Instituto Tecnológico de Costa Rica

Vicerrectoría de Investigación y Extensión

Dirección de Proyectos

Área de Ingeniería en Computadores

Informe final de proyecto de investigación y extensión
Documento 1

Proyecto: Diseño de métodos de analítica visual (AV) en el contexto de Big Data para apoyar el proceso de desarrollo y mantenimiento de software (AVIB)

Código del proyecto: 1320060

Investigador responsable: Jennier Solano Cordero

Investigadores participantes: Antonio González Torres, José Navas Sú, Marco Hernández Vásquez y Franklin Hernández Castro

Período del proyecto: 01/01/2018 – 30/06/2021

Contenido

1 Código del proyecto 3

2 Título del proyecto 3

3 Autores y direcciones 3

4 Resumen 4

5 Introducción 4

6 Marco Teórico 7

7 Metodología 9

8 Resultados 10

9 Discusión y conclusiones 16

10 Recomendaciones 19

11 Agradecimientos 19

12 Referencias 19

1 Código del proyecto

El código del proyecto es el 1320060.

2 Título del proyecto

El proyecto se denomina Diseño de métodos de analítica visual (AV) en el contexto de Big Data para apoyar el proceso de desarrollo y mantenimiento de software (AVIB)

3 Autores y direcciones

El periodo ordinario de duración del proyecto fue de 36 meses, del 1 de enero de 2018 al 31 de diciembre del 2020, y los participantes son los investigadores que se detallan en la Tabla 1. Sin embargo, posteriormente se concedió una extensión al proyecto por 6 meses adicionales, entre el 1 de enero y el 30 de junio del 2021 (ver Tabla 2). El coordinador del proyecto tanto en el periodo ordinario como de ampliación fue el Dr. Jennier Solano Cordero.

Tabla 1. Participación de investigadores durante el periodo ordinario del proyecto.

Nombre y apellidos	Escuela o Área Académica	Nombramiento (definido e indefinido)	Jornada (h/semana)	Meses en el proyecto	Tipo de plaza (VIE, REC, DOC, CONS)
Dr. Jennier Solano Cordero	Ingeniería en computadores	Indefinido	4	36	VIE
Dr. Antonio González Torres	Ingeniería en computadores	Indefinido	16	36	VIE
M.Sc. Marco Hernández Vásquez	Ingeniería en computadores	Indefinido	10	36	VIE
Dr. Franklin Hernández Castro	Ingeniería en Diseño Industrial	Indefinido	3	36	Rec
M.Sc. José Navas Sú	Ingeniería en Computación	Definido	24	36	VIE

Tabla 2. Participación de investigadores durante el periodo de ampliación del proyecto.

Nombre y apellidos	Escuela o Área Académica	Nombramiento (Definido e indefinido)	Jornada (h/semana)	Meses en el proyecto	Tipo de plaza (VIE, REC, DOC, CONS)
Dr. Jennier Solano Cordero	Ingeniería en computadores	Indefinido	6	6	REC
Dr. Antonio González Torres	Ingeniería en computadores	Indefinido	6	6	REC

Nombre y apellidos	Escuela o Área Académica	Nombramiento (Definido e indefinido)	Jornada (h/semana)	Meses en el proyecto	Tipo de plaza (VIE, REC, DOC, CONS)
M.Sc. Marco Hernández Vásquez	Ingeniería en computadores	Indefinido	6	6	REC
M.Sc. José Navas Sú	Ingeniería en Computación	Definido	6	6	REC

4 Resumen

El desarrollo y mantenimiento del software son procesos complejos que producen un gran número de cambios y grandes volúmenes de datos en la forma de líneas de código, variables, relaciones de acoplamiento, cohesión, herencia e implementación de interfaces. El volumen de datos se multiplica por el número de revisiones del sistema que se cuentan por miles después de unos pocos meses de evolución de un sistema mediano o grande. Estos datos cumplen con las propiedades de Big Data, y requiere el uso de enfoques novedosos para transformarlos en conocimiento. El análisis automático del código fuente facilita la toma de decisiones para efectuar cambios a los sistemas y promover su mantenibilidad. Como consecuencia, el objetivo de esta investigación consistió en diseñar métodos para efectuar el análisis del código fuente del software. Los resultados obtenidos contemplan la revisión detallada de bibliografía sobre el análisis avanzado de código utilizando técnicas de análisis estático y machine learning, la definición de técnicas de minería de repositorios de software y el diseño de un framework para calcular métricas complejas. La metodología de investigación que se utilizó fue una adaptación del modelo de *Investigación-Acción* (Kemmis, 2005). El desarrollo del proyecto y los productos obtenidos son satisfactorios, pero además dejan amplias enseñanzas para continuar realizando investigación en la misma línea. A lo que se suma el conocimiento acumulado por la amplia participación de profesores, estudiantes asistentes, de maestría y doctorado.

Palabras clave

Desarrollo y mantenimiento de software, calidad de software, métricas de software, minería de repositorios de software

5 Introducción

En la tesis de doctorado titulada “Evolutionary Visual Software Analytics” (González-Torres, 2015) y otras investigaciones realizadas por Antonio González Torres (González-Torres 2013a;

González-Torres 2013b y González-Torres, 2016) se discutió y analizó el proceso de evolución de software y la complejidad que conlleva el desarrollo de grandes sistemas, los cuales con frecuencia tienen las siguientes características:

1. Los diseños no registran todos los detalles, la documentación se encuentra incompleta o desactualizada.
2. Se componen de miles de componentes e incluso millones de líneas de código.
3. Los desarrolladores pueden trabajar en diferentes localidades geográficas, en diversos países, y hablan diferentes idiomas.
4. Se realizan miles de cambios en un día que pueden afectar a cientos o incluso miles de componentes.
5. No se registran formalmente el detalle de los cambios en la documentación de los proyectos.
6. El desarrollo se extiende por varios años, y cuando son puestos en producción su vida útil se puede extender por más de 10 años.
7. El costo del desarrollo y mantenimiento puede superar los cientos de miles o incluso millones de dólares.
8. El costo de un sistema durante su vida útil está conformado por, aproximadamente, un 10% asociado al desarrollo y un 90% relacionado con el mantenimiento.

