a elegir más adecuado para un determinado problema de análisis.
- **Definición**: Consiste en un conjunto de herramientas integradas que brindan la funcionalidad necesaria para diseñar, modelar y adaptar diferentes tipos de visualizaciones de software, de manera que puedan ser evaluadas bajo las mismas condiciones de usabilidad (Figura 8).

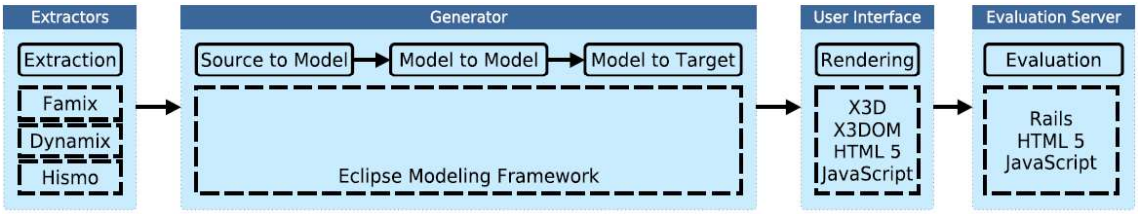



*Figura 8. Diagrama de técnicas de implementación de GETAVIZ [17]*

- **Descripción**: La Figura 8 resume la estructura de la implementación del conjunto de herramientas desarrollado. Se compone de *extractores* del modelo para diferentes lenguajes de programación, un *generador* de la visualización, una *interfaz de usuario* basada en explorador web y un *servidor de evaluación*. Se presenta a continuación una breve descripción de cada componente, sin mayor detalle técnico ya que no es el fin principal de esta investigación.

  *Extractores*: Componentes encargados de obtener el modelo de entrada. Se utilizaron los siguientes tres metamodelos, cada uno con sus respectivos extractores para diferentes lenguajes de programación: FAMIX para modelar aspectos estructurales (extrae lenguajes Java, Ruby, ABAP y C#), DYNAMIX para agregar aspectos de comportamiento (con extractores para Java y Ruby), y HISMO para modelar aspectos de evolución del software (con extractores para git y SVN).

  *Generador*: Soporta visualización en 2D y 3D de los tres aspectos mencionados anteriormente (estructura, comportamiento y evolución). Diseñado para ser flexible y extensible, de manera que sea posible agregar nuevas implementaciones de visualizaciones por parte de nuevos investigadores que quieran integrar sus modelos al proceso unificado de evaluación empírica. Soporta varias metáforas de

visualización como la **ciudad** *(City metaphor)*, el **disco recursivo** *(RD metaphor)*, la **planta** (*Plant metaphor*) y la **esfera múltiple** (*Multisphere metaphor*) [17], ilustradas en la Figura 9.

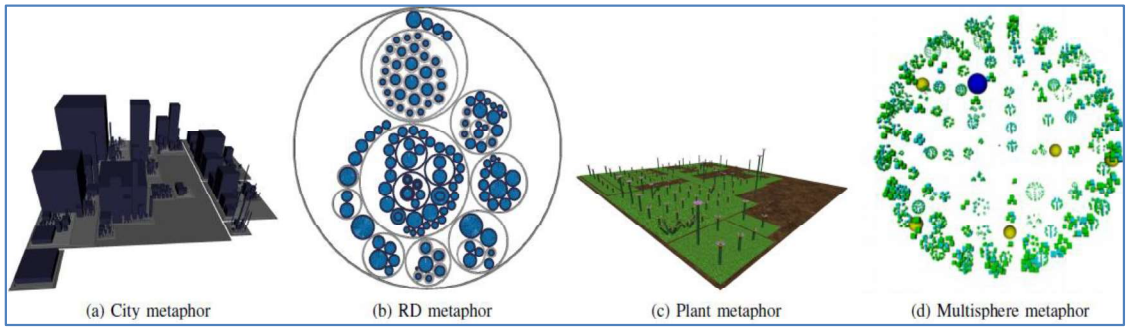



*Figura 9. Metáforas soportadas por GETAVIZ [17]*

Del mismo modo es posible visualizar diferentes variantes de la misma metáfora, por ejemplo la metáfora *RD* (observada en la Figura 10) puede mostrar una **vista estructural** en la que la altura de los elementos representa la complejidad ciclomática (cantidad de flujos distintos de ejecución), la **vista de comportamiento** que tiene la altura de los elementos como una representación de la duración de los llamados a los métodos, y una tercera **vista de evolución** en la que se visualizan diferentes versiones del sistema de software.

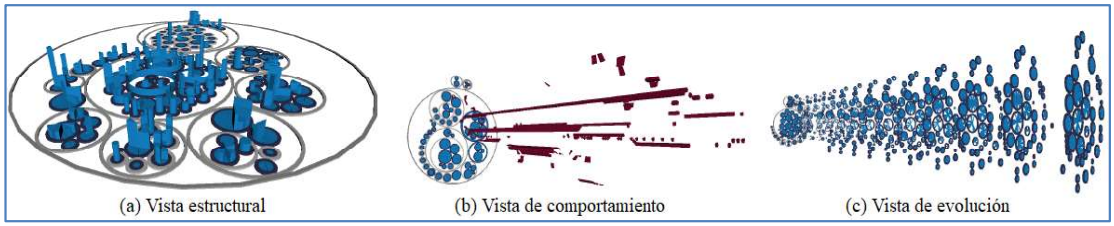



*Figura 10. Metáfora RD y sus variantes soportadas por GETAVIZ [17]*

*Interfaz de Usuario:* Basada en navegación web, la interfaz fue construida implementando HTLM5 y Javascript, y utilizando los motores X3D y X3DOM para el renderizado a partir de los modelos de visualización que fueron la salida del generador.

*Servidor de evaluación:* Consta de una aplicación *Rails* construida para facilitar el planeamiento y la conducción de evaluaciones empíricas sobre las visualizaciones de

software generadas. Provee funcionalidades que permiten definir experimentos de evaluación con sus respectivos cuestionarios, tareas de análisis, reglas, tiempos, participantes, entre otros factores. La ventaja que ofrece este componente radica en la automatización del proceso total de evaluación y su flexibilidad, reduciendo la influencia del evaluador sobre los participantes.

- **Métricas**: el servidor de evaluación permite obtener datos a partir de la interacción de los participantes con las diferentes visualizaciones. Entre estos datos se encuentran el número de clics y el tiempo que un dispositivo de apuntar (por ejemplo: el ratón) es presionado por el usuario, lo que abre muchas posibilidades para elaborar métricas relacionadas a la usabilidad de cada visualización.
- **Validación**: La publicación menciona que se han realizado varios experimentos en los que se ha utilizado GETAVIZ; sin embargo, no se muestran sus resultados en el documento.

### 4.6. GoCity

Rodrigo Brito y los otros autores consideran que al igual que el resto de sistemas de software, los proyectos desarrollados en *Go*[2] sufren de cambios para mejorar la calidad del código fuente, por lo que una parte importante de la evolución de estos proyectos se relaciona con tareas de mantenimiento y comprensión del código fuente. Tal motivo, sumado al limitado número de herramientas que apoyan a los desarrolladores de *Go* [18], es que se presenta el siguiente modelo de visualización de software.

*GoCity* se creó para ayudar a los desarrolladores de sistemas de software en lenguaje *Go* en sus tareas de mantenimiento, mostrando información relevante sobre el diseño del código fuente y la arquitectura.

- **Definición**: *GoCity* es una herramienta web que implementa la "metáfora de ciudad" [15] para visualizar proyectos desarrollados en lenguaje *Go* como si fueran ciudades, donde cada componente es representado mediante un edificio (Figura 11).

[2] Lenguaje de programación creado por Google [18]

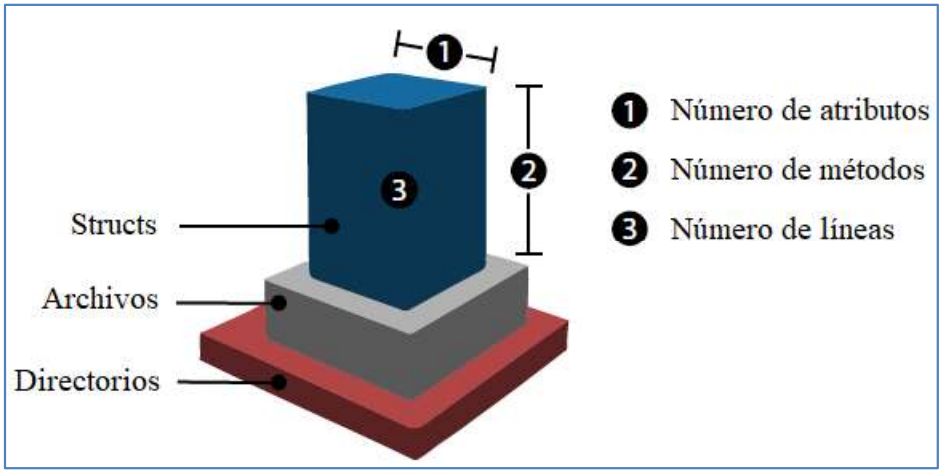



*Figura 11. Simbología de GoCity [18]*

- **Descripción**: Esta herramienta se basa en la herramienta *CodeCity* [15], adaptada a las características del lenguaje *Go*. Tiene la capacidad de extraer un proyecto desde su repositorio en GitHub y generar la visualización del sistema. Como se muestra en la Figura 11, los directorios en los cuales están contenidos los archivos del código fuente se visualizan como grandes rectángulos de color rojo. Seguidamente, los rectángulos grises representan cada archivo del código, y finalmente los rectángulos azules representan las clases (estructura de datos compuesta de atributos y métodos [18]). El ancho de cada edificio es proporcional al número de atributos que contiene, y la altura representa el número de métodos de la clase. Adicionalmente, la intensidad del color de cada estructura denota el número de líneas de código que contiene, de tal forma que entre más oscuro el color, mayor el número de líneas de código. La herramienta ofrece una opción para consultar las métricas de cada estructura, así como también obtener el código fuente en GitHub, tal y como se ilustra en la Figura 12.


*Figura 12. Visualización de métricas en GoCity [18]*

- **Métricas**: La herramienta muestra bajo demanda las siguientes métricas por cada componente: número de líneas de código, número de métodos y número de atributos. Con estos indicadores se espera que el usuario pueda tener una mejor noción sobre la forma en la que está estructurado el código fuente.
- **Evaluación:** Para evaluar la efectividad del modelo, los autores elaboraron una encuesta en la que tomaron 60 proyectos de software de GitHub (utilizando ciertos criterios de selección) y enviaron un correo a sus respectivos desarrolladores. El correo contenía un enlace (*URL*) a *GoCity* mostrando la visualización de su respectivo sistema, así como una pregunta referente a si consideraría que *GoCity* provee aspectos importantes sobre diseño y estructura, y otra pregunta sobre qué tanto puede *GoCity* ayudar en el mantenimiento y evolución de su sistema de software. Al analizar las respuestas recibidas por parte de los desarrolladores, los autores concluyeron que la herramienta puede ser utilizada para ayudar en problemas de diseño y modularización del código fuente, además de ser una guía para identificar posibles puntos de refactorización y para apoyar tareas relacionadas a la comprensión del código fuente.

## 4.7. Requerimientos de una herramienta de visualización de software.

Los requerimientos funcionales siguientes fueron identificados a partir del análisis de las herramientas y visualizaciones listadas en la sección anterior. Para esto se identificaron las coincidencias de las características funcionales de dichas herramientas de visualización, las cuales realizan lo siguiente:

- Extraen o reciben el modelo de entrada del sistema que se va a visualizar [6], [7], [14], [17], [18].
- Interpretan el modelo con base en el metamodelo [6], [7], [14], [17], [18].
- Transforman el modelo a una representación visual específica utilizando una metáfora [6], [7], [14], [17], [18].
- Ofrecen diferentes vistas o perspectivas según el tipo de información que desean mostrar, evitando saturación visual [6], [7], [14], [17].
- Permiten al usuario interactuar con los componentes del modelo visual al ejecutar diferentes acciones de navegabilidad [7], [14], [17], [18].
- Muestran datos o métricas de interés [7], [17], [18].

El Cuadro 2 enumera los principales requerimientos funcionales deseables en una herramienta de visualización de software, indicando el nombre de las herramientas analizadas en este capítulo de las cuales fue extraído cada requerimiento.

*Cuadro 2. Requerimientos de las herramientas de visualización*

| **Requerimiento** | **Herramientas** |
|---|---|
| Carga o extracción del modelo de entrada | Todas las herramientas analizadas. |
| Interpretación del modelo de entrada | Todas las herramientas analizadas. |
| Generación del modelo visual | Todas las herramientas analizadas. |
| Soporte para acciones de navegabilidad (*zoom in/out, drag and drop, tooltip*, entre otras) que permitan manipular los componentes en pantalla | Jinsight [7], Extravis [14], ExploreViz [17], GETAVIZ [17], GoCity [18]. |

| Requerimiento | Herramientas |
|---|---|
| Capacidad de mostrar diferentes vistas según el tipo de información a visualizar | Program Explorer [6], Jinsight [7], Extravis [14], ExploreViz [14], GETAVIZ [17]. |
| Capacidad de mostrar métricas al usuario | Jinsight [7], GETAVIZ [17], GoCity [18]. |

Los requerimientos identificados para una herramienta de visualización de software, a partir del análisis de las visualizaciones propuestas en la literatura, fueron la base para el desarrollo de la herramienta prototipo que se detalla en el siguiente capítulo.

# Capítulo 5. Desarrollo de la herramienta

Este capítulo detalla el diseño e implementación del prototipo de la herramienta de visualización de software desarrollada durante esta investigación, la cual considera los requerimientos identificados en el Capítulo 4. El prototipo permite mapear el modelo de una aplicación de software y generar un grafo, el cual permite obtener métricas de tamaño funcional y la visualización del modelo, permitiendo tener una perspectiva funcional de los componentes (por ejemplo: interfaces de usuario, controles, eventos, funciones, tablas) y sus interacciones. Esta herramienta genera la visualización mediante un modelo de comportamiento basado en grafos, el cual visualiza los componentes y sus relaciones, así como las métricas que permiten obtener una noción de tamaño y complejidad. El prototipo se desarrolló utilizando la plataforma .NET y el lenguaje de programación C#, en combinación con el motor de base de datos de grafos Neo4j. La Figura 13 resume el proceso que implementa la herramienta de visualización.

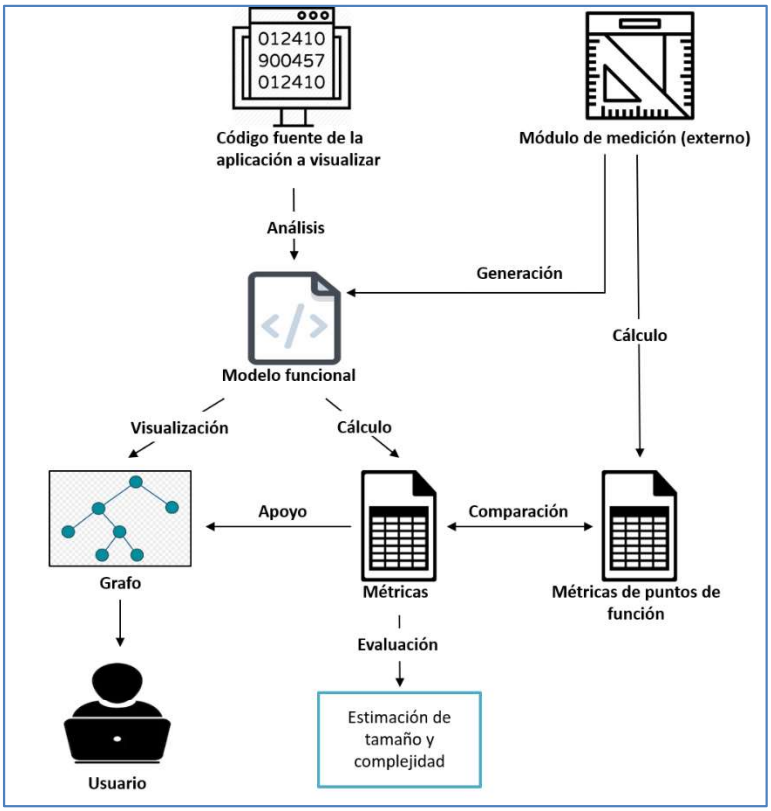



*Figura 13. Proceso de la herramienta de visualización*

Como se observa en la Figura 13, el flujo del proceso inicia mediante el análisis del código fuente de la aplicación que se va a visualizar, generando el modelo funcional. Este análisis es realizado por una herramienta existente de la metodología *aFPA* presentada en el proyecto de investigación "Medición automatizada del tamaño funcional de aplicaciones transaccionales", No. 834-B5-A18 [20]. A partir de un archivo XML que representa este modelo funcional, la herramienta prototipo construye la visualización del grafo, al mismo tiempo que calcula las métricas que cuantifican los elementos visuales que son mostrados al usuario. Estas métricas son comparadas con las métricas generadas por el módulo de medición presentado en [20] para determinar su eficacia.