Las investigaciones realizadas por González Torres determinaron mediante un extenso análisis bibliográfico y una encuesta que respondieron 69 profesionales de 5 países, que durante los procesos de desarrollo y mantenimiento de software no se están usando métodos para dar seguimiento a los efectos que producen los cambios en las relaciones de herencia, implementación de interfaces, acoplamiento y cohesión entre los elementos que componen un sistema de software, ni para comprender el impacto de esos cambios en la calidad medida por métricas de complejidad, mantenibilidad y susceptibilidad a pruebas. De forma adicional, esos trabajos determinaron que se requiere investigación adicional para diseñar métodos que permitan:

1. Construir de forma automática la representación de la arquitectura de los sistemas.
2. Comparar los cambios efectuados, determinar las partes de los sistemas que han sido afectadas por dichos cambios y quién los ha realizado.
3. Realizar el cálculo de métricas de calidad para determinar si los cambios efectuados han contribuido de forma positiva o negativa con la calidad de un sistema, y por lo tanto si comprometen sus posibilidades futuras de mantenimiento.
4. Comparar las métricas de calidad entre versiones, las contribuciones de cada programador y determinar los cambios en la estructura del sistema.
5. Identificar las relaciones de herencia, implementación de interfaces, acoplamiento, cohesión entre los elementos que componen el sistema y cómo los cambios afectan a cada elemento.
6. Encontrar copias del mismo código en diferentes partes del sistema para ayudar a optimizar el proceso de mantenimiento.
7. Facilitar la colaboración entre los programadores y líderes de proyectos al brindar información sobre los cambios que se están realizando al sistema, quién está efectuando esos cambios y a qué hora del día se realizan.

Sin embargo, el diseño e implementación de cualquier método en ese sentido, requiere efectuar el análisis de grandes volúmenes de información, que se derivan de los cambios a los sistemas, utilizando técnicas y métodos para identificar los patrones y relaciones que permitan la creación de conocimiento útil para realizar cambios adicionales, mejorar la calidad y reducir la complejidad de los sistemas. Los resultados de los métodos existentes de análisis de datos, aunque reducen la dimensionalidad y complejidad de la información, usualmente son voluminosos, complejos y difíciles de analizar por parte de un ser humano.

El análisis de los sistemas de software utiliza técnicas de minería de repositorios de software para extraer detalles sobre los cambios, pero los resultados que se obtienen no ofrecen los detalles necesarios para construir el conocimiento que requieren los desarrolladores y administradores de proyectos para fabricar software bajo principios de producción industrial. Esto ha motivado el análisis de grandes y complejos conjuntos de datos, e incentivado la realización de diversas investigaciones. Sin embargo, las investigaciones realizadas cuentan con ciertas limitaciones. Por esa razón, la investigación realizada por González Torres significó una importante contribución a la ingeniería de software por la definición de un proceso que utiliza de forma intensiva varias visualizaciones y la interacción entre estas. Sin embargo, se requiere de investigación adicional para diseñar métodos adicionales, lo cual da origen al proyecto AVIB.

La investigación del proyecto AVIB realizó aportes significativos a un problema no resuelto, tanto desde un punto de vista teórico como aplicado. En el contexto institucional, esta propuesta supuso una investigación innovadora y de impacto científico porque impulsa el mejoramiento de los procesos de desarrollo de software en la industria nacional, que tiene un alto impacto en el empleo de los egresados del TEC. A lo que se suma el aprendizaje acumulado por los estudiantes e investigadores del proyecto, así como de aquellos que asistieron a las charlas y conferencias que se llevaron a cabo.

El objetivo general de la investigación gira en torno a diseñar métodos para realizar el análisis del código del software con el fin de determinar el impacto de los cambios en la calidad medida por métricas de complejidad, mantenibilidad y susceptibilidad a pruebas. En función de este objetivo se definieron los siguientes objetivos específicos:

1. Efectuar un estudio con desarrolladores y líderes de proyectos en empresas de software y departamentos de desarrollo para determinar los requerimientos prácticos que se deben considerar para comprender los efectos que producen los cambios en los sistemas.
2. Identificar y comparar las herramientas que se utilizan en la industria en las diferentes fases del ciclo de vida de los sistemas y los datos que estas recogen sobre los cambios a los sistemas.
3. Diseñar una metodología para la minería de repositorios de software a partir de los requerimientos de los desarrolladores y líderes de proyectos, y los datos disponibles, de acuerdo con las herramientas que se utilizan en las diferentes fases del ciclo de vida de los sistemas.
4. Diseñar un framework de análisis para métricas de complejidad, mantenibilidad y susceptibilidad a pruebas de los sistemas.
5. Diseñar una metodología de analítica visual para el análisis de información derivada de la evolución de software.
6. Efectuar un caso de estudio para analizar múltiples proyectos de software, con la participación de programadores y líderes de proyectos de las empresas colaboradoras.
7. Realizar un Simposio internacional y charlas.

La metodología que se utilizó es una adaptación del modelo de Investigación-Acción (Kemmis, 2005), que contempló:

- La determinación de requerimientos y análisis.
- La definición de la metodología de minería de repositorios de software.
- El framework de métricas complejas.
- La preparación de un caso de estudio.

6 Marco Teórico

Los sistemas de software son utilizados en un gran número de dispositivos que se encuentran casi omnipresentes en la vida diaria de personas y empresas. Los individuos usan estos dispositivos y el software asociado para trabajar, aprender (tomar cursos en línea, leer e investigar temas de interés), entretenerse (jugar, ver televisión o videos, escuchar música), comunicarse (con amigos, familia, compañeros de trabajo, participar en foros, colaborar y reuniones de trabajo), comprar, hacer trámites (banca, impuestos y otros pagos) y teletrabajo. Esto es una realidad social presente futura, de la cual dependen social y económicamente personas y organizaciones.