## 5.1. Contexto de la herramienta

En el proyecto de investigación "Medición automatizada del tamaño funcional de aplicaciones transaccionales", No. 834-B5-A18 [20] se presenta una metodología para la automatización de la medición del tamaño funcional de aplicaciones de software (*aFPA*). Esta incluye tres herramientas desarrollados para el soporte de dicha metodología. En la Figura 14 se observa primeramente el módulo de medición del tamaño funcional, el cual se encarga de extraer los artefactos de software de la aplicación en análisis, construir el modelo funcional y aplicar las reglas de mapeo entre los elementos del modelo funcional y los elementos funcionales de los métodos de medición del tamaño funcional IFPUG FPA y COSMIC FFP. El producto de salida de dicho módulo es un reporte con los resultados de la medición del tamaño funcional, que se utiliza como entrada para el prototipo de verificación de exactitud (ubicado en la parte central). Éste se encarga de hacer una comparación entre las mediciones obtenidas y los valores de control para verificar la exactitud de las mismas. Finalmente, se muestra el módulo de evaluación de los modelos de estimación de esfuerzo, generados a partir de las mediciones del tamaño funcional.

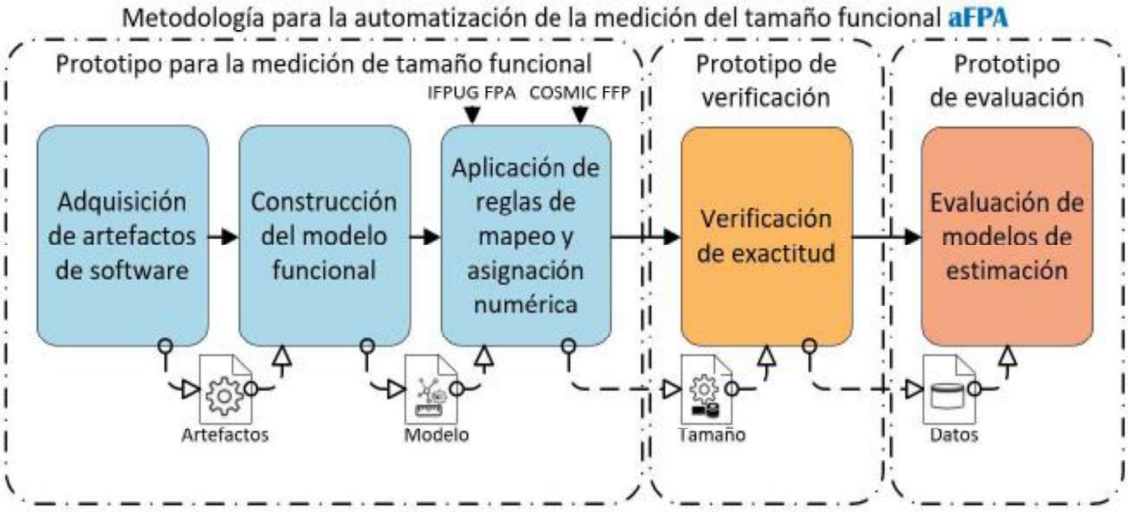



*Figura 14. Prototipos para el soporte de la metodología aFPA [20]*

La herramienta desarrollada en esta investigación apoya la metodología *aFPA* al ser un prototipo de visualización del modelo funcional construido por el módulo de medición (Figura 15).

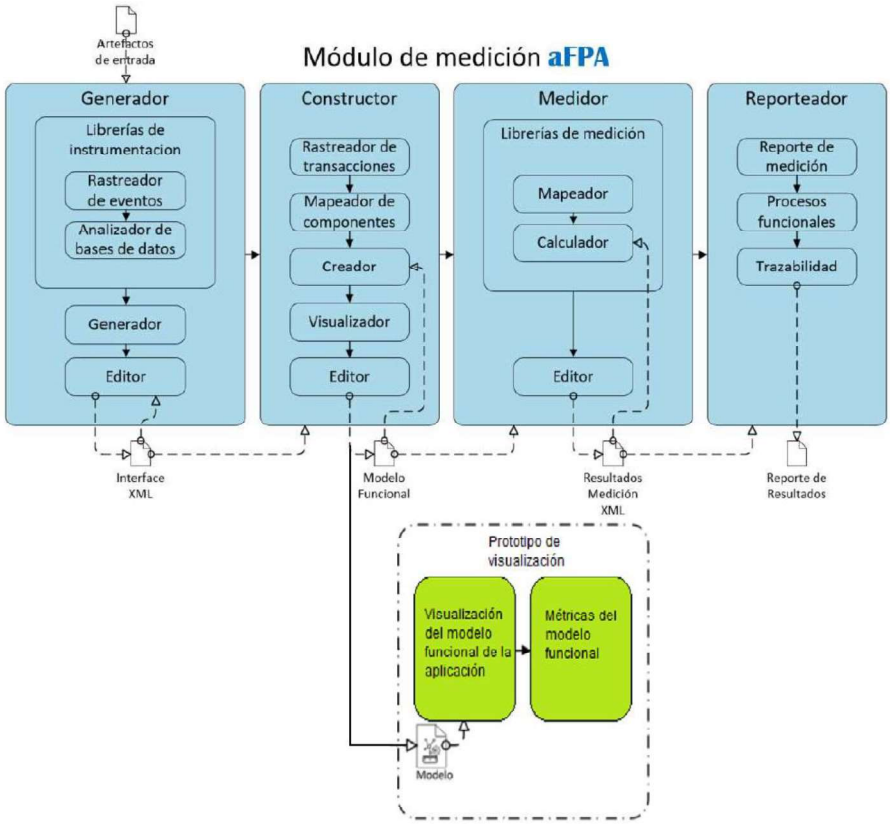



*Figura 15. Integración del prototipo de visualización para el soporte de la herramienta de medición aFPA. Adaptado de [20]*

En la parte inferior de la Figura 15 se añade el prototipo de visualización al conjunto de herramientas de la metodología *aFPA*. Se representa la relación de la herramienta desarrollada en esta investigación y el prototipo de medición, de tal forma que el modelo funcional generado por el componente *Constructor* en el módulo de medición se utiliza como entrada para el prototipo de visualización. La forma en la que el modelo funcional es transmitido de un prototipo al otro es mediante la carga en el módulo de visualización del archivo de estructurado en lenguaje *Cypher*[3], el cual es generado a partir del archivo de interfaz XML que representa los trazos funcionales de la aplicación. Este modelo es definido por el metamodelo de representación funcional del software requerido para realizar el conteo del tamaño funcional del procedimiento de medición *aFPA* [20] y mostrado en la Figura 16.

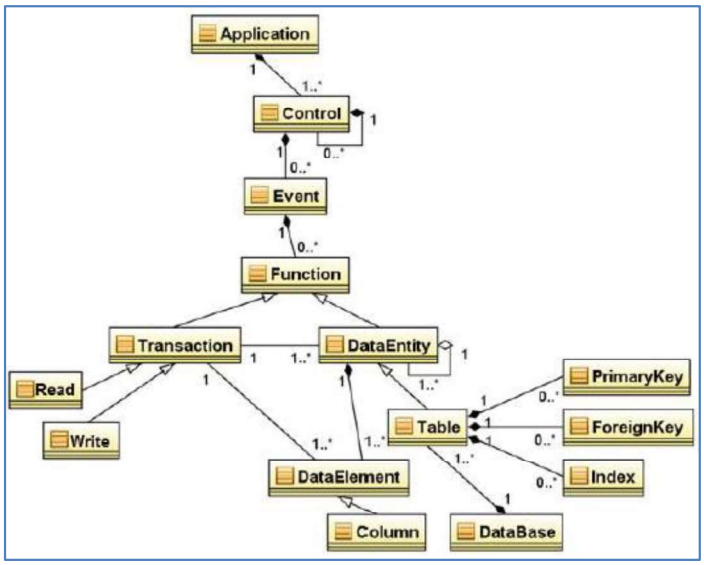



*Figura 16. Metamodelo del procedimiento aFPA utilizado como entrada para el prototipo de visualización [20]*

Los elementos del metamodelo se identifican de la siguiente forma [20]:

- *Application*: representa la aplicación de software visualizada.
- *User Interface*: se trata de un atributo del elemento *Control*, referenciando la interfaz de usuario o pantalla en la que se encuentra dentro de la aplicación visualizada.

[3] https://neo4j.com/developer/cypher/

- *Control*: corresponde a un componte de la interfaz de usuario (botones, campos de edición, entre otros).
- *Event*: acción disparada mediante un control, puede o no ejecutar funciones.
- *Function*: puede representar un proceso funcional (elemento *Procedure*), una función transaccional o de datos, o un grupo de datos.
- *Data_Base*: base de datos de la aplicación visualizada.
- Componentes relacionados a la base de datos en el modelo: *View, Index, Table, Column, Primary_Key, Foreign_Key*.

La Figura 17 muestra la relación entre los elementos del metamodelo de la metodología *aFPA* y el método de medición de tamaño funcional IFPUG FPA detallado en [20]. Este mapeo entre metamodelos se considera parte la definición de las métricas de estimación del tamaño funcional ya que deben representar dichos componentes.

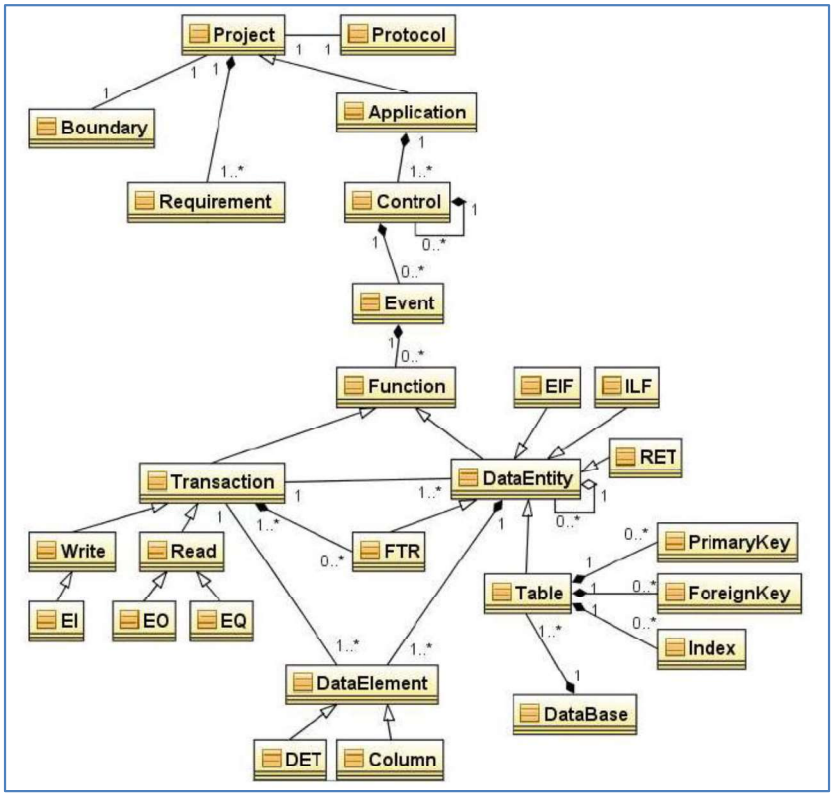



*Figura 17. Mapeo con el metamodelo IFPUG FPA [20]*

Como se observa en la Figura 17, un elemento *Function* del modelo aFPA se mapea con el concepto función transaccional (TF) de IFPUG FPA. Del mismo modo, un elemento *Transaction Read* se mapea con una salida (EO) o una consulta (EQ), un elemento *Transaction Write* se mapea con una entrada (EI). Un elemento *DataEntity* se mapea con un archivo lógico interno (ILF), interfaz externa (EIF), función de datos (DF), elemento de tipo registro (RET) o elemento de tipo archivo referenciado (FTR). Un *DataElement* se relaciona con el concepto de elemento de tipo de dato referenciado (DET) [20].

## 5.2. Especificación de requerimientos

En esta sección se listan los requerimientos funcionales definidos para el prototipo de visualización. Estos requerimientos se definieron con base en las funcionalidades básicas de una herramienta de visualización de software, citadas en la sección 4.7 de este documento. De esta forma, se define la especificación de requerimientos detallada, siguiendo las recomendaciones y el formato organizado por funcionalidad, sugerido por la IEEE [19]. Este formato enumera las funcionalidades y explica el comportamiento deseado para cada una, de manera que se agrupan los requerimientos por funcionalidad.

1. **Generar el diagrama de grafo para representar la aplicación de entrada.**
   - 1.1. Propósito: Iniciar el flujo de la aplicación.
   - 1.2. Estímulo/Respuesta: El usuario ingresa a la herramienta y presiona el botón para cargar el archivo de texto que contiene el modelo funcional de la aplicación a visualizar (modelo descrito en [20]). Se muestra en pantalla el diagrama de grafo resultante.
   - 1.3. Requerimientos funcionales asociados
     - 1.3.1. Selección del archivo de texto que corresponde al modelo funcional.
     - 1.3.2. Lectura y procesamiento del archivo de texto.
     - 1.3.3. Generación del grafo.
2. **Permitir al usuario manipular el diagrama de grafo en pantalla.**

2.1. Propósito: Ofrecer al usuario funcionalidades para recorrer el grafo y obtener mayor facilidad de análisis visual.

2.2. Estímulo/Respuesta: El usuario solicita enfocar, mostrar y ocultar secciones del grafo.

2.3. Requerimientos funcionales asociados

2.3.1. Soporte de funciones básicas de interacción con el grafo: selección y arrastre de los nodos del grafo, *zoom in/out*, mostrar propiedades de los nodos y aristas mediante *tooltip*.

2.3.2. Capacidad de mostrar una vista del grafo completo.

2.3.3. Capacidad de mostrar una vista de un subgrafo específico, seleccionando como elemento raíz un nodo que representa una "pantalla" en la interfaz de usuario de la aplicación a visualizar (según el modelo de entrada [20]).

**3. Mostrar métricas.**

3.1. Propósito: Complementar el análisis de la información presentada en pantalla mediante datos numéricos.

3.2. Estímulo/Respuesta: Se despliega en pantalla una tabla con las métricas correspondientes a cada pantalla de interfaz de usuario de la aplicación visualizada, mostrando la cantidad de funciones invocadas y la cantidad de tablas de la base de datos afectadas (según el modelo presentado por Christian Quesada [20]).

3.3. Requerimientos funcionales asociados

3.3.1. Capacidad de mostrar la cantidad de columnas afectadas por "interfaz de usuario".

3.3.2. Capacidad de mostrar la cantidad de tablas afectadas por "interfaz de usuario".

3.3.3. Capacidad de mostrar la cantidad de funciones de lectura invocadas por "interfaz de usuario".

3.3.4. Capacidad de mostrar la cantidad de funciones de escritura invocadas por "interfaz de usuario".

## 5.3. Arquitectura del prototipo

La herramienta fue desarrollada utilizando el patrón Modelo-Vista-Controlador (MVC). A continuación se explica cada uno de los componentes de la arquitectura representada en la Figura 18.

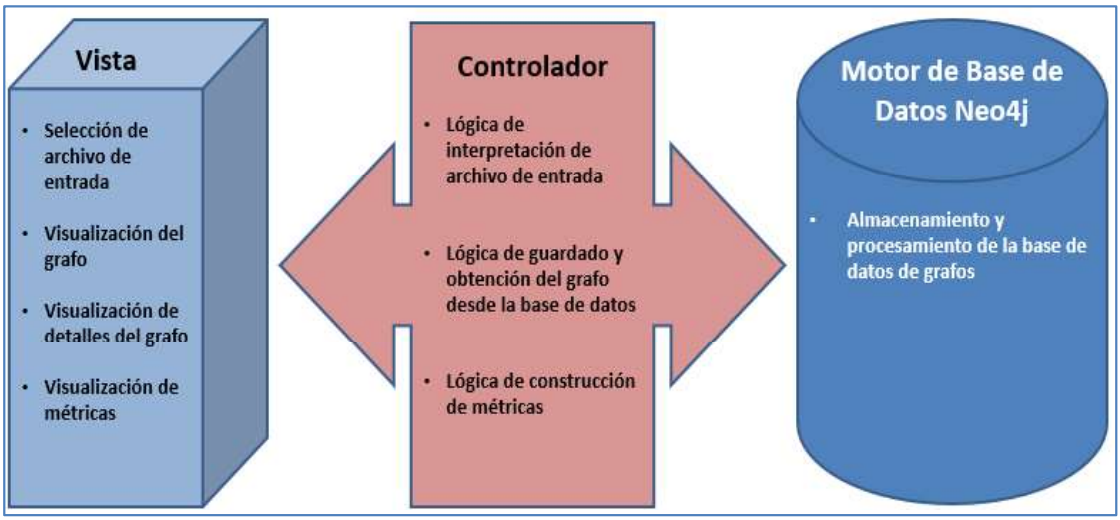



*Figura 18. Diagrama de arquitectura del prototipo de visualización de software*

- *Vista:* Interfaz de usuario de la aplicación, encargada de mostrar al usuario la visualización y las métricas. El grafo es generado mediante la biblioteca *alchemy.js*[4]. La vista consulta al controlador la información del grafo (nodos y aristas con sus respectivos atributos) y los utiliza como entrada para el componente *alchemy* que tiene la configuración visual y de usabilidad del diagrama que es desplegado pantalla. Así mismo, la vista realiza la consulta de las métricas al controlador y despliega los resultados en pantalla.
- *Motor de base de Datos Neo4j*[5]*:* Motor para almacenamiento y procesamiento de bases de datos de grafos. Utiliza el lenguaje de consulta de grafos *Cypher*.
- *Controlador:* Maneja la lógica de procesamiento para leer e interpretar el modelo de entrada, y almacenar el grafo en la base de datos. Además, consulta la información relacionada al grafo (tanto su estructura, como las métricas). Se conecta al motor de

[4] https://graphalchemist.github.io/Alchemy/#/
[5] https://neo4j.com/product/

base de datos *Neo4j* utilizando el conector *Neo4jClient*[6], el cual permite construir sentencias en lenguaje *Cypher* para comunicarse con la base de datos y así almacenar o consultar el grafo. De esta forma el controlador ejecuta la sentencia para almacenar el grafo en la base de datos, consulta la estructura del grafo almacenado para pasarlo a la vista, y obtiene las métricas que son desplegadas también mediante la vista.

## 5.4. Descripción del prototipo

Esta sección explica en detalle el funcionamiento de la herramienta de visualización. Primeramente, se describen los componentes generales de la interfaz de usuario y luego se cubre cada funcionalidad contenida en la especificación de requerimientos anteriormente detallada en la sección 5.2.

### 5.4.1. Interfaz de usuario

En la Figura 19 se muestra la pantalla principal del sistema, en la cual se identifican con los números del ① al ④ cada una de las principales secciones.

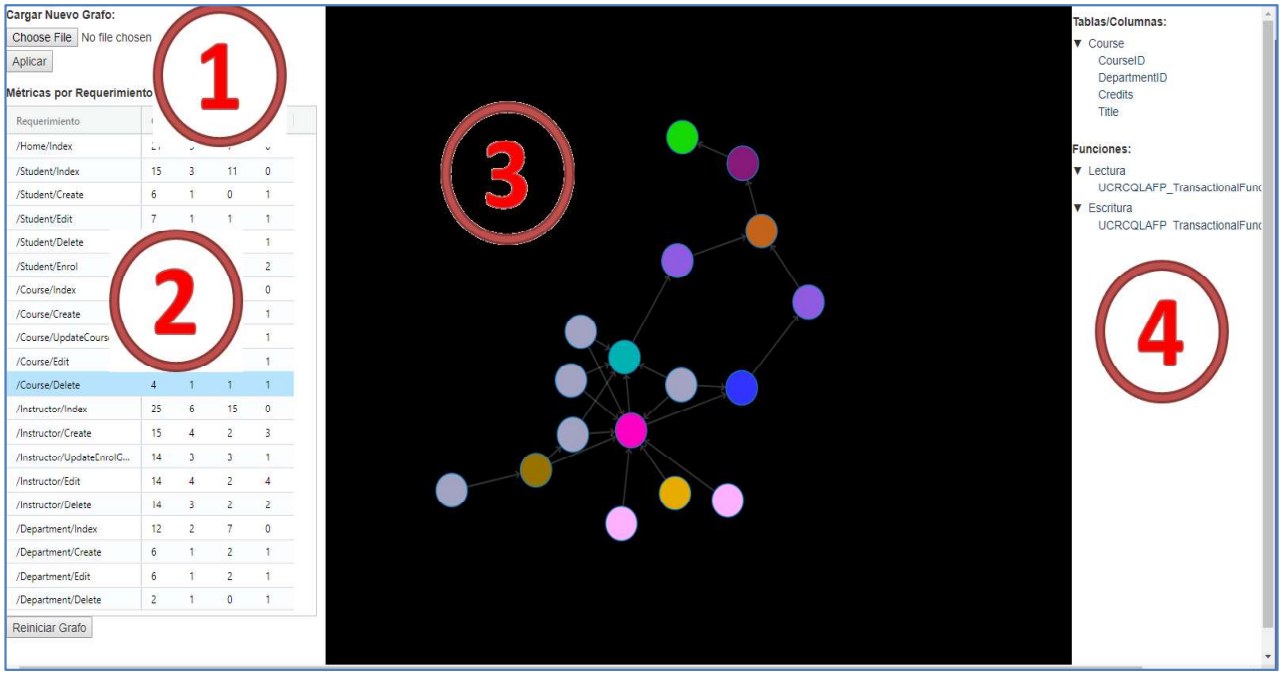



*Figura 19. Vista principal del prototipo*

[6] https://github.com/Readify/Neo4jClient

1. *Panel de carga de grafo*: Permite al usuario seleccionar el archivo de interfaz que representa el modelo entrante del sistema de software a visualizar.
2. *Panel de métricas:* Muestra las métricas obtenidas del grafo total en pantalla. El análisis detallado de las métricas que se visualizan se especifica más adelante en este documento.
3. *Panel del grafo:* Componente central que despliega el grafo en su totalidad, o el subgrafo definido por el registro seleccionado en el panel de métricas.
4. *Panel de detalles:* Componente que muestra la jerarquía de elementos según el registro seleccionado en el panel de métricas.

### 5.4.2. Funcionalidades de la aplicación

Se listan a continuación cada una de las características esperadas según la especificación de requerimientos de la sección 5.2.

**1. Generar el diagrama de grafo que representa la aplicación de software.**

En el panel de carga de grafo, se muestra al usuario el botón de "Seleccionar archivo" o "*Choose File*" mostrado en la Figura 20. De esta forma es posible elegir el archivo del grafo a desplegar mediante el cuadro de diálogo "Abrir" como se ilustra en la Figura 21. Tras seleccionar el archivo, el botón "Aplicar" permite ejecutar la carga del grafo.

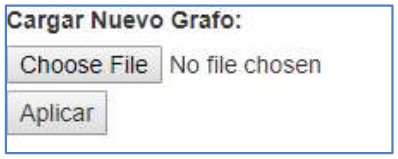



*Figura 20. Panel de carga de grafo*

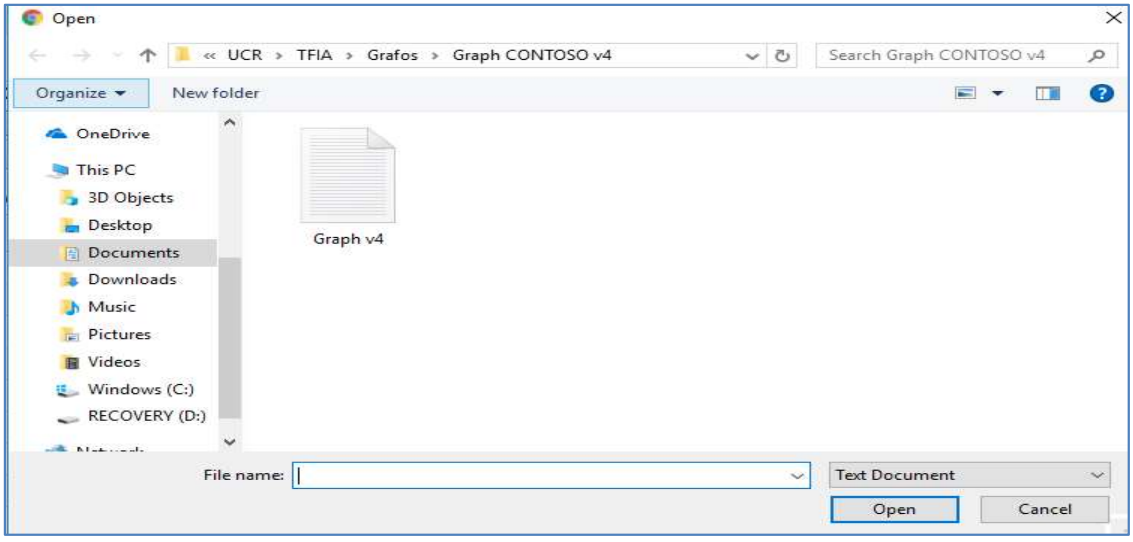



*Figura 21. Ventana de selección de archivo*

La Figura 22 contiene un extracto del archivo de interfaz XML utilizado como entrada para el prototipo:

```
<?xml version="1.0" encoding="utf-8"?>
<event_model>
  <application_id>CONTOSO</application_id>
  <application_name>CONTOSO</application_name>
  <events>
    <event>
      <user_interface_name>CONSULTAESTUDIANTES</user_interface_name>
      <control_name>PRUEBAS.CONTOSO.MNTESTUDIANTES.CONSULTAESTUDIANTES.ff_MNTESTUDIANTES_filtroestudiantes</control_name>
      <event_name>onLoad</event_name>
      <file_name />
      <position />
      <requirement>(R4.4) Consulta Estudiantes y (R4.5) Filtro por Nombre</requirement>
      <description>consulta de marco</description>
      <funtions>
        <funtion>
          <sql>SELECT 1 AS id,CONVERT(TIMESTAMP, 0x0) AS cambios</sql>
        </funtion>
      </funtions>
    </event>
    <event>
      <user_interface_name>CONSULTAESTUDIANTES</user_interface_name>
      <control_name>PRUEBAS.CONTOSO.MNTESTUDIANTES.CONSULTAESTUDIANTES.ll_MNTESTUDIANTES_listaestudiantes</control_name>
      <event_name>onLoad</event_name>
      <file_name />
      <position />
      <requirement>(R4.4) Consulta Estudiantes y (R4.5) Filtro por Nombre</requirement>
      <description>consulta de marco</description>
      <funtions>
        <funtion>
          <sql>SELECT e.idestudiante,
    e.identificador,
    e.apellido,
    e.nombre,
    e.correo,
    e.fecharegistro,
    e.cambios
FROM contoso.Estudiante e
WHERE e.Nombre = ''</sql>
        </funtion>
      </funtions>
    </event>
```

*Figura 22. Interfaz XML*

Como se muestra en la Figura 22, el archivo contiene la estructura XML con una etiqueta raíz *event_model* seguida por las etiquetas correspondientes al identificador y

nombre de la aplicación a visualizar (*application_id* y *application_name*). Seguidamente se listan en el archivo las etiquetas de tipo *event* correspondientes a los eventos capturados a partir de las funcionalidades de la aplicación, mostrando para cada uno de éstos los elementos relacionados correspondientes al metamodelo expuesto previamente en la sección 5.1. Este archivo XML es transformado en una secuencia de sentencias en lenguaje *Cypher* (Figura 23) las cuales son ejecutadas por el motor *Neo4j* para almacenar el grafo previo a su visualización.

```
CREATE (h63292653:APPLICATION {name:'CONTOSO', hash:632926
CREATE (h32762966:USER_INTERFACE {name:'CONSULTAESTUDIANTE
CREATE (h64976233:CONTROL {name:'PRUEBAS.CONTOSO.MNTESTUDI
CREATE (h55925239:EVENT {name:'onLoad', hash:55925239, id:
CREATE (h33565108:FUNCTION {name:'UCRCQLAFP_TransactionalF
```

*Figura 23. Archivo Cypher*

En la figura se muestran las etiquetas del archivo utilizadas por la herramienta: el comando "CREATE", el correspondiente código alfanumérico *hash* (el cual inicia con la letra "h"), y los atributos "name" y "id". El resto de atributos son parte del archivo transformado, sin embargo, son ignorados por la herramienta. Las sentencias CREATE en lenguaje Cypher son las encargadas de crear cada uno de los nodos y las relaciones entre los mismos. Por ejemplo, la primera línea que se aprecia en la Figura 23 corresponde a la creación del nodo raíz de la aplicación a visualizar, especificándole un identificador único y su tipo de nodo APPLICATION, además de los atributos propios de dicho nodo. En las líneas subsecuentes se agrega a la base de datos el resto de los nodos correspondientes a los elementos dentro del modelo funcional.

**2. Permitir al usuario manipular el diagrama de grafo en pantalla.**

La herramienta permite realizar las siguientes acciones sobre el grafo:

- Identificar el tipo de nodo según su color específico. El Cuadro 3 detalla el código de colores utilizado para la visualización del grafo de acuerdo a los componentes del metamodelo:

*Cuadro 3. Código de colores del grafo*

| Color | Código hexadecimal | Tipo de nodo |
|---|---|---|
| | E60000 | APPLICATION |
| | 16D803 | USER_INTERFACE |
| | 871A78 | CONTROL |
| | C4641A | EVENT |
| | 8E5BE1 | FUNCTION |
| | FF00C5 | TABLE |
| | 3333FF | PROCEDURE |
| | CC4400 | DATA_BASE |
| | A3A3C2 | COLUMN |
| | E6AC00 | PRIMARY_KEY |
| | 997300 | FOREIGN_KEY |
| | FFB3FF | INDEX |
| | 00B3B3 | VIEW |

- Alejar y acercar la vista del grafo mediante la rueda de navegación del *mouse* (*scroll*), además de mover el grafo al arrastrar un nodo específico o bien el fondo de la pantalla, como se muestra en la Figura 24.

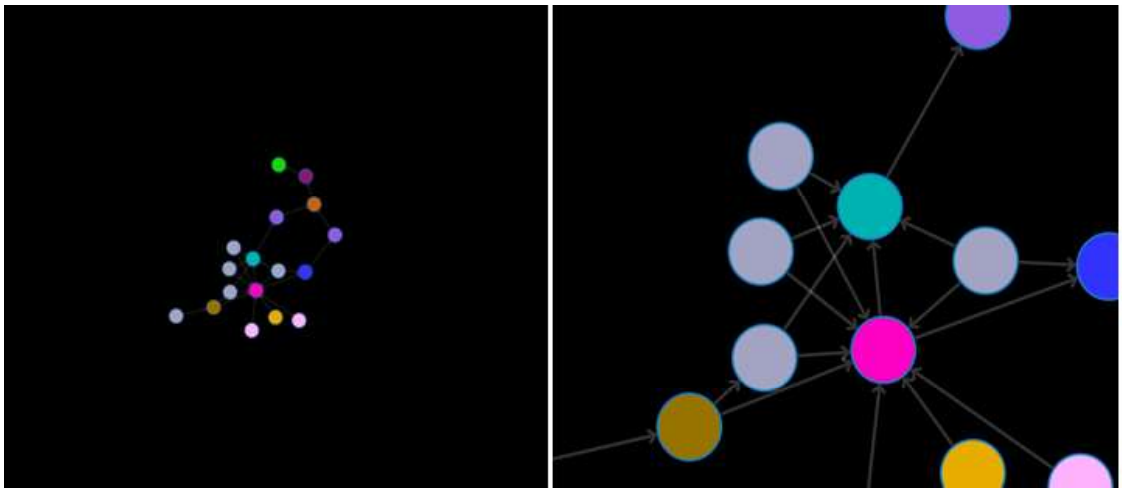


*Figura 24. Vista lejana y cercana del grafo*

- Ver el nombre de los nodos y aristas al posicionar el cursor encima mediante *tooltip* (Figura 25).