Sin embargo, el desarrollo y mantenimiento de los sistemas de software es complejo y la posibilidad de fracaso está presente en todas las etapas y niveles del proceso (Mohagheghi, 2017; Coelho, 2017; Taherdoost y Keshavarzsaleh, 2015). Esto lo ponen en evidencia los informes de evaluación del desempeño general de la industria de software y las tasas de éxito de proyectos particulares que se publican todos los años (The Standish Group, 2016). Dichos informes muestran que con frecuencia un gran número de proyectos ocasionan pérdidas por millones de dólares y la erosión del prestigio de las organizaciones involucradas. Por esa razón, se han realizado grandes esfuerzos para mejorar el levantamiento de requerimientos (Arnaut et al., 2016), el cálculo de costos y riesgos (Tavares et al., 2017), la planificación (Shmueli et al., 2016), el diseño (Robillard, 2016), el desarrollo y el mantenimiento de los sistemas (Pang y Hindle, 2016).

Como parte de los esfuerzos por mejorar el desarrollo, prueba y mantenimiento de software se ha impulsado el análisis de los cambios que realizan los programadores al código fuente, con el fin de comprender el impacto que producen en la complejidad, mantenibilidad y susceptibilidad a pruebas de los programas (i.e., posibilidad de ser probado). Este análisis tiene por fin brindar información a los programadores y líderes de proyectos sobre el estado de los sistemas para que puedan realizar cambios adicionales de forma satisfactoria.

La realización de cambios en el software se basa en la comprensión del estado actual de un sistema, el cual es la acumulación de los cambios que han sido realizados con anterioridad. Los cambios que realiza un desarrollador a un elemento de software pueden afectar a una serie de elementos de software relacionados en los que están trabajando otros programadores, por las relaciones que existen entre estos: el elemento C puede utilizar al elemento B, mientras que este podría usar al elemento A. Adicionalmente, dos o más programadores pueden estar haciendo cambios al mismo elemento de software, de forma simultánea (González-Torres 2015).

Lo anterior se vuelve más crítico cuando se tiene en consideración que los programadores cambian con frecuencia de trabajo debido a la alta demanda que existe en el mercado y la competencia entre las empresas por atraer el mejor talento (Zylka y Fischbach, 2017). Esto obliga que las empresas contraten personal de forma constante y reasignen los programadores a

diferentes proyectos, quienes tienen que hacer frente al desarrollo y mantenimiento de aplicaciones en las cuales no habían trabajado antes. Como resultado, la evolución de los sistemas se ve comprometida con frecuencia, no solo por los constantes cambios de personal, sino también debido a la falta de documentación adecuada del sistema, la cual con frecuencia está incompleta, obsoleta o no existe.

En general, los programadores y líderes de proyectos necesitan contar con información que les permita comprender los detalles de los proyectos en que participan, la evolución de estos y los cambios que se les han realizado, en plazos muy cortos, para llevar a cabo los cambios o tareas de mantenimiento más urgentes. En este contexto, la investigación de doctorado realizada por González Torres proporciona las bases de esta investigación con la extensa revisión que realizó y que comprendió más de 300 referencias bibliográficas sobre los tipos de análisis más utilizados (González-Torres 2015).

Los tipos de análisis más utilizados, de acuerdo con la investigación mencionada, son los siguientes:

1. Análisis estático de código.
2. Análisis de metadatos.
3. Análisis de las trazas de ejecución de los sistemas.
4. Extracción de métricas de software.
5. Métodos de recuperación de información.
6. Análisis de relaciones sociotécnicas.
7. Análisis de fuentes de datos enlazadas.
8. Particionamiento de programas y arquitecturas.
9. Análisis de origen y refactorización.

Los resultados de los análisis mencionados son utilizados para comprender los cambios y los efectos que estos tienen en las relaciones entre los elementos de los programas. De acuerdo con el detallado trabajo de González Torres, estas representaciones permiten:

1. Detectar problemas de diseño.
2. Comprender el funcionamiento de sistemas distribuidos.
3. Obtener detalles sobre la calidad del software.
4. Entender la seguridad del código fuente.
5. Estudiar la asignación de memoria a las estructuras de datos de los programas.
6. Conocer sobre la ejecución de los sistemas multihilos.
7. Interpretar el análisis de la ejecución de los programas paralelos.
8. Comprender el desempeño de los sistemas.
9. Interpretar la ejecución de los sistemas.
10. Obtener detalles sobre el diseño y modelo de los sistemas a partir del código fuente
11. Estudiar los ecosistemas de los programas.
12. Facilitar la portación, reutilización y depurado de código.
13. Apoyar la ingeniería inversa.
14. Analizar la cobertura de las pruebas de los sistemas.
15. Interpretar y comprender las relaciones entre los elementos del sistema.
16. Facilitar la colaboración y el conocimiento sobre las tareas que realizan los programadores de un sistema.
17. Entender los cambios y las dependencias.
18. Comprende la arquitectura de los sistemas.

7 Metodología

En este apartado se debe detallar la manera en que se efectuó el estudio. En esta sección se describen todos los materiales y metodologías utilizadas, incluyendo el diseño experimental. Se debe incluir:

- Población y muestra del estudio.
- Diseño de investigación.
- Métodos, técnicos e instrumentos de investigación.
- Procedimientos de recolección de información.
- Diseño de procesamiento y análisis de datos.

La naturaleza de la investigación involucró la utilización del código fuente de once proyectos de software, entre los que se cuentan cinco proyectos proporcionados por dos empresas colaboradoras que se dedican al desarrollo de software y seis proyectos de código libre. Los proyectos utilizados se muestran en la Tabla 3, en la cual los nombres del software proporcionado por las empresas se encuentran anonimizados y listados como Sistema 1 … Sistema 6. El código fuente se utilizó para validar los métodos y efectuar pruebas.

Los proyectos de código libre que fueron analizados se tomaron de Github un repositorio de software en el cual más de 73 millones de programadores almacenan sus proyectos de software de forma pública o privada. La URL de Github es https://github.com. El código de los proyectos y los metadatos asociados fueron extraídos utilizando un módulo que fue creado durante el desarrollo del proyecto de forma específica para ese fin.

Tabla 3. Proyectos analizados durante la investigación

Nombre del sistema	Clases	Métodos	Líneas de código
Sistema 1	842	2.832	17.020
Sistema 2	2.633	20.362	188.463
Sistema 3	3.507	18.525	177.050
Sistema 4	1.563	3.786	29.077
Sistema 5	336	1.065	6.063
Sistema 6	251	825	4.603
MongoDB C# Driver	1.415	8.034	37.181
Json.NET	167	1.574	10.185
.NET Micro Framework	1.824	9.038	67.633
NodeJS Tools	596	3.288	18.507
Neo4j .NET Driver	210	1.018	3.403

Los repositorios de Github de donde se tomaron los proyectos de libre son los que se muestran a continuación:

- MongoDB C# Driver: https://github.com/mongodb/mongo-csharp-driver

- Json.NET: https://github.com/JamesNK/Newtonsoft.Json
- .NET Micro Framework: https://github.com/NETMF
- NodeJS Tools: https://github.com/microsoft/nodejstools
- Neo4j .NET Driver : https://github.com/neo4j/neo4j-dotnet-driver

8 Resultados

La determinación de requerimientos y análisis se realizó mediante una encuesta y entrevistas con programadores y líderes de proyectos de una empresa colaboradoras, para determinar los requerimientos de las metodologías y el framework mencionado. El objetivo de esta encuesta fue obtener información sobre las herramientas que se usan en las diferentes etapas del ciclo de desarrollo, para identificar los datos que se recolectan de los cambios, definir el método de minería de repositorios de software, diseñar un framework para la recolección de métricas varios métodos de analítica visual para el análisis de información derivada del del análisis de software.

Una vez que se determinaron los requerimientos se diseñó la arquitectura del framework, la cual se muestra en la Figura 1. Este proceso se divide varias grandes etapas que involucran la minería de repositorios de software, la transformación del código de diversos lenguajes en un lenguaje generalizado, el cálculo de métricas complejas y el uso de analítica visual para apoyar el estudio de las estructuras de datos resultantes. La versión inicial de la arquitectura y el planteamiento general de la investigación publicado por la revista ecuatoriana Enfoque UTE en el artículo “A visual analytics architecture for the analysis and understanding of software systems”. Además, también se publicó dicho planteamiento en el artículo “A Proposal towards the Design of an Architecture for Evolutionary Visual Software Analytics” que apareció en los proceedings de la IEEE 2018 International Conference on Information Systems and Computer Science (INCISCOS).

El detalle de las fases mencionadas que incluye los componentes que la conforman se encuentra a continuación:

Minería de repositorios de software: Esta fase contempla la conexión a los repositorios de software y recupera el código fuente, escrito en cualquier lenguaje para el cual se cuenta con soporte durante el proceso de transformación entre lenguajes. Este proceso contempla obtener tanto el código y el metalenguaje. Esta fase requiere los parámetros de conexión (conexión URL, tipo de servidor, credenciales) a los repositorios de software y luego pasa el código posteriormente al proceso de transformación de lenguajes particulares a un lenguaje genérico.

Transformación de lenguajes particulares a uno genérico: Se desconoce la cantidad de lenguajes de programación que existen en la actualidad, debido a las características dinámicas de la ingeniería de software y la aparición constante de nuevos lenguajes. Sin embargo, el estudio realizado por Nanz y Furia, basado en el análisis de 7.087 programas que resolvieron 745 problemas diferentes, identificó que los lenguajes más utilizados son C, Java, C#, Python, Go, Haskell, F \ # y Ruby.

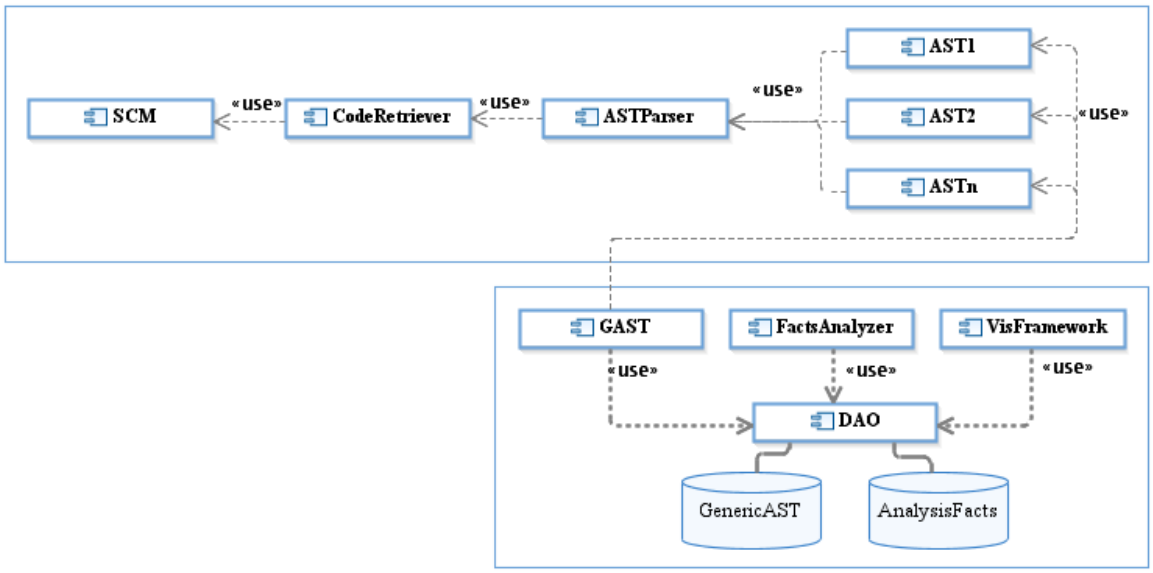


Figura 1. Arquitectura de análisis avanzado de código fuente.

El desarrollo de aplicaciones de software modernas utiliza artefactos escritos en varios idiomas, especialmente en proyectos grandes. La razón es aprovechar las características específicas de cada idioma para producir sistemas más eficientes, efectivos y completos. Sin embargo, esto hace que el desarrollo sea más desafiante porque aumenta la complejidad de las tareas de programación y requiere desarrolladores con habilidades en más de un idioma. La complejidad radica en la especificidad de la sintaxis de cada idioma.