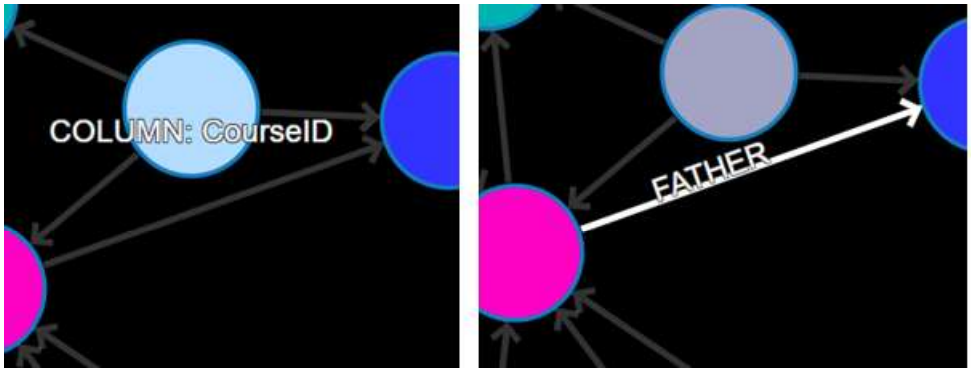



*Figura 25. Tooltip en nodos y aristas*

- Cambiar entre la vista del grafo completo y una parte del grafo seleccionada en el panel de métricas. La Figura 26 muestra el subgrafo determinado por los nodos involucrados según el registro seleccionado en el panel de métricas (explicadas en detalle más adelante). La Figura 27 muestra el grafo completo luego de presionar el botón "Reiniciar Grafo"

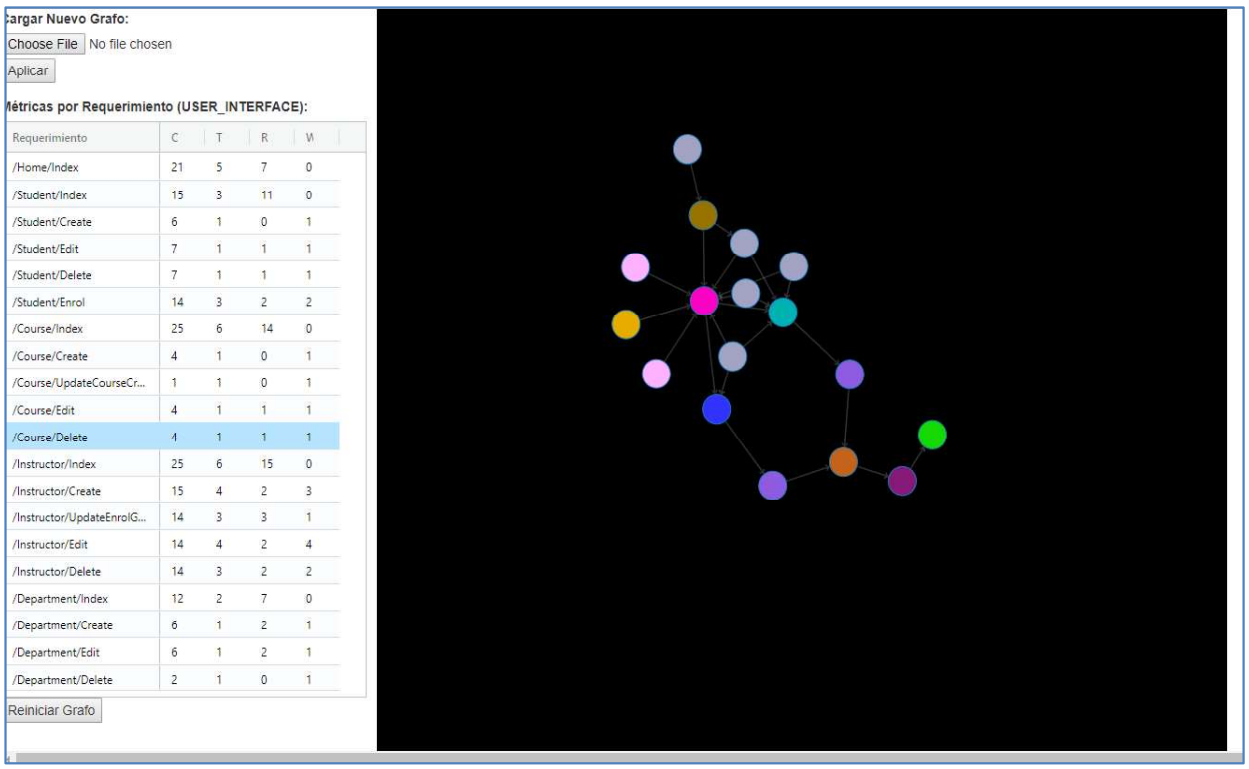
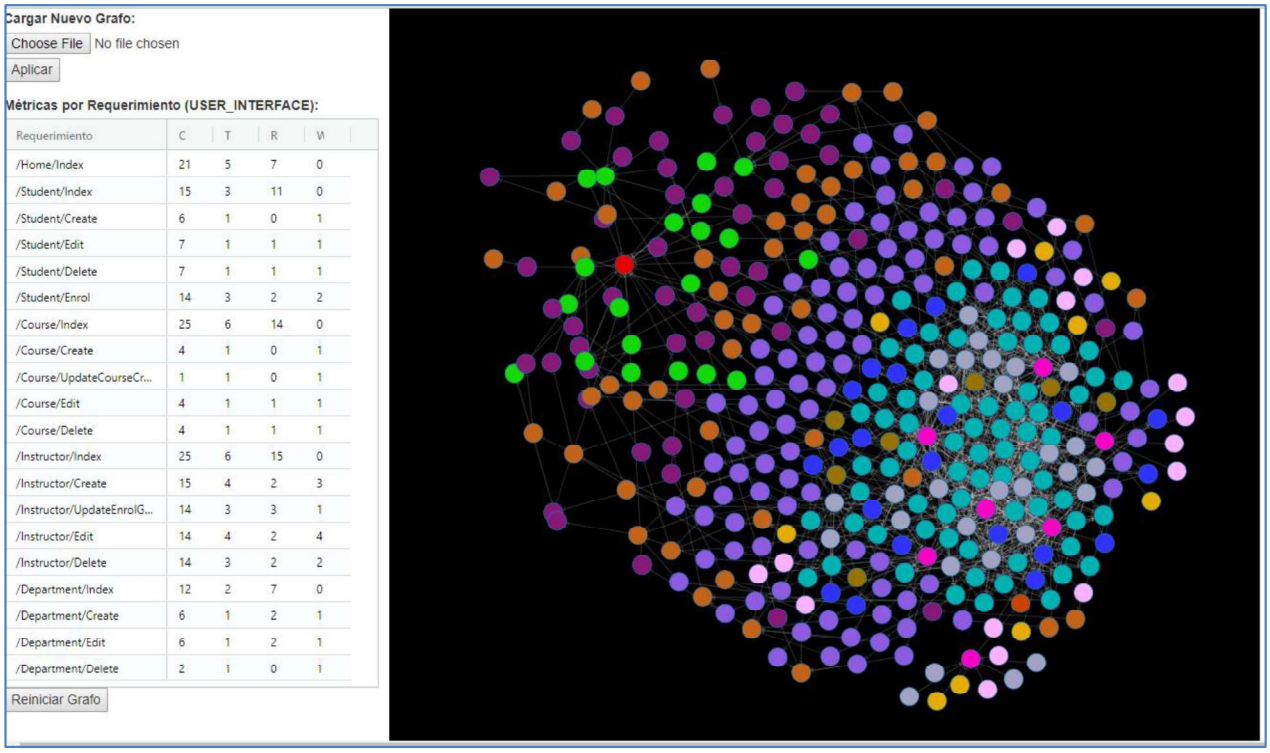



*Figura 26. Vista de subgrafo*




| Requerimiento | C | T | R | W |
|---|---|---|---|---|
| /Home/Index | 21 | 5 | 7 | 0 |
| /Student/Index | 15 | 3 | 11 | 0 |
| /Student/Create | 6 | 1 | 0 | 1 |
| /Student/Edit | 7 | 1 | 1 | 1 |
| /Student/Delete | 7 | 1 | 1 | 1 |
| /Student/Enrol | 14 | 3 | 2 | 2 |
| /Course/Index | 25 | 6 | 14 | 0 |
| /Course/Create | 4 | 1 | 0 | 1 |
| /Course/UpdateCourseCr... | 1 | 1 | 0 | 1 |
| /Course/Edit | 4 | 1 | 1 | 1 |
| /Course/Delete | 4 | 1 | 1 | 1 |
| /Instructor/Index | 25 | 6 | 15 | 0 |
| /Instructor/Create | 15 | 4 | 2 | 3 |
| /Instructor/UpdateEnrolG... | 14 | 3 | 3 | 1 |
| /Instructor/Edit | 14 | 4 | 2 | 4 |
| /Instructor/Delete | 14 | 3 | 2 | 2 |
| /Department/Index | 12 | 2 | 7 | 0 |
| /Department/Create | 6 | 1 | 2 | 1 |
| /Department/Edit | 6 | 1 | 2 | 1 |
| /Department/Delete | 2 | 1 | 0 | 1 |

*Figura 27. Vista de grafo completo*

### 3. Mostrar métricas.

El prototipo carga por defecto las métricas descritas en la sección de requerimientos de este capítulo, obtenidas a partir de operaciones de grafos aplicadas sobre la estructura del grafo visualizado. Como ejemplo se muestra la Cuadro 4, donde cada fila representa un “Requerimiento” o “Interfaz de Usuario” de la aplicación visualizada (según el metamodelo descrito en la sección 5.1). Para ejemplificar la interpretación de las métricas detalladas a continuación, en el Cuadro 4 se puede deducir que el requerimiento seleccionado consulta o modifica cuatro columnas pertenecientes a una única tabla en la base de datos. Así mismo se interpreta que esa tabla es leída o modificada por una transacción funcional de lectura y una de escritura.

*Cuadro 4. Métricas*

**Métricas por Requerimiento (USER_INTERFACE):**

| Requerimiento | C | T | R | W |
|---|---|---|---|---|
| /Home/Index | 21 | 5 | 7 | 0 |
| /Student/Index | 15 | 3 | 11 | 0 |
| /Student/Create | 6 | 1 | 0 | 1 |
| /Student/Edit | 7 | 1 | 1 | 1 |
| /Student/Delete | 7 | 1 | 1 | 1 |
| /Student/Enrol | 14 | 3 | 2 | 2 |
| /Course/Index | 25 | 6 | 14 | 0 |
| /Course/Create | 4 | 1 | 0 | 1 |
| /Course/UpdateCourseCr... | 1 | 1 | 0 | 1 |
| /Course/Edit | 4 | 1 | 1 | 1 |
| /Course/Delete | 4 | 1 | 1 | 1 |
| /Instructor/Index | 25 | 6 | 15 | 0 |
| /Instructor/Create | 15 | 4 | 2 | 3 |
| /Instructor/UpdateEnrolG... | 14 | 3 | 3 | 1 |
| /Instructor/Edit | 14 | 4 | 2 | 4 |
| /Instructor/Delete | 14 | 3 | 2 | 2 |
| /Department/Index | 12 | 2 | 7 | 0 |
| /Department/Create | 6 | 1 | 2 | 1 |
| /Department/Edit | 6 | 1 | 2 | 1 |
| /Department/Delete | 2 | 1 | 0 | 1 |

Las métricas determinadas por las columnas en el Cuadro 4 se calculan de la siguiente manera:

*Columna C*: cantidad de atributos o columnas de tablas de la base de datos afectadas por el requerimiento, ya sea mediante funciones transaccionales de lectura o escritura. La Figura 28 muestra el extracto del código fuente de la herramienta de visualización que calcula esta métrica a partir del lenguaje *Cypher*.

```
var queryMetricaColumnas = WebApiConfig.GraphClient.Cypher
    .Match("(ui:USER_INTERFACE)<-[*]-(x)<--(n:COLUMN)").Where("x:VIEW or x:PROCEDURE")
    .Return((ui, n) => new
    {
        name = Return.As<string>("ui.name"),
        count = n.CountDistinct()
    });
```

*Figura 28. Métrica cantidad de columnas*

La cantidad de columnas es calculada para cada requerimiento al realizar el conteo en la base de datos Neo4j de todos los nodos distintos de tipo COLUMN que se encuentran conectados indirectamente a un nodo USER_INTERFACE mediante algún nodo intermedio de tipo VIEW o PROCEDURE.

*Columna T*: cantidad de tablas afectadas por el requerimiento, ya sea mediante funciones transaccionales de lectura o escritura. La Figura 29 detalla el cálculo de esta métrica.

```
var queryMetricaTablas = WebApiConfig.GraphClient.Cypher
    .Match("(ui:USER_INTERFACE)<-[*]-(x)<--(n:TABLE)").Where("x:VIEW or x:PROCEDURE")
    .Return((ui, n) => new
    {
        name = Return.As<string>("ui.name"),
        count = n.CountDistinct()
    });
```

*Figura 29. Métrica cantidad de tablas*

De manera similar a la métrica anterior, para cada requerimiento se obtiene el conteo de todos los nodos distintos de tipo TABLE que se encuentran conectados indirectamente a un nodo de tipo USER_INTERFACE mediante algún nodo intermedio de tipo VIEW o PROCEDURE.

*Columna R:* cantidad de funciones transaccionales (descritas en el metamodelo de la sección 5.1) de lectura invocadas por el requerimiento. En la Figura 30 se muestra el fragmento de código que obtiene esta métrica.

```
var queryMetricaFunLectura = WebApiConfig.GraphClient.Cypher
    .Match("(ui:USER_INTERFACE)<-[*]-(n:FUNCTION)").Where("n.id = 'Select'")
    .Return((ui, n) => new
    {
        name = Return.As<string>("ui.name"),
        count = n.CountDistinct()
    });
```

*Figura 30. Métrica funciones de lectura*

Para calcular esta métrica para cada requerimiento, se obtiene el conteo de todos los nodos distintos de tipo FUNCTION que se encuentran conectados indirectamente a un nodo de tipo USER_INTERFACE mediante algún nodo que contenga la sentencia "Select" dentro de su atributo "Id".

*Columna W:* cantidad de funciones transaccionales de escritura invocadas por el requerimiento. La Figura 31 detalla el cálculo de esta métrica.

```
var queryMetricaFunEscritura = WebApiConfig.GraphClient.Cypher
    .Match("(ui:USER_INTERFACE)<-[*]-(n:FUNCTION)").Where("n.id <> 'Select'")
    .Return((ui, n) => new
    {
        name = Return.As<string>("ui.name"),
        count = n.CountDistinct()
    });
```

*Figura 31. Métrica funciones escritura*

Esta métrica realiza el conteo de todos los nodos distintos de tipo FUNCTION que se encuentran conectados indirectamente a un nodo de tipo USER_INTERFACE mediante algún nodo cuyo atributo "Id" no contenga la sentencia "Select".

Para apoyar la información mostrada en el panel de métricas, la herramienta despliega también en el panel de detalles el nombre de los componentes afectados por el requerimiento seleccionado en el panel de métricas. En este caso la Figura 32 muestra el nombre de la tabla afectada *Course* y sus cuatro columnas afectadas (*CourseID, DepartmentID, Credits, Title*) por las funciones de lectura y escritura ilustradas.

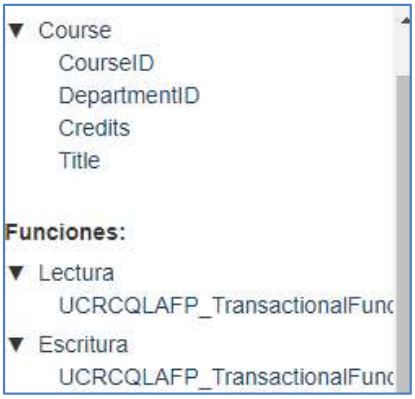



*Figura 32. Panel de detalles*

En la Figura 33 se muestra el estado de la visualización completa siguiendo el ejemplo anterior, en donde es posible apreciar las métricas del requerimiento seleccionado, los nodos correspondientes en el grafo y los componentes afectados.