Por lo tanto, los equipos de desarrollo necesitan utilizar herramientas que les ayuden durante las tareas de desarrollo y mantenimiento. Analizar la estructura y las relaciones entre los elementos de un sistema es un desafío que aumenta cuando un programador debe analizar programas en diferentes lenguajes de programación.

Sin embargo, el diseño de métodos para el análisis de código fuente en diferentes lenguajes de programación que pertenezcan a diversos paradigmas y tengan diferente sintaxis requiere la creación de un analizador gramatical específico para cada lenguaje. Por lo tanto, crear un analizador individual para cada lenguaje no es un enfoque práctico considerando los muchos lenguajes existentes y la aparición frecuente de otros nuevos. Por lo que una alternativa es implementar un único analizador que funcione sobre una representación genérica de lenguajes. Lo que tiene como consecuente que este analizador recopile datos para el cálculo de métricas, los componentes internos del software y las relaciones entre los elementos de un sistema. En consecuencia, esto puede contribuir con la reducción de costos y la necesidad de agregar analizadores para cada lenguaje.

Una representación genérica de lenguajes debe contemplar todos los aspectos sintácticos de los lenguajes representados y debe ser escalable para permitir la adición de nuevos lenguajes. Además, debe ser flexible para adaptarse a las características cambiantes de los lenguajes. Por lo que en esta fase se diseñó un método para transformar automáticamente la sintaxis de lenguajes particulares en un lenguaje con una sintaxis genérica. El diseño captura las características individuales de cada lenguaje para permitir el análisis de proyectos de software.

En consecuencia, en esta investigación se implementó un árbol de sintaxis abstracta genérica (GAST) que tiene una estructura equivalente para múltiples lenguajes de programación. El

método se probó a través de dos experimentos, que muestran la viabilidad de transformar varios lenguajes al GAST y realizar diferentes tipos de análisis en su estructura. Estos resultados se exponen en el artículo “GAST: A generic AST representation for language-independent source code analysis” que fue sometido a la conferencia “29th IEEE International Conference onSoftware Analysis, Evolution and Reengineering” que realizará la notificación de aceptación el 16 de diciembre del 2021.

El proceso se muestra en las Figuras 2 y 3, mientras que el mapeo entre lenguajes específicos y el lenguaje genérico, y su correspondiente validación se ilustra en la Figura 3. La transformación que se ilustra en la arquitectura de la Figura 1 la realizan los elementos ASTParser, los parsers específicos de cada lenguaje denotados por AST1, AST2 … ASTn y el GAST.

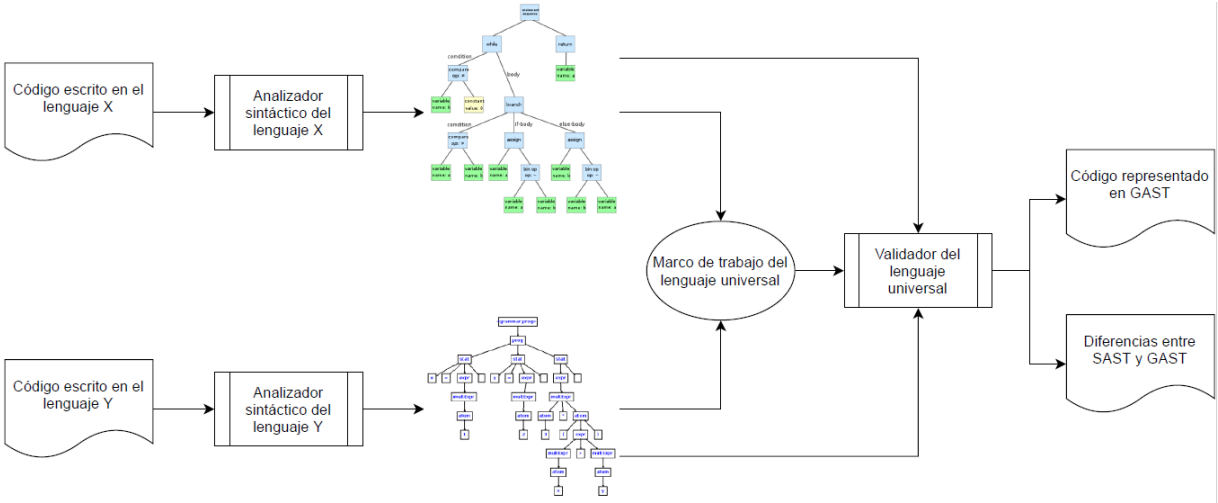


Figura 2. Proceso de mapeo de lenguajes especifícos al lenguaje genérico.

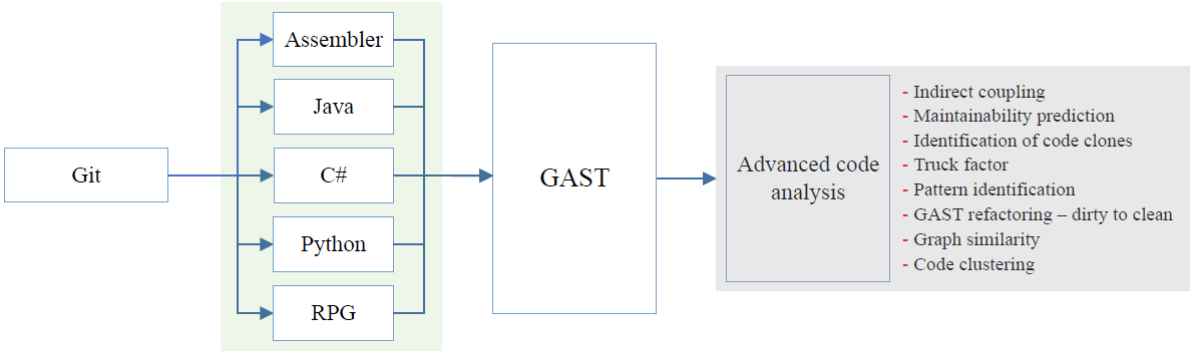


Figura 3. Proceso general de transformación análisis.

Framework para la obtención de métricas complejas: En esta fase se realiza el análisis del árbol genérico abstracto y se extraen los diversos tipos de métricas. Esta fase puede tomar como entrada directamente el GAST o de un servidor de base de datos en donde el GAST se almacena en formato JSON. Entre las métricas que se extraen están las básicas que se describen en la literatura (e.g., LOC y NOM), otras más complejas (e.g., CYCLO y Halstead) y las que se proponen como resultado de la investigación y que conforman un conjunto de métricas complejas relacionadas con el acoplamiento indirecto.

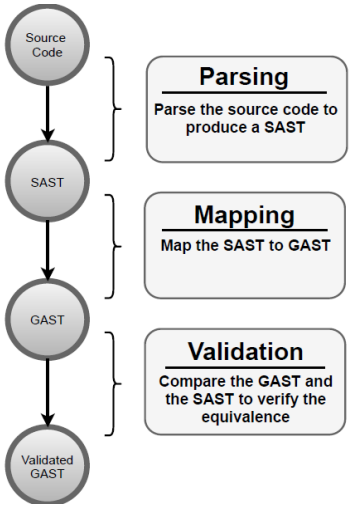
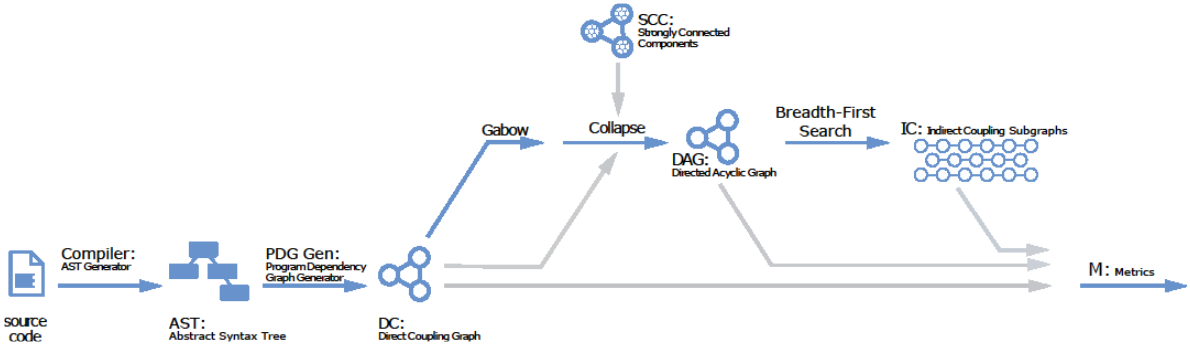


Figura 4. Mapeo entre los lenguajes específicos y un lenguaje genérico, así como la validación de esa transformación.

Los resultados de esta fase se exponen en los artículos "A Method to Extract Indirect Coupling and Measure Its Complexity", "GAST: A generic AST representation for language-independent source code analysis" y "Measuring Indirect Coupling Complexity and Size of Software Systems", en donde el primero apareció en los proceedings de IEEE 2018 International Conference on Information Systems and Computer Science (INCISCOS) y los dos últimos fueron sometidos a la conferencia "29th IEEE International Conference on Software Analysis, Evolution and Reengineering" que realizará la notificación de aceptación el 16 de diciembre del 2021.



Métodos de analítica visual: Esta fase tiene por fin utilizar visualizaciones y técnicas de interacción persona-computadora para representar visualmente los datos de una manera accesible para que los humanos los entiendan en un corto período de tiempo. Este componente permitiría decodificar datos y transformarlos en conocimiento.

Durante esta fase se diseñaron e implementaron varios métodos de forma parcial, los cuales se espera publicar durante los próximos meses. El diseño general para la visualización de clones de código se muestra en la Figura 5, mientras que el detalle de la visualización como un plugin de Eclipse se encuentra en la Figura 6. La implementación de esta visualización se completó en un 60%.

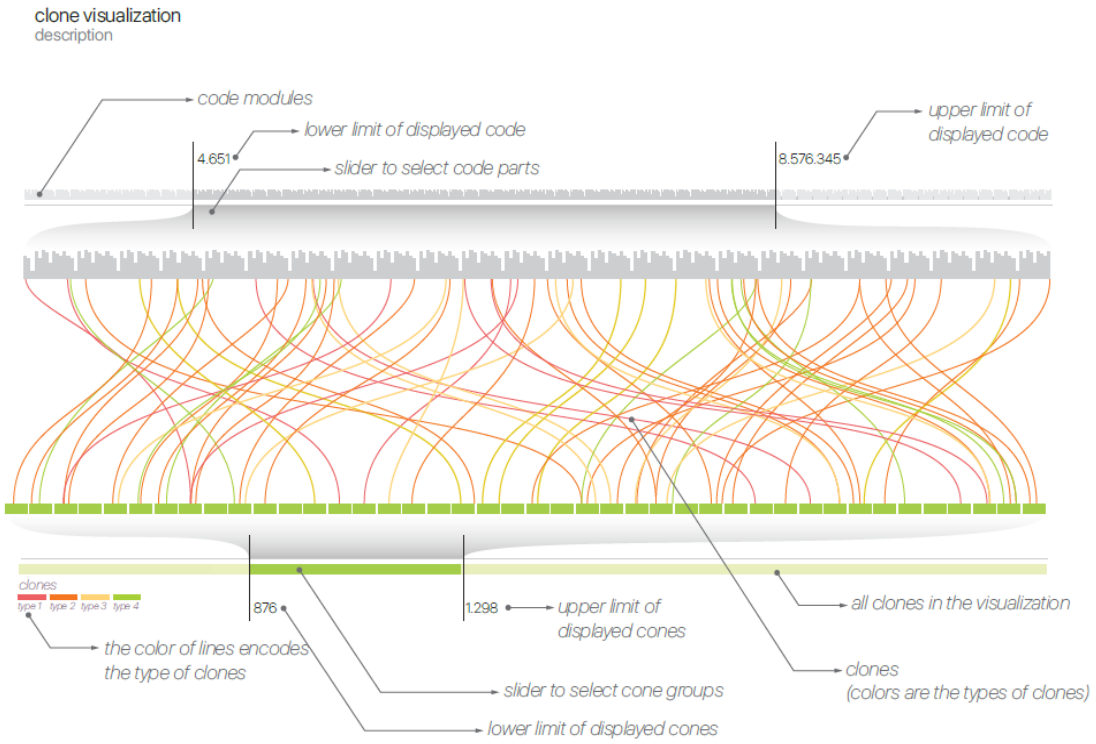


Figura 5. Diseño de la visualización de clones.