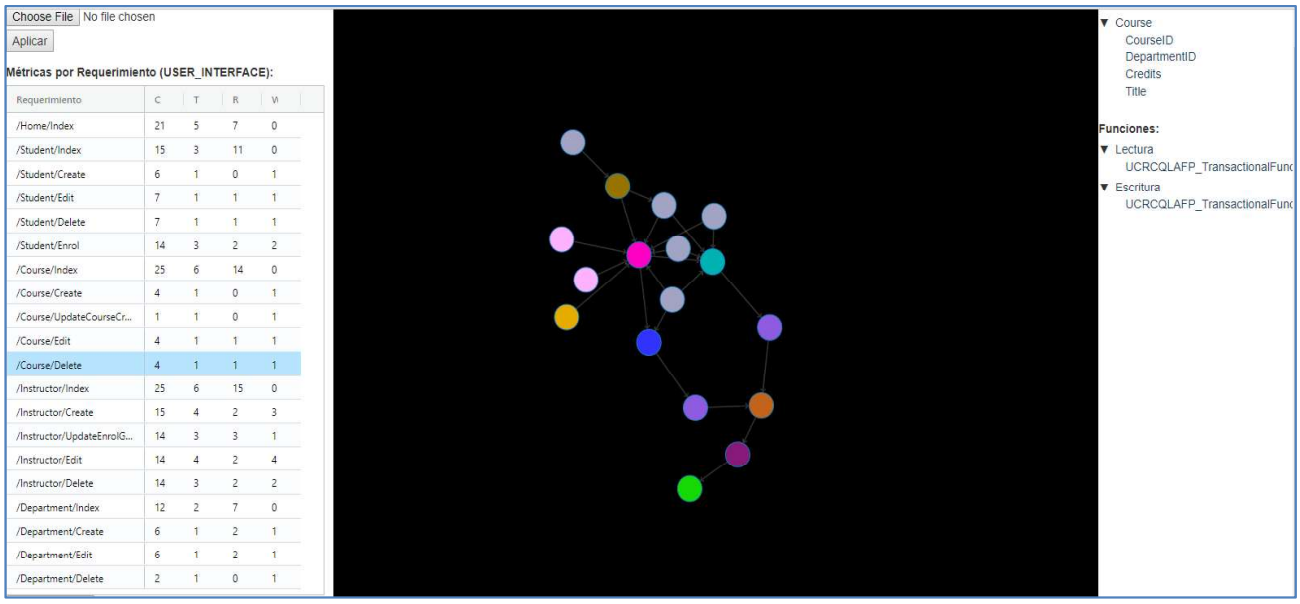



*Figura 33. Ejemplo de visualización completa*

El elemento visual de la herramienta complementa los datos numéricos para dar una noción de tamaño y complejidad de la aplicación de software que se está visualizando. Por ejemplo, en la Figura 34 se observa el subgrafo correspondiente a un requerimiento que posee unas métricas pequeñas en relación con los demás requerimientos.

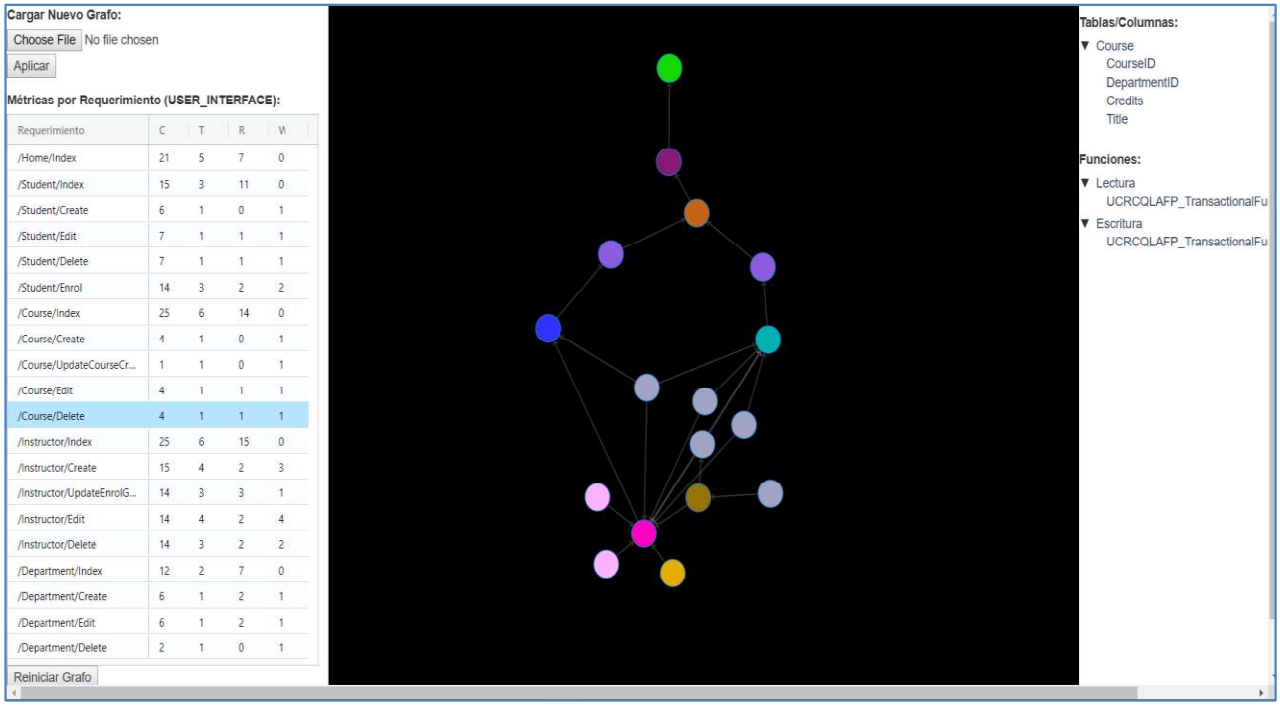



*Figura 34. Subgrafo pequeño*

En la Figura 34 se observa como el subgrafo muestra los elementos correspondientes de un requerimiento que tiene solamente cuatro columnas afectadas en una sola tabla de la base de datos, así como una única función de lectura y otra de escritura (registro seleccionado en el panel de métricas ubicado en la parte izquierda de la pantalla). Del mismo modo, la Figura 35 muestra el subgrafo que corresponde a un requerimiento de mediano tamaño si se compara con el resto de los requerimientos de la aplicación visualizada.

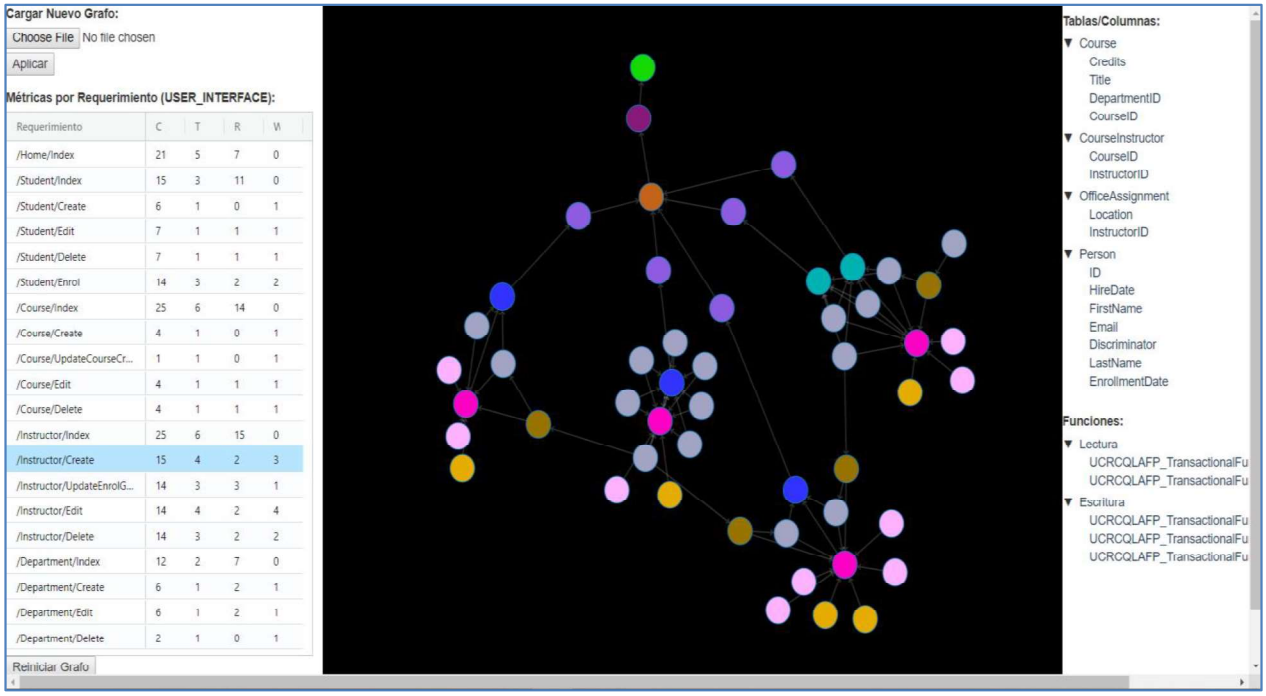



*Figura 35. Subgrafo de requerimiento mediano*

La Figura 35 muestra el subgrafo relacionado a un requerimiento que involucra quince columnas distribuidas en cuatro tablas dentro de la base de datos, afectadas por dos funciones de lectura y tres funciones de escritura. Es posible establecer una comparación visual entre los subgrafos de las Figuras 34 y 35, evidenciando un mayor tamaño en el subgrafo correspondiente a un requerimiento con métricas mayores. Siguiendo el mismo criterio de comparación es posible determinar que un requerimiento con métricas aún mayores que los dos anteriormente ilustrados mostrará un subgrafo aún más grande, como se muestra en la Figura 36.

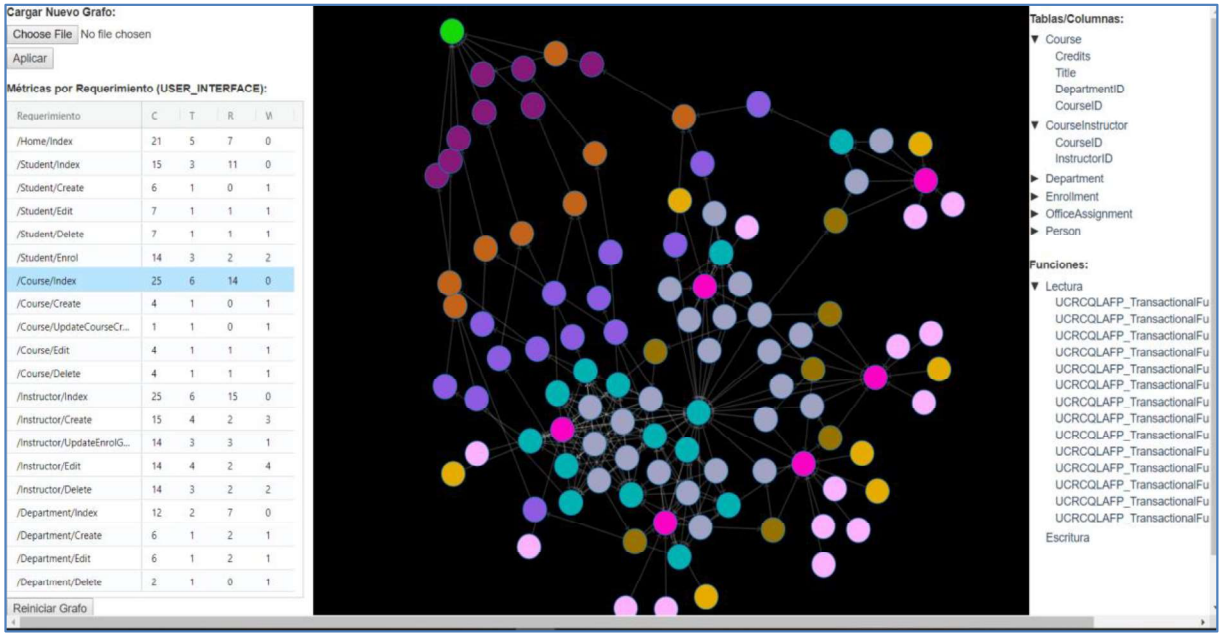



*Figura 36. Subgrafo de requerimiento grande*

La Figura 36 despliega el subgrafo que representa un requerimiento de tamaño grande en comparación con el resto de los requerimientos visualizados, afectando un total de veinticinco columnas distribuidas en seis tablas en la base de datos, utilizadas por catorce funciones de lectura. Al comparar las Figuras 34, 35 y 36 es posible observar la diferencia de tamaño existente entre los tres subgrafos, permitiendo al usuario tener una noción de tamaño y complejidad sobre los requerimientos visualizados.

En este capítulo se describió el prototipo de la herramienta de visualización de software desarrollado como parte de esta investigación. Todos los requerimientos funcionales especificados en la sección 5.2 fueron implementados en el prototipo. A continuación el Capítulo 6 explica el procedimiento de evaluación empírica al que fueron sometidas las métricas calculadas por el prototipo de visualización desarrollado. Las métricas calculadas fueron analizadas mediante casos de estudio que comparan los resultados generados por la herramienta, contra las métricas de puntos de función calculadas previamente por el módulo de medición [20].

# Capítulo 6. Evaluación

Este capítulo describe los casos de estudio para evaluar la efectividad de las métricas extraídas del modelo funcional para estimar el tamaño y la complejidad de la aplicación de software representada. Los modelos funcionales se visualizaron mediante la herramienta prototipo y a partir de estos se calcularon las métricas para estimar el tamaño. La estructura del reporte de esta evaluación se realizó siguiendo los lineamientos recomendados para estudios empíricos en ingeniería de software propuestos en [22].

En esta investigación se desarrolló un prototipo de herramienta de visualización de software y evaluó la efectividad de las métricas extraídas del modelo que representa para determinar el tamaño y la complejidad de una aplicación de software. La herramienta visualiza el modelo funcional y calcula las métricas que permiten estimar el tamaño funcional.

## 6.1. Diseño de la evaluación

Los objetos de estudio son cuatro aplicaciones de software transaccionales, de las cuales se extrajeron los modelos funcionales que sirvieron como entrada para el análisis. Además se cuenta con los correspondientes conteos de puntos de función sin ajustar (UFP) para las funciones transaccionales, tomados de investigaciones previas. Los detalles de cada herramienta y los datos de entrada obtenidos se explican posteriormente en este capítulo.

En este estudio se planteó la siguiente pregunta de investigación:

(RQ1) ¿Cuál es la efectividad de las métricas extraídas a partir del modelo funcional para estimar el tamaño y la complejidad de la aplicación de software representada?

A partir del modelo de representación funcional y sus componentes se calculó un conjunto de métricas extraídas del grafo y se analizó la correlación que existe entre estas métricas obtenidas por la herramienta de visualización (componentes de tipo tabla, columna, función de lectura y escritura), contra el tamaño funcional en puntos de función sin ajustar de las funciones transaccionales previamente calculadas.

Se analizaron los resultados para cada una de las mediciones y componentes de cada uno de los requerimientos de las aplicaciones analizadas. Respondiendo a la pregunta de investigación podemos determinar la eficacia de las métricas seleccionados del modelo para estimar el tamaño funcional de la aplicación de software que se visualiza.

### 5.5.1. Procedimiento de recolección de datos

Las tareas realizadas durante el caso de estudio para la recolección de datos de cada una de las aplicaciones de software son las siguientes:

1. Se extrajo el modelo funcional de la aplicación de software utilizando el prototipo de medición de tamaño funcional propuesto en [20].
2. Se ejecutó la herramienta de visualización utilizando como entrada el modelo funcional de la aplicación seleccionada.
3. Se obtuvieron las métricas calculadas por la herramienta de visualización. El resultado es un conteo de componentes afectados por requerimiento.
4. Se tabularon los resultados del conteo de puntos de función sin ajustar por requerimiento reportados en [23] para ser utilizado como valor real o de referencia.
5. Se realizó el cálculo de correlación múltiple entre los valores de las métricas extraídas por medio de la herramienta de visualización a partir del modelo, y el valor en puntos de función sin ajustar de cada requerimiento funcional (funciones transaccionales).
6. Se realizó el análisis de la exactitud de las estimaciones del tamaño funcional.

### 5.5.2. Amenazas a la validez

El conteo de puntos de función fue validado por dos profesionales con experiencia en medición [23], sin embargo, no son contadores certificados. Así mismo, se realizó un preprocesamiento de datos que permitió homologar algunos de los conteos de las transacciones por requerimiento por parte del autor de este trabajo (por ejemplo, unir grupos de requerimientos obtenidos mediante la herramienta, ya que el modelado de los requerimientos entre ambos conteos, en algunos casos, presentaba diferencias).

## 6.2. Resultados

El Cuadro 5 presenta la comparación de los resultados obtenidos al analizar la exactitud de las estimaciones para cada una de las aplicaciones de software medidas, mostrando los valores totales de la exactitud para el tamaño funcional total y la exactitud basado en los valores de cada uno de los requerimientos funcionales. Se presentan las columnas de izquierda a derecha: nombre de la aplicación, total de UFP (*Unadjusted Function Points*) para las funciones transaccionales de la aplicación, la fórmula de correlación, el coeficiente de correlación $R^2$ obtenido, la cantidad de requerimientos analizados, la métrica MRE (magnitud del error relativo) del total de UFP, y la mediana de las métricas MRE de cada requerimiento de la aplicación.

Según los resultados mostrados es posible visualizar una varianza en la exactitud que existe entre los UFP de referencia totalizados y los valores estimados (valores entre 0% y 1%). Del mismo modo, la mediana de la métrica MRE para los requerimientos individuales de cada aplicación muestra valores entre 0% y 26%, sugiriendo una exactitud alta tanto a nivel de conteo total como individual por requerimiento. Es importante destacar además que el coeficiente $R^2$ relativamente bajo en el caso de estudio PMS (en comparación con Contoso V1 y Santana) no determinó el porcentaje de exactitud de las mediciones, reforzando que es necesario complementar los análisis de correlación con las comparaciones empíricas con el análisis de exactitud para evaluar las métricas del prototipo.

*Cuadro 5. Resumen de resultados de casos de estudio*

| Nombre de Aplicación | Conteo Manual UFP (TF) | Correlacion | $R^2$ | Cantidad de Requerimientos | MRE Total UFP | Mediana MRE por requerimiento |
|---|---|---|---|---|---|---|
| Contoso V1 | 125 | 3.99 + (Columnas * 0.32) + (Tablas * 2.46) + (FR * 0.73) + (FW * 2.08) | 0.969 | 19 | 0% | 15% |
| Contoso V2 | 129 | 2.31 + (Columnas * -0.49) + (Tablas * 1.96) + (FR * 1.94) + (FW * -1.08) | 0.593 | 27 | 0% | 26% |
| PMS | 85 | 1.87 + (Columnas * -0.08) + (Tablas * 0.16) + (FR * 0.48) + (FW * 0.56) | 0.524 | 25 | 1% | 10% |
| Santana | 23 | -4.00 + (Columnas * -0.24) + (Tablas * 0.00) + (FR * 4.33) + (FW * 0.00) | 1 | 3 | 0% | 0% |

La Figura 37 muestra la comparación entre los valores estimados según los cálculos a partir de la fórmula de regresión, y los valores de referencia correspondientes a los conteos de UFP para las funciones transaccionales.

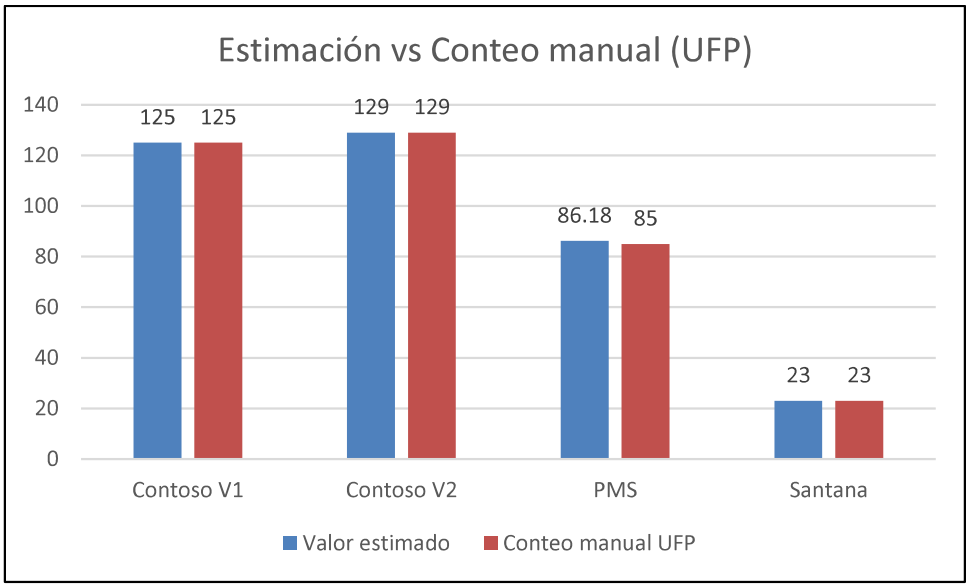



*Figura 37. Comparación entre valores estimados y conteos manuales (UFP)*

A continuación se detalla el análisis de cada una de las aplicaciones de software objeto de cada caso de estudio.

Contoso V1:

Consiste en una aplicación con funcionalidades básicas para la administración de la matrícula de una organización educativa. Esta es una aplicación web transaccional desarrollada en ASP.NET MVC que utiliza bases de datos SQL Server. El detalle de los requerimientos, que son base para este caso de estudio, fueron obtenidos de [23] y se muestran en el Cuadro 6.

*Cuadro 6. Requerimientos funcionales de Contoso V1*

| Requerimientos de Contoso |
|---|
| (R1.1) Crear un curso |
| (R1.2) Modificar información de curso |
| (R1.3) Borrar curso |
| (R1.4) Consultar información de cursos |
| (R1.5) Filtrar o buscar información de cursos por nombre |
| (R1.6) Filtrar o buscar información de cursos por departamento |
| (R1.7) Reporte de información de un curso |

| |
|---|
| (R1.8) Reporte de información de un curso, los instructores asignados y los estudiantes matriculados |
| (R1.9) Actualizar el número de créditos de todos los cursos de la universidad. |
| (R2.1) Crear un instructor |
| (R2.2) Modificar información de instructor |
| (R2.3) Borrar instructor |
| (R2.4) Consultar información de instructores |
| (R2.5) Consultar información de detalle de cursos por instructor |
| (R2.6) Consultar información de detalle de estudiantes del curso por instructor |
| (R2.7) Reporte de información de un instructor |
| (R2.8) Reporte de datos de Instructor y cursos asignados |
| (R3.1) Crear un departamento |
| (R3.2) Modificar información de un departamento |
| (R3.3) Borrar departamento |
| (R3.4) Consultar información de departamentos |
| (R3.5) Reporte de información de departamento |
| (R4.1) Crear un estudiante |
| (R4.10) Reporte de estadísticas de estudiantes matriculados |
| (R4.2) Modificar información de un estudiante |
| (R4.3) Borrar estudiante |
| (R4.4) Consultar información de estudiantes |
| (R4.5) Filtrar o buscar información de estudiantes por nombre |
| (R4.6) Matricular cursos a un estudiante |
| (R4.7) Reporte de datos de estudiantes y cursos matriculados |
| (R4.8) Asignar nota de curso a estudiante |
| (R4.9) Reporte de datos de estudiante y cursos matriculados (Reporte de Notas) |

El Cuadro 7 muestra los datos recolectados para el análisis. Para cada requerimiento de la aplicación Contoso se detalla el tamaño funcional (UFP, Conteo Manual). Los campos marcados con asterisco "*" en la columna "User_Interface" representan los requerimientos que debieron ser agrupados en el conteo manual para poder compararse con un requerimiento resultante de la métrica del modelo funcional que se obtuvo por la herramienta. En las restantes columnas se detallan los valores de las métricas del modelo que incluyen las métricas de Tablas, Columnas, Funciones de Lectura y Escritura.

*Cuadro 7. Métricas recolectadas para el análisis de Contoso V1*

| Requerimiento | User_Interface (Herramienta) | UFP (Conteo Manual) | Columnas | Tablas | Funciones Read | Funciones Write |
|---|---|---|---|---|---|---|
| (R1.1) Crear un curso | /Course/Create | 4 | 4 | 1 | 0 | 1 |
| (R1.2) Modificar información de curso | /Course/Edit | 4 | 4 | 1 | 1 | 1 |
| (R1.3) Borrar curso | /Course/Delete | 4 | 4 | 1 | 1 | 1 |
| (R1.4) Consultar información de cursos | /Course/Index* | 23 | 25 | 6 | 14 | 0 |
| (R1.9) Actualizar el número de créditos de todos los cursos de la universidad. | /Course/UpdateCourseCredits | 3 | 1 | 1 | 0 | 1 |
| (R2.1) Crear un instructor | /Instructor/Create | 3 | 15 | 4 | 2 | 3 |
| (R2.2) Modificar información de instructor | /Instructor/Edit | 3 | 14 | 4 | 2 | 4 |
| (R2.3) Borrar instructor | /Instructor/Delete | 3 | 14 | 3 | 2 | 2 |
| (R2.4) Consultar información de instructores | /Instructor/Index* | 21 | 25 | 6 | 15 | 0 |
| (R3.1) Crear un departamento | /Department/Create | 4 | 6 | 1 | 2 | 1 |
| (R3.2) Modificar información de un departamento | /Department/Edit | 4 | 6 | 1 | 2 | 1 |
| (R3.3) Borrar departamento | /Department/Delete | 4 | 2 | 1 | 0 | 1 |
| (R3.4) Consultar información de departamentos | /Department/Index* | 8 | 12 | 2 | 7 | 0 |
| (R4.1) Crear un estudiante | /Student/Create | 3 | 6 | 1 | 0 | 1 |
| (R4.2) Modificar información de un estudiante | /Student/Edit | 3 | 7 | 1 | 1 | 1 |
| (R4.3) Borrar estudiante | /Student/Delete | 4 | 7 | 1 | 1 | 1 |
| (R4.4) Consultar información de estudiantes | /Student/Index* | 15 | 15 | 3 | 11 | 0 |
| (R4.6) Matricular cursos a un estudiante | /Student/Enrol | 6 | 14 | 3 | 2 | 2 |
| (R4.8) Asignar nota de curso a estudiante | /Instructor/UpdateEnrolGrade | 6 | 14 | 3 | 3 | 1 |

Para esta aplicación y con las métricas del modelo, el análisis de correlación indica un coeficiente de determinación ($R^2$) con un valor de 0.969, lo que representa una correlación lineal múltiple positiva alta entre los puntos de función UFP y las métricas del modelo. El Cuadro 8 presenta el resultado de la correlación múltiple establecida entre los valores del tamaño funcional en puntos de función sin ajustar (UFP, IFPUG FPA) y las métricas de componentes del modelo funcional.

*Cuadro 8. Cálculo de correlación múltiple para Contoso V1*

| *Regression Statistics* | |
|---|---|
| Multiple R | 0.985 |
| **R Square** | **0.969** |
| Adjusted R Square | 0.961 |
| Standard Error | 1.215 |
| Observations | 19.000 |

A partir de estos resultados, analizamos las diferencias de la exactitud entre los valores reales y los valores estimados a partir de la correlación. El Cuadro 9 muestra la lista de requerimientos, seguido del conteo de UFP y las métricas del modelo por requerimiento. Asimismo, se muestran los valores estimados de tamaño funcional a partir

de la ecuación de regresión múltiple calculada con los coeficientes obtenidos en el análisis, la diferencia con el valor real y las métricas de exactitud MRE (magnitud del error relativo), MER (magnitud relativa del error) y BRE (error relativo balanceado).

*Cuadro 9. Análisis de exactitud para Contoso V1*

| User_Interface (Herramienta) | UFP (Conteo Manual) | Columnas | Tablas | Funciones Read | Funciones Write | Valor Estimado | Diferencia | MRE | MER | BRE |
|---|---|---|---|---|---|---|---|---|---|---|
| /Course/Create | 4 | 4 | 1 | 0 | 1 | 3.11 | 0.89 | 22% | 29% | 29% |
| /Course/Edit | 4 | 4 | 1 | 1 | 1 | 3.84 | 0.16 | 4% | 4% | 4% |
| /Course/Delete | 4 | 4 | 1 | 1 | 1 | 3.84 | 0.16 | 4% | 4% | 4% |
| /Course/Index* | 23 | 25 | 6 | 14 | 0 | 21.10 | 1.90 | 8% | 9% | 9% |
| /Course/UpdateCourseCredits | 3 | 1 | 1 | 0 | 1 | 4.06 | -1.06 | 35% | 26% | 35% |
| /Instructor/Create | 3 | 15 | 4 | 2 | 3 | 4.32 | -1.32 | 44% | 31% | 44% |
| /Instructor/Edit | 3 | 14 | 4 | 2 | 4 | 2.56 | 0.44 | 15% | 17% | 17% |
| /Instructor/Delete | 3 | 14 | 3 | 2 | 2 | 4.26 | -1.26 | 42% | 30% | 42% |
| /Instructor/Index* | 21 | 25 | 6 | 15 | 0 | 21.83 | -0.83 | 4% | 4% | 4% |
| /Department/Create | 4 | 6 | 1 | 2 | 1 | 3.94 | 0.06 | 2% | 2% | 2% |
| /Department/Edit | 4 | 6 | 1 | 2 | 1 | 3.94 | 0.06 | 2% | 2% | 2% |
| /Department/Delete | 4 | 2 | 1 | 0 | 1 | 3.74 | 0.26 | 7% | 7% | 7% |
| /Department/Index* | 8 | 12 | 2 | 7 | 0 | 10.24 | -2.24 | 28% | 22% | 28% |
| /Student/Create | 3 | 6 | 1 | 0 | 1 | 2.48 | 0.52 | 17% | 21% | 21% |
| /Student/Edit | 3 | 7 | 1 | 1 | 1 | 2.89 | 0.11 | 4% | 4% | 4% |
| /Student/Delete | 4 | 7 | 1 | 1 | 1 | 2.89 | 1.11 | 28% | 38% | 38% |
| /Student/Index* | 15 | 15 | 3 | 11 | 0 | 14.67 | 0.33 | 2% | 2% | 2% |
| /Student/Enrol | 6 | 14 | 3 | 2 | 2 | 4.26 | 1.74 | 29% | 41% | 41% |
| /Instructor/UpdateEnrolGrade | 6 | 14 | 3 | 3 | 1 | 7.07 | -1.07 | 18% | 15% | 18% |
| | | | | | | | MEDIANA | 15% | 15% | 17% |

Los resultados indican que es posible estimar el tamaño funcional a partir de las métricas del modelo, donde para la métrica de exactitud MRE se obtiene una mediana del 15% con valores que van desde 2% hasta 44%. En el caso de los requerimientos que registran una mayor diferencia entre el valor esperado y el real, se encuentran "/Instructor/Create" como el valor menos exacto (44%), "/Instructor/Delete" (42%) y "/Course/UpdateCourseCredits" (35%). Una posible justificación para los dos resultados más inexactos podría radicar en el hecho de que los requerimientos relacionados a al módulo "Instructor" presentan una diferencia con el resto de módulos al tener un conteo manual de puntos de función con el valor más bajo (3 puntos cada uno), y al mismo tiempo tener una cantidad considerable de tablas y columnas afectadas (entre 14 y 15 columnas distribuidas entre 3 y 4 tablas). La tendencia del resto de módulos con un número similar de tablas y columnas afectadas es de tener una cantidad de puntos de función entre los 4 y los 23.

Para obtener una mayor fiabilidad en los resultados de la evaluación empírica, se replicó la metodología del caso de estudio aplicado a "Contoso" para analizar tres modelos adicionales. Seguidamente se detalla cada uno de estos análisis:

*Contoso V2:*

Este modelo corresponde a una nueva versión de la misma aplicación de software analizada previamente. Adicionalmente, el proceso de extracción del archivo de entrada fue distinto al del archivo utilizado para el análisis presentado anteriormente, ya que en este caso fue extraído utilizando la herramienta descrita en [23]. El listado de requerimientos y el conteo de UFP se mantiene con respecto a Contoso V1, sin embargo, en el Cuadro 10 se muestran los resultados de las métricas obtenidas por el prototipo al ejecutarlo utilizando este nuevo modelo de entrada.