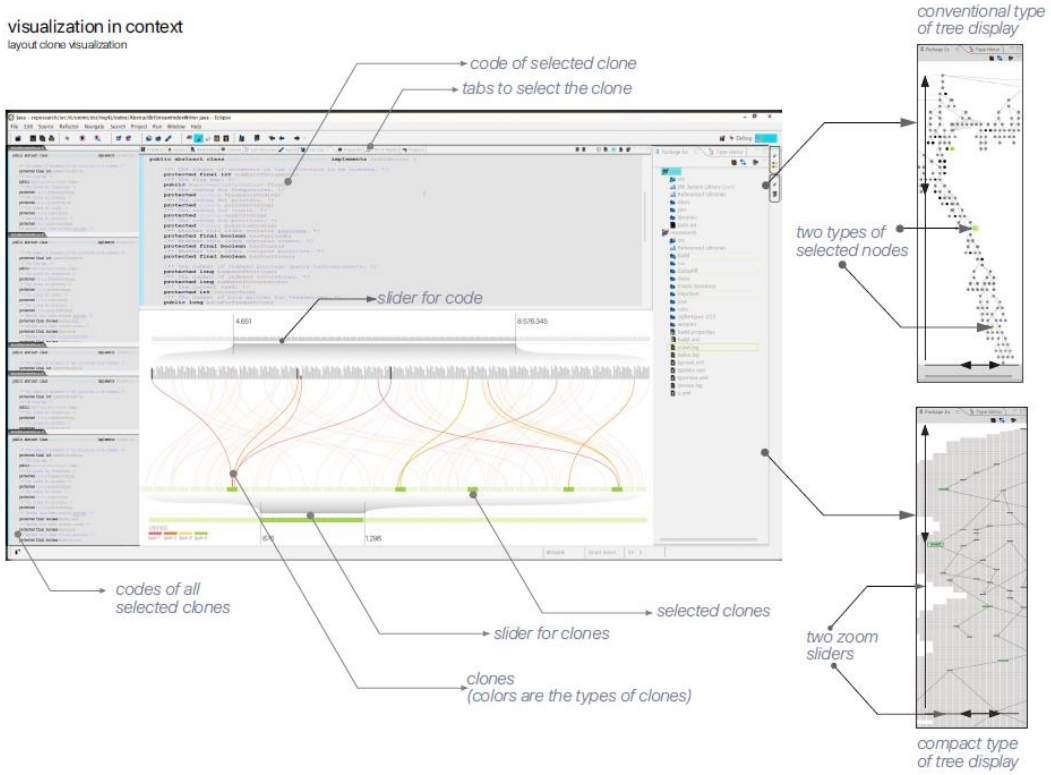


Figura 6. Visualización de clones como un plugin de Eclipse.

En tanto, los diseños que se muestran en las Figuras 7, 8 y 9 corresponden al acoplamiento indirecto entre elementos de código, la combinación del acoplamiento indirecto de código con el acoplamiento lógico, y la combinación de los dos anteriores con el acoplamiento lógico que se deriva de los clones de código. Los datos que se utilizan para estas visualizaciones son los que se obtienen en la fase **Framework para la obtención de métricas complejas**. El avance de la implementación de esta visualización se completó en un 70%.

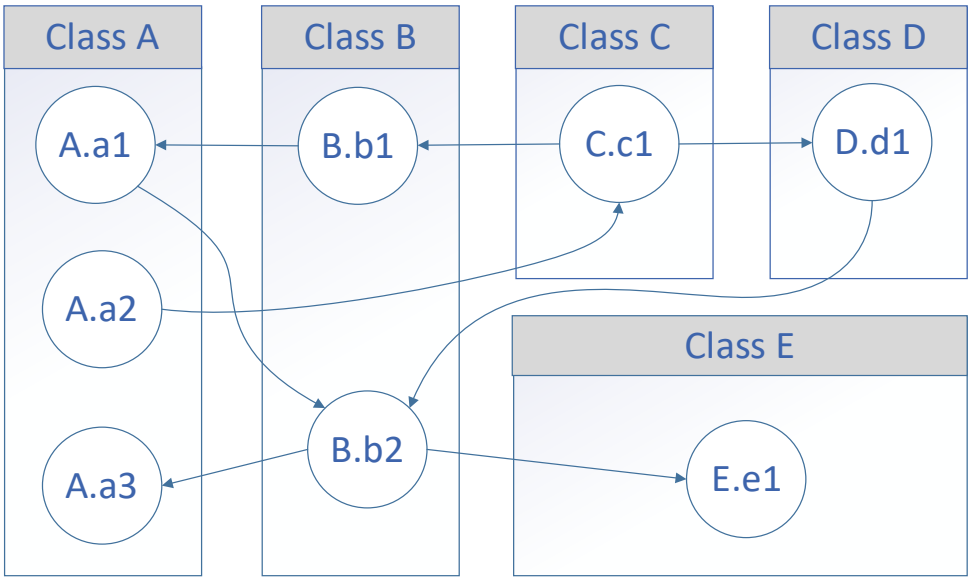


Figura 7. Acoplamiento directo entre métodos.