*Cuadro 10. Métricas recolectadas para el análisis de Contoso Completo*

| Requerimiento | User_Interface (Herramienta) | UFP (Conteo Manual) | Columnas | Tablas | Funciones Read | Funciones Write |
|---|---|---|---|---|---|---|
| (R1.1) Crear un curso | CREARCURSO | 4 | 6 | 2 | 2 | 1 |
| (R1.2) Modificar información de curso | EDITARCURSO | 4 | 8 | 2 | 2 | 1 |
| (R1.3) Borrar curso | BORRARCURSO | 4 | 8 | 2 | 1 | 1 |
| (R1.4) Consultar información de cursos | CONSULTACURSOS* | 15 | 10 | 3 | 4 | 0 |
| (R1.7) Reporte de información de un curso | DETALLECURSO | 4 | 8 | 2 | 1 | 0 |
| (R1.8) Reporte de información de un curso, los instructores asignados y los estudiantes matriculados | REPORTECURSOS | 7 | 28 | 7 | 3 | 0 |
| (R1.9) Actualizar el número de créditos de todos los cursos de la universidad. | MULTIPLICAR | 3 | 2 | 1 | 1 | 1 |
| (R2.1) Crear un instructor | CREARINSTRUCTOR | 3 | 13 | 4 | 2 | 3 |
| (R2.2) Modificar información de instructor | EDITARINSTRUCTOR | 3 | 15 | 4 | 2 | 3 |
| (R2.3) Borrar instructor | BORRARINSTRUCTOR | 3 | 9 | 3 | 1 | 3 |
| (R2.4) Consultar información de instructores | CONSULTAINSTRUCTORES* | 10 | 13 | 4 | 1 | 0 |
| (R2.7) Reporte de información de un instructor | DETALLEINSTRUCTOR | 3 | 8 | 2 | 1 | 0 |
| (R2.8) Reporte de datos de Instructor y cursos asignados | REPORTEINSTRUCTORES | 5 | 18 | 5 | 2 | 0 |
| (R3.1) Crear un departamento | CREARDEPARTAMENTO | 4 | 8 | 2 | 2 | 1 |
| (R3.2) Modificar información de un departamento | EDITARDEPARTAMENTO | 4 | 10 | 2 | 2 | 1 |
| (R3.3) Borrar departamento | BORRARDEPARTAMENTO | 4 | 10 | 2 | 2 | 1 |
| (R3.4) Consultar información de departamentos | CONSULTADEPARTAMENTOS | 4 | 10 | 2 | 1 | 0 |
| (R3.5) Reporte de información de departamento | DETALLEDEPARTAMENTO | 4 | 10 | 2 | 2 | 0 |
| (R4.1) Crear un estudiante | CREARESTUDIANTE | 3 | 5 | 1 | 1 | 1 |
| (R4.10) Reporte de estadísticas de estudiantes matriculados | ESTADISTICASESTUDIANTES | 4 | 1 | 1 | 1 | 0 |
| (R4.2) Modificar información de un estudiante | EDITARESTUDIANTE | 3 | 7 | 1 | 1 | 1 |
| (R4.3) Borrar estudiante | BORRARESTUDIANTE | 4 | 8 | 2 | 1 | 2 |
| (R4.4) Consultar información de estudiantes | CONSULTAESTUDIANTES* | 6 | 14 | 3 | 3 | 0 |
| (R4.6) Matricular cursos a un estudiante | MATRICULARESTUDIANTE | 6 | 14 | 3 | 2 | 1 |
| (R4.7) Reporte de datos de estudiantes y cursos matriculados | DETALLEESTUDIANTE | 5 | 14 | 3 | 2 | 0 |
| (R4.8) Asignar nota de curso a estudiante | NOTA | 6 | 12 | 3 | 2 | 1 |
| (R4.9) Reporte de datos de estudiante y cursos matriculados (Reporte de Notas) | REPORTEESTUDIANTES | 4 | 15 | 3 | 2 | 0 |

Siguiendo la misma metodología de evaluación, se realizó el análisis de correlación múltiple para el modelo, cuyos resultados se muestran en el Cuadro 11.

*Cuadro 11. Cálculo de correlación múltiple para Contoso Completo*

| Regression Statistics | |
|---|---|
| Multiple R | 0.770 |
| **R Square** | **0.593** |
| Adjusted R Square | 0.519 |
| Standard Error | 1.777 |
| Observations | 27.000 |

Según el resultado obtenido, existe un menor factor de correlación entre las métricas del prototipo y los valores de referencia. No obstante, se realizó el análisis de exactitud y los resultados fueron de igual aceptables, como se detalla en el Cuadro 12.

*Cuadro 12. Análisis de exactitud para Contoso V2*

| User_Interface (Herramienta) | UFP (Conteo Manual) | Columnas | Tablas | Funciones Read | Funciones Write | Valor Estimado | Diferencia | MRE | MER | BRE |
|---|---|---|---|---|---|---|---|---|---|---|
| CREARCURSO | 4 | 6 | 2 | 2 | 1 | 6.07 | -2.07 | 52% | 34% | 52% |
| EDITARCURSO | 4 | 8 | 2 | 2 | 1 | 5.09 | -1.09 | 27% | 21% | 27% |
| BORRARCURSO | 4 | 8 | 2 | 1 | 1 | 3.15 | 0.85 | 21% | 27% | 27% |
| CONSULTACURSOS* | 15 | 10 | 3 | 4 | 0 | 11.03 | 3.97 | 26% | 36% | 36% |
| DETALLECURSO | 4 | 8 | 2 | 1 | 0 | 4.23 | -0.23 | 6% | 5% | 6% |
| REPORTECURSOS | 7 | 28 | 7 | 3 | 0 | 8.06 | -1.06 | 15% | 13% | 15% |
| MULTIPLICAR | 3 | 2 | 1 | 1 | 1 | 4.14 | -1.14 | 38% | 28% | 38% |
| CREARINSTRUCTOR | 3 | 13 | 4 | 2 | 3 | 4.38 | -1.38 | 46% | 31% | 46% |
| EDITARINSTRUCTOR | 3 | 15 | 4 | 2 | 3 | 3.39 | -0.39 | 13% | 12% | 13% |
| BORRARINSTRUCTOR | 3 | 9 | 3 | 1 | 3 | 2.44 | 0.56 | 19% | 23% | 23% |
| CONSULTAINSTRUCTORES* | 10 | 13 | 4 | 1 | 0 | 5.69 | 4.31 | 43% | 76% | 76% |
| DETALLEINSTRUCTOR | 3 | 8 | 2 | 1 | 0 | 4.23 | -1.23 | 41% | 29% | 41% |
| REPORTEINSTRUCTORES | 5 | 18 | 5 | 2 | 0 | 7.12 | -2.12 | 42% | 30% | 42% |
| CREARDEPARTAMENTO | 4 | 8 | 2 | 2 | 1 | 5.09 | -1.09 | 27% | 21% | 27% |
| EDITARDEPARTAMENTO | 4 | 10 | 2 | 2 | 1 | 4.10 | -0.10 | 3% | 2% | 3% |
| BORRARDEPARTAMENTO | 4 | 10 | 2 | 2 | 1 | 4.10 | -0.10 | 3% | 2% | 3% |
| CONSULTADEPARTAMENTOS | 4 | 10 | 2 | 1 | 0 | 3.24 | 0.76 | 19% | 23% | 23% |
| DETALLEDEPARTAMENTO | 4 | 10 | 2 | 2 | 0 | 5.19 | -1.19 | 30% | 23% | 30% |
| CREARESTUDIANTE | 3 | 5 | 1 | 1 | 1 | 2.66 | 0.34 | 11% | 13% | 13% |
| ESTADISTICASESTUDIANTES | 4 | 1 | 1 | 1 | 0 | 5.72 | -1.72 | 43% | 30% | 43% |
| EDITARESTUDIANTE | 3 | 7 | 1 | 1 | 1 | 1.68 | 1.32 | 44% | 79% | 79% |
| BORRARESTUDIANTE | 4 | 8 | 2 | 1 | 2 | 2.06 | 1.94 | 48% | 94% | 94% |
| CONSULTAESTUDIANTES* | 6 | 14 | 3 | 3 | 0 | 7.12 | -1.12 | 19% | 16% | 19% |
| MATRICULARESTUDIANTE | 6 | 14 | 3 | 2 | 1 | 4.09 | 1.91 | 32% | 47% | 47% |
| DETALLEESTUDIANTE | 5 | 14 | 3 | 2 | 0 | 5.18 | -0.18 | 4% | 3% | 4% |
| NOTA | 6 | 12 | 3 | 2 | 1 | 5.08 | 0.92 | 15% | 18% | 18% |
| REPORTEESTUDIANTES | 4 | 15 | 3 | 2 | 0 | 4.68 | -0.68 | 17% | 15% | 17% |
| | | | | | | | MEDIANA | 26% | 23% | 27% |

Los datos del Cuadro 12 muestran que existe una mayor variación en este modelo con respecto al modelo de Contoso V1 previamente detallado, ya que una mayor cantidad de requerimientos se encuentran arriba del 40% de variación sobre su valor de referencia, teniendo al requerimiento “CREARCURSO” como el que más difiere de su valor esperado (52% MRE), seguido del requerimiento “CREARINSTRUCTOR” con un 46%. A pesar de esta condición, la mediana se mantiene en un 26% y se tiene una mayoría de

requerimientos que están por debajo del 40%, sugiriendo que las métricas del prototipo siguen siendo bastante aceptables en cuanto a la noción de tamaño funcional que sugieren. La diferencia de varianza del modelo Contoso V2 con respecto al modelo inicial de Contoso V1 puede obedecer a que existe un mayor nivel de granularidad en la definición de los requerimientos funcionales de Contoso V2, generando una mayor probabilidad de varianza entre los componentes analizados individualmente.

*Project Management Software (PMS):*

Como tercer caso de estudio se analizó una herramienta con funcionalidades básicas para la administración de proyectos y control de tareas de una compañía de desarrollo de software. Los requerimientos de dicha herramienta se listan en el Cuadro 13 y fueron obtenidos de [20].

*Cuadro 13. Requerimientos funcionales de PMS*

| Requerimiento |
|---|
| (R1.1) Crear proyecto |
| (R1.2)Asignar empleado a cargo de proyecto |
| (R1.3)Borrar empleado a cargo de proyecto |
| (R1.4)Borrar proyecto |
| (R1.5) Modificar la descripción de proyecto |
| (R1.6) Modificar la fecha inicial del proyecto |
| (R1.7)Consultar el proyecto |
| (R1.8)Asignar fecha final del proyecto |
| (R1.9)Cerrar un proyecto |
| (R2.1)Crear tarea |
| (R2.2) Borrar tarea |
| (R2.3) Modificar fecha inicial de tarea |
| (R2.4) Modificar fecha final de tarea |
| (R2.5) Modificar duración de tarea |
| (R2.6)Consultar tareas de empleado |
| (R3.1) Crear empleado |
| (R3.2) Borrar empleado |
| (R3.3) Modificar clave de empleado |
| (R3.4) Promover empleado |
| (R3.5) Degradar el tipo de empleado |
| (R3.6) Consultar empleado |
| (R3.7) Consultar tareas de empleado a cargo por intervalo de fechas |

| |
|---|
| (R3.8) Consultar tareas de empleado a cargo por proyecto |
| (R3.9) Consultar tareas de empleado a cargo por tipo de tarea |
| (R4.1) Crear tipo de proyecto |
| (R4.2) Borrar tipo de proyecto |
| (R4.3) Modificar descripción del tipo de proyecto |
| (R4.4) Consultar tipo de proyecto |
| (R4.5)Asignar tipo de proyecto |
| (R5.1) Crear tipo de tarea |
| (R5.2)Borrar tipo de tarea |
| (R5.3)Modificar descripción del tipo de tarea |
| (R5.4)Consultar tipos de tarea de un proyecto |
| (R5.5) Asignar tipo de tarea a una tarea |

Los resultados recolectados para cada uno de los requerimientos se detallan en el Cuadro 14.

*Cuadro 14. Métricas recolectadas para el análisis de PMS*

| Requerimiento | User_Interface (Herramienta) | UPF (Conteo Manual) | Columnas | Tablas | Funciones Read | Funciones Write |
|---|---|---|---|---|---|---|
| (R1.1) Crear proyecto | CREARPROYECTO | 4 | 12 | 3 | 1 | 3 |
| (R1.2)Asignar empleado a cargo de proyecto | ASIGNARPROYECTOENCARGADO | 4 | 5 | 2 | 3 | 1 |
| (R1.3)Borrar empleado a cargo de proyecto | BORRARPROYECTOENCARGADO | 4 | 3 | 2 | 2 | 1 |
| (R1.4)Borrar proyecto | BORRARPROYECTO | 4 | 4 | 2 | 1 | 2 |
| (R1.5) Modificar la descripción de proyecto | MODIFICARPROYECTODESCRIPCION | 3 | 3 | 1 | 2 | 1 |
| (R1.6) Modificar la fecha inicial del proyecto | MODIFICARPROYECTOFECHAINICIO | 3 | 3 | 1 | 2 | 1 |
| (R1.7)Consultar el proyecto | CONSULTAPROYECTOS | 5 | 3 | 2 | 5 | 0 |
| (R1.8)Asignar fecha final del proyecto | ASIGNARPROYECTOFECHAFINESTIMADA | 3 | 3 | 1 | 2 | 1 |
| (R1.9)Cerrar un proyecto | CERRARPROYECTO | 3 | 4 | 1 | 2 | 1 |
| (R2.1)Crear tarea | CREARTAREA | 3 | 11 | 3 | 1 | 3 |
| (R2.2) Borrar tarea | BORRARTAREA | 4 | 3 | 2 | 2 | 1 |
| (R2.3) Modificar fecha inicial de tarea | MODIFICARTAREAFECHAINICIO | 4 | 4 | 2 | 3 | 1 |
| (R2.4) Modificar fecha final de tarea | MODIFICARTAREAFECHAFIN | 4 | 4 | 2 | 3 | 1 |
| (R2.5) Modificar duración de tarea | MODIFICARTAREADURACION | 4 | 4 | 2 | 3 | 1 |
| (R2.6)Consultar tareas de empleado | CONSULTATAREAS | 4 | 9 | 4 | 5 | 0 |
| (R3.1) Crear empleado | CREAREMPLEADO | 3 | 5 | 1 | 1 | 1 |
| (R3.2) Borrar empleado | BORRAREMPLEADO | 3 | 2 | 1 | 1 | 1 |
| (R3.3) Modificar clave de empleado | MODIFICAREMPLEDO | 3 | 3 | 1 | 2 | 1 |
| (R3.4) Promover empleado | PROMOVEREMPLEADO | 3 | 3 | 1 | 2 | 1 |
| (R3.5) Degradar el tipo de empleado | DEGRADAREMPLEADO | 3 | 3 | 1 | 2 | 1 |
| (R3.6) Consultar empleado | CONSULTAEMPLEADOS | 3 | 5 | 1 | 3 | 0 |
| (R3.7) Consultar tareas de empleado a cargo por intervalo de fech | CONSULTATAREASFECHAS | 3 | 10 | 4 | 5 | 0 |
| (R3.8) Consultar tareas de empleado a cargo por proyecto | CONSULTATAREASPROYECTO | 4 | 10 | 4 | 5 | 0 |
| (R3.9) Consultar tareas de empleado a cargo por tipo de tarea | CONSULTATAREASTIPO | 4 | 10 | 3 | 4 | 0 |
| (R4.1) Crear tipo de proyecto | CREARTIPOSPROYECTOS | 3 | 3 | 1 | 1 | 1 |
| (R4.2) Borrar tipo de proyecto | BORRARTIPOSPROYECTOS | 3 | 2 | 1 | 1 | 1 |
| (R4.3) Modificar descripción del tipo de proyecto | MODIFICARTIPOSPROYECTOS | 3 | 3 | 1 | 2 | 1 |
| (R4.4) Consultar tipo de proyecto | CONSULTATIPOSPROYECTOS | 5 | 9 | 3 | 5 | 0 |
| (R4.5)Asignar tipo de proyecto | ASIGNARTIPOSPROYECTOS | 3 | 3 | 2 | 2 | 1 |
| (R5.1) Crear tipo de tarea | CREARTIPOSTAREAS | 3 | 5 | 1 | 1 | 1 |
| (R5.2)Borrar tipo de tarea | BORRARTIPOSTAREAS | 3 | 2 | 1 | 1 | 1 |
| (R5.3)Modificar descripción del tipo de tarea | MODIFICARTIPOSTAREAS | 3 | 3 | 1 | 2 | 1 |
| (R5.4)Consultar tipos de tarea de un proyecto | CONSULTATIPOSTAREAS | 3 | 3 | 2 | 3 | 0 |
| (R5.5) Asignar tipo de tarea a una tarea | ASIGNARTIPOSTAREAS | 4 | 4 | 3 | 3 | 1 |

Una vez recolectados los datos, se realizó el análisis de correlación múltiple de manera homóloga a los demás casos de estudio, obteniendo el siguiente resultado según el Cuadro 15.

*Cuadro 15. Cálculo de correlación múltiple para PMS*

| Regression Statistics | |
|---|---|
| Multiple R | 0.724 |
| **R Square** | **0.524** |
| Adjusted R Square | 0.459 |
| Standard Error | 0.452 |
| Observations | 34.000 |

Al tomar estos datos como referencia, se realizó de igual forma el análisis de exactitud que se presenta en el Cuadro 16.

*Cuadro 16. Análisis de exactitud para PMS*

| User_Interface (Herramienta) | UFP (Conteo Manual) | Columnas | Tablas | Funciones Read | Funciones Write | Valor Estimado | Diferencia | MRE | MER | BRE |
|---|---|---|---|---|---|---|---|---|---|---|
| CREARTAREA | 3 | 11 | 3 | 1 | 3 | 3.63 | -0.63 | 21% | 17% | 21% |
| BORRARTAREA | 4 | 3 | 2 | 2 | 1 | 3.47 | 0.53 | 13% | 15% | 15% |
| MODIFICARTAREAFECHAINICIO | 4 | 4 | 2 | 3 | 1 | 3.87 | 0.13 | 3% | 3% | 3% |
| MODIFICARTAREAFECHAFIN | 4 | 4 | 2 | 3 | 1 | 3.87 | 0.13 | 3% | 3% | 3% |
| MODIFICARTAREADURACION | 4 | 4 | 2 | 3 | 1 | 3.87 | 0.13 | 3% | 3% | 3% |
| CONSULTATAREAS | 4 | 9 | 4 | 5 | 0 | 4.21 | -0.21 | 5% | 5% | 5% |
| CREAREMPLEADO | 3 | 5 | 1 | 1 | 1 | 2.67 | 0.33 | 11% | 12% | 12% |
| BORRAREMPLEADO | 3 | 2 | 1 | 1 | 1 | 2.91 | 0.09 | 3% | 3% | 3% |
| MODIFICAREMPLEDO | 3 | 3 | 1 | 2 | 1 | 3.31 | -0.31 | 10% | 9% | 10% |
| PROMOVEREMPLEADO | 3 | 3 | 1 | 2 | 1 | 3.31 | -0.31 | 10% | 9% | 10% |
| DEGRADAREMPLEADO | 3 | 3 | 1 | 2 | 1 | 3.31 | -0.31 | 10% | 9% | 10% |
| CONSULTAEMPLEADOS | 3 | 5 | 1 | 3 | 0 | 3.08 | -0.08 | 3% | 3% | 3% |
| CONSULTATAREASFECHAS | 3 | 10 | 4 | 5 | 0 | 4.13 | -1.13 | 38% | 27% | 38% |
| CONSULTATAREASPROYECTO | 4 | 10 | 4 | 5 | 0 | 4.13 | -0.13 | 3% | 3% | 3% |
| CONSULTATAREASTIPO | 4 | 10 | 3 | 4 | 0 | 3.49 | 0.51 | 13% | 15% | 15% |
| CREARTIPOSPROYECTOS | 3 | 3 | 1 | 1 | 1 | 2.83 | 0.17 | 6% | 6% | 6% |
| BORRARTIPOSPROYECTOS | 3 | 2 | 1 | 1 | 1 | 2.91 | 0.09 | 3% | 3% | 3% |
| MODIFICARTIPOSPROYECTOS | 3 | 3 | 1 | 2 | 1 | 3.31 | -0.31 | 10% | 9% | 10% |
| CONSULTATIPOSPROYECTOS | 5 | 9 | 3 | 5 | 0 | 4.05 | 0.95 | 19% | 23% | 23% |
| ASIGNARTIPOSPROYECTOS | 3 | 3 | 2 | 2 | 1 | 3.47 | -0.47 | 16% | 14% | 16% |
| CREARTIPOSTAREAS | 3 | 5 | 1 | 1 | 1 | 2.67 | 0.33 | 11% | 12% | 12% |
| BORRARTIPOSTAREAS | 3 | 2 | 1 | 1 | 1 | 2.91 | 0.09 | 3% | 3% | 3% |
| MODIFICARTIPOSTAREAS | 3 | 3 | 1 | 2 | 1 | 3.31 | -0.31 | 10% | 9% | 10% |
| CONSULTATIPOSTAREAS | 3 | 3 | 2 | 3 | 0 | 3.40 | -0.40 | 13% | 12% | 13% |
| ASIGNARTIPOSTAREAS | 4 | 4 | 3 | 3 | 1 | 4.03 | -0.03 | 1% | 1% | 1% |
| | | | | | | | MEDIANA | 10% | 9% | 10% |

Para este modelo en específico, los valores arrojados por el análisis de exactitud son muy positivos manteniendo una mediana para la métrica MRE de un 10%. El valor más elevado se registra en el requerimiento “CONSULTATAREASFECHAS” con un 38%, y el valor que le sigue es un 21% (requerimiento “CREARTAREA”). Este modelo presenta los valores más aceptables dentro de los casos estudiados, y esto sugiere que las métricas del prototipo son un buen indicador de tamaño funcional. La exactitud al analizar este modelo puede deberse también a que no fue necesario realizar ningún tipo de

agrupamiento o conciliación entre los requerimientos del conteo de UFP y los requerimientos del archivo de entrada, ya que estaban completamente homologados entre sí.

*Venta de Vehículos Usados (Santana):*

Finalmente se utilizó como caso de estudio una aplicación de un tamaño mucho menor a las anteriores. Dicha herramienta consta de tres requerimientos principales y fue desarrollada para automatizar el proceso de venta de vehículos en una empresa de compra y venta de vehículos usados. Los requerimientos del sistema se listan en el Cuadro 17 y fueron obtenidos de [20].

*Cuadro 17. Requerimientos funcionales de Santana*

| Requerimiento |
|---|
| (R1) Consulta de Vehículos |
| (R2) Consulta de Detalle de Vehículo |
| (R3) Solicitud de Compra de Vehículo |

Los requerimientos con sus respectivos conteos de UFP, así como los resultados de las métricas arrojados por la herramienta prototipo se detallan en el Cuadro 18.

*Cuadro 18. Métricas recolectadas para el análisis de Santana*

| Requerimiento | User_Interface (Herramienta) | UFP (Conteo Manual) | Columnas | Tablas | Funciones Read | Funciones Write |
|---|---|---|---|---|---|---|
| (R1) Consulta de Vehículos | form_query_cars | 16 | 7 | 2 | 5 | 0 |
| (R2) Consulta de Detalle de Vehículo | form_query_car | 3 | 7 | 2 | 2 | 0 |
| (R3) Solicitud de Compra de Vehículo | form_buy_car | 4 | 21 | 3 | 3 | 2 |

Al contar con los datos recolectados en el Cuadro 18, se procedió a realizar el análisis de correlación, el cual generó los siguientes resultados mostrados en el Cuadro 19.

*Cuadro 19. Cálculo de correlación múltiple para Santana*

| Regression Statistics | |
|---|---|
| Multiple R | 1 |
| **R Square** | **1** |
| Adjusted R Square | 65535 |
| Standard Error | 0 |
| Observations | 3 |

Seguidamente se realizó el correspondiente análisis de exactitud, cuyos resultados son presentados en el Cuadro 20.

*Cuadro 20. Análisis de exactitud para Santana*

| User_Interface (Herramienta) | UFP (Conteo Manual) | Columnas | Tablas | Funciones Read | Funciones Write | Valor Estimado | Diferencia | MRE | MER | BRE |
|---|---|---|---|---|---|---|---|---|---|---|
| form_query_cars | 16 | 7 | 2 | 5 | 0 | 16.00 | 0.00 | 0% | 0% | 0% |
| form_query_car | 3 | 7 | 2 | 2 | 0 | 3.00 | 0.00 | 0% | 0% | 0% |
| form_buy_car | 4 | 21 | 3 | 3 | 2 | 4.00 | 0.00 | 0% | 0% | 0% |
| | | | | | | | MEDIANA | 0% | 0% | 0% |

Los resultados de este análisis señalan que para los tres requerimientos de la aplicación Santana, se tiene una métrica MRE de 0%. Esto significa que no existe diferencia entre los valores estimados y los valores de referencia para cada requerimiento. Se tiene una relación directa entre el conteo de UFP y los elementos transaccionales afectados según las métricas del prototipo. Es importante destacar que este análisis de exactitud se ve afectado por la poca cantidad de requerimientos que posee la aplicación, ya que existe una mayor probabilidad de variación con una mayor cantidad de elementos a considerar dentro del cálculo de correlación.

Luego de analizar los datos resultantes de los casos de estudio aplicados se concluye que el modelo funcional resultó un indicador eficaz del tamaño funcional de las aplicaciones que se visualizaron. Además, los elementos visuales del grafo ayudaron también a dar una noción del tamaño funcional y permiten tener un mejor entendimiento de la funcionalidad de la herramienta de software que se está modelando, ofreciendo la posibilidad de analizar los componentes relacionados con cada requerimiento funcional de la aplicación analizada.

Otro factor a considerar es que los requerimientos que se ajustaron (marcados con asterisco cada una de las tablas análisis de exactitud, Cuadros 12, 16 y 20) no mostraron una diferencia determinante en los resultados, lo que sugiere que el agrupamiento de requerimientos realizado es necesario, y gracias al uso de la herramienta visual para verificar cuáles de los requerimientos separados en el conteo manual formaban parte de un mismo requerimiento en la visualización (o sea, formaban parte de una pantalla de interfaz de usuario) fue factible.

# Capítulo 7. Conclusiones y trabajo futuro

En esta investigación se desarrolló un prototipo de herramienta de predicción y visualización de software, cuyo modelo es basado en grafos y se evaluó su eficacia para predecir métricas de tamaño y complejidad. Dicha herramienta permite generar la representación visual de un sistema de software a partir de su modelo funcional, el cual es la salida de un prototipo para la medición de tamaño funcional presentado en [20]. De esta forma, el prototipo de visualización interpreta el modelo de entrada, genera el modelo de visualización, y calcula y despliega las métricas de representación de tamaño obtenidas a partir del modelo generado.

Se evaluaron empíricamente las métricas obtenidas del modelo funcional generadas por la herramienta desarrollada, con el fin de determinar la eficacia para estimar el tamaño funcional y la complejidad del sistema de software bajo análisis mediante los casos de estudio presentados. Los resultados finales de cada caso de estudio sugieren que las métricas extraídas del modelo funcional pueden utilizarse como un indicador del tamaño funcional de la aplicación de entrada. Además, los elementos visuales del grafo ayudan también a dar una mejor noción de tamaño funcional ya que permite percibir claramente la cantidad de elementos afectados por un requerimiento específico y su tamaño.

Dentro de los aportes de esta investigación pueden mencionarse:

- La definición de una especificación de requerimientos para una herramienta de visualización de software basada en modelos funcionales de grafos.
- El desarrollo de un prototipo de herramienta de visualización de software basada en grafos y el cálculo automático de métricas sobre el modelo.
- La evaluación de la eficacia de las métricas del modelo funcional para la estimación del tamaño funcional.

Para finalizar, como trabajo futuro se puede considerar extender la cantidad de métricas que la herramienta calcula para considerar también funciones de datos, adicional a las funciones transaccionales ya utilizadas. Otro aspecto que se podría mejorar en la

herramienta es el componente visual que despliega el grafo, ya que para efectos de la investigación se utilizó la biblioteca de software libre *alchemy,js*, la cual puede considerarse como limitada si se compara con otros componentes de pago en el mercado. Un componente visual más funcional y flexible podría mejorar la experiencia del usuario y, por consiguiente, facilitaría las labores de comprensión y análisis. Finalmente, sería posible la exploración de modelos alternativos a la regresión múltiple para el análisis de los resultados obtenidos mediante los casos de estudio, así como también evaluar la usabilidad de la herramienta propuesta en esta investigación, y comparar su eficiencia contra herramientas existentes o prácticas estandarizadas de un grupo de usuarios que necesiten estimar el tamaño y la complejidad de las aplicaciones de software.

# Referencias

[1] Robert Dąbrowski, Krzysztof Stencel, y Grzegorz Timoszuk. "Software is a directed multigraph." *European Conference on Software Architecture*. Springer Berlin Heidelberg, 2011.

[2] Christian Quesada López y Marcelo Jenkins. “Un estudio sobre las prácticas de la ingeniería del software en Costa Rica: Resultados preliminares”. *CITIC.* Universidad de Costa Rica, 2017

[3] Standish Group. "CHAOS manifesto 2013." *The Standish Group International.* EUA, 2011.

[4] Michael J. Pacione, Marc Roper, y Murray Wood. "A novel software visualisation model to support software comprehension." *Proceedings of the 11th Working Conference on Reverse Engineering*. IEEE, 2004.

[5] Vianna Ferreira, Paulo José Azevedo, y Márcio de Oliveira Barros. "Traceability between function point and source code." *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*. ACM, 2011.

[6] Danny B. Lange y Yuichi Nakamura. “Object-Oriented Program Tracing and Visualization”. *IBM Tokyo Research Laboratory*. IEEE, 1997.

[7] Wim De Pauw, Erik Jensen, Nick Mitchell, Gary Sevitsky, John Vlissides, y Jeaha Yang. "Visualizing the execution of Java programs." *Software Visualization*. Springer Berlin Heidelberg, 2002.

[8] Claes Wohlin, Per Runeson, Martin Host, Magnus C. Ohlsson, Bjorn Regnell y Anders Wesslén. “Experimentation in software engineering”. *Springer Science & Business Media*, 2012.

[9] Duarte, L. M., Maia, P. H. M., & Silva, A. C. S. “Extraction of probabilistic behaviour models based on contexts”. In *2018 IEEE/ACM 10th International Workshop on Modelling in Software Engineering (MiSE)* (págs. 25-32). IEEE, 2018.

[10] Duarte, L. M., Kramer, J., & Uchitel, S. “Towards faithful model extraction based on contexts”. *International Conference on Fundamental Approaches to Software Engineering* (pp. 101-115). Springer, Berlin, Heidelberg, 2008

[11] Washizaki, H., Guéhéneuc, Y. G., & Khomh, F. “ProMeTA: a taxonomy for program metamodels in program reverse engineering”. *Empirical Software Engineering*, *23*(4), 2323-2358, 2018

[12] Diehl, S. “Software visualization: visualizing the structure, behaviour, and evolution of software”. *Springer Science & Business Media*, 2007

[13] Gračanin, D., Matković, K., & Eltoweissy, M. “Software visualization”. *Innovations in Systems and Software Engineering*, *1*(2), 221-230, 2005

[14] Fittkau, F., Finke, S., Hasselbring, W., & Waller, J. “Comparing trace visualizations for program comprehension through controlled experiments”. *2015 IEEE 23rd International Conference on Program Comprehension* (págs. 266-276). IEEE, 2015

[15] Wettel, R., & Lanza, M. “Visualizing software systems as cities”. *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis* (págs. 92-99). IEEE, 2007.

[17] Baum, D., Schilbach, J., Kovacs, P., Eisenecker, U., & Müller, R. “GETAVIZ: generating structural, behavioral, and evolutionary views of software systems for empirical evaluation”. *2017 IEEE Working Conference on Software Visualization (VISSOFT)* (págs. 114-118). IEEE, 2017.

[18] Brito, R., Brito, A., Brito, G., & Valente, M. T. “GoCity: Code City for Go”. *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (págs. 649-653). IEEE, 2019

[19] IEEE. “IEEE Recommended Practice for Software Requirements Specifications”. USA, 2009.

[20] Quesada López, C. “Automatización de la medición del tamaño funcional del software para modelos funcionales obtenidos a partir del análisis dinámico del código fuente”. Universidad de Costa Rica, 2018

[21] Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., & Wesslen, A. “Experimentation in Software Engineering”. Springer, 2012.

[22] Basili, V., Caldiera, C., & Rombach, H. “Goal Question Metric Paradigm. Encyclopedia of Software Engineering” (págs. 520-532). John Wiley and Sons, 1994.

[23] Salas Villalobos, L. “Medición del tamaño funcional en el desarrollo de software dirigido por modelos”. Universidad de Costa Rica, 2018.

[24] Quesada-López, C. & Jenkins, M. “Procedimientos de medición del tamaño funcional: un mapeo sistemático de literatura.” *Proceedings of the XX Ibero-American Conference on Software Engineering (CibSE 2017)*. Buenos Aires, Argentina, 22-23, 2018.

[25] Wieringa, R. “Design Science Methodology for Information Systems and Software Engineering”. Berling: Springer, 2014.

[26] Lakoff, G. “The contemporary theory of metaphor”. UC Berkeley, 1993.

[27] Bondy, J. A., & Murty, U. S. R. “Graph theory with applications”. Gran Bretaña, 1976.