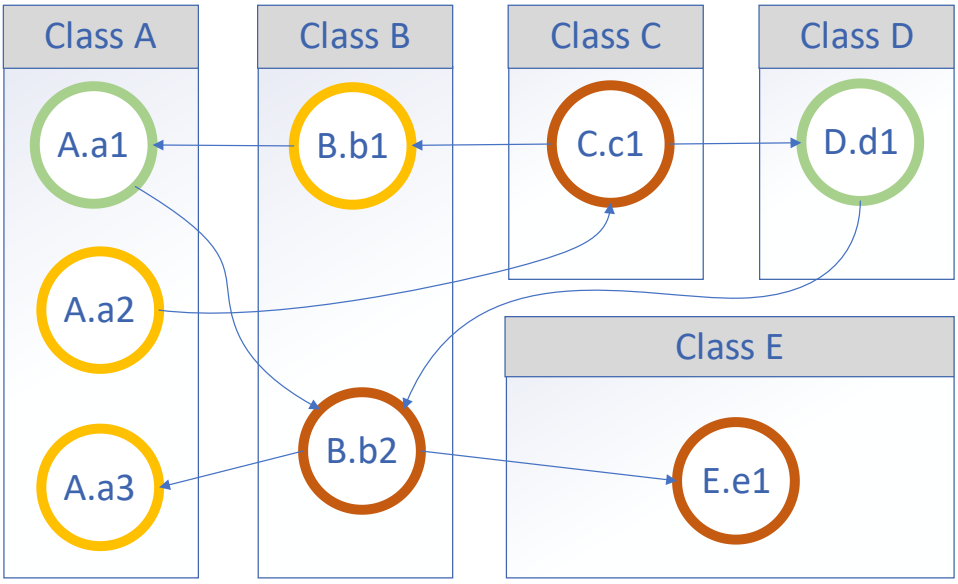


Figura 8. Acomplamiento indirecto y lógico denotado por el borde de los nodos.

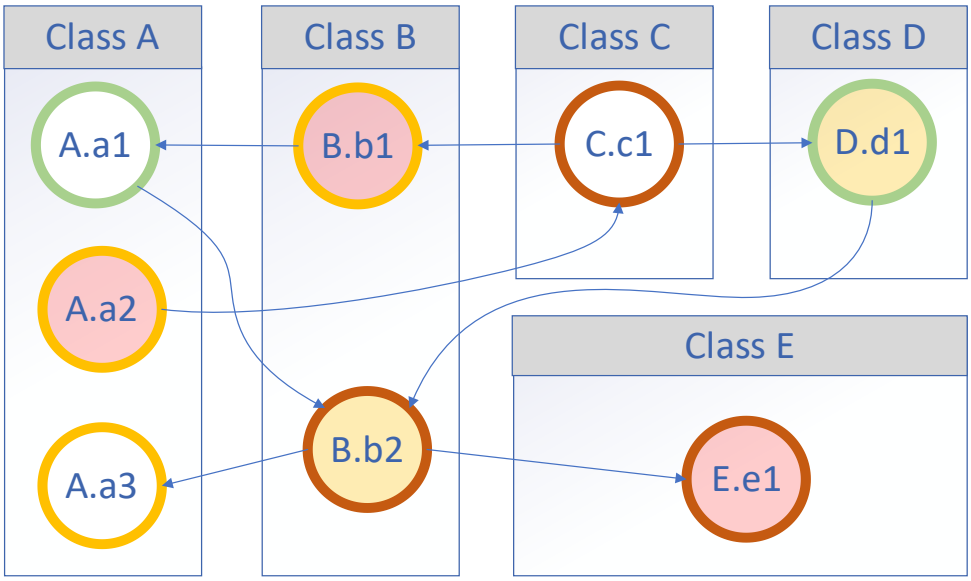


Figura 9. Combinación del acoplamiento indirecto, lógico y el acoplamiento lógico que se deriva de los clones de código.

El trabajo relacionado con los acoplamientos fue inspirado por un trabajo previo que se incluyó en un artículo titulado "Acoplamiento estructural y acoplamiento lógico" que se encuentra pendiente de publicación.

En el caso de ambas visualizaciones, una vez que finalizada su implementación se requiere efectuar su validación con programadores, lo cual requiere el contacto directo con las personas. Esto último es un impedimento en las circunstancias de la pandemia por el COVID-19, pero se espera poder efectuarlo tan pronto como la implementación esté concluida.

En cuanto al caso de estudio, así como el análisis de los resultados de este se no se pudo realizar debido a la emergencia causada por la pandemia del COVID-19 debido a que se requiere contar con participantes con conocimiento y experiencia para validar con casos reales de la industria la utilidad de los métodos y el framework propuesto.

9 Discusión y conclusiones

Los desarrolladores y gerentes de proyectos necesitan comprender el software que están desarrollando y manteniendo, cuando no tienen conocimiento previo o documentación de esos sistemas. Esta situación adquiere mayor importancia con el hecho de que la evolución del software es un proceso que suele durar varios años y produce datos que comparten muchas de las características típicas del Big Data. Por lo tanto, las capacidades de los programadores y gerentes de proyectos son particularmente limitadas cuando necesitan analizar grandes proyectos y no pueden extraer información útil.

El uso de métodos avanzados de análisis de código es una alternativa práctica debido a sus ventajas para transformar grandes volúmenes de datos en conocimiento utilizando un enfoque de embudo potenciado mediante análisis avanzados y automáticos, representaciones visuales y las habilidades de los humanos para detectar patrones y tomar decisiones. Sin embargo, no hay evidencia del uso de análisis visual en la industria y el uso de herramientas visuales simples es

limitado. Esta investigación ha considerado tal factor y que el análisis del código fuente es un proceso no trivial que necesita métodos y técnicas probados y validados en otras áreas para sustentar el razonamiento analítico y la toma de decisiones. de una arquitectura para definir frameworks basados en el proceso de analítica de software visual evolutivo.

La arquitectura se definió tomando como base la investigación previa realizada para la definición de Maleku y los requerimientos de los programadores y jefes de proyecto de las empresas colaboradoras. Además, esta investigación incorpora nuevos métodos para el análisis de acoplamiento indirecto, clones de código, dependencias de programas, así como visualizaciones novedosas para apoyar el razonamiento analítico. El papel de las empresas es clave para validar resultados y viabilizar la transferencia de conocimiento y generar mayor impacto en la sociedad. En consecuencia, los resultados de esta investigación serán validados por las empresas asociadas y los profesionales de otras organizaciones, lo cual, lamentablemente ha sido afectado por la pandemia ocasionada por el COVID-19.

El análisis del código fuente es una de las formas más importantes para que los desarrolladores evalúen la calidad de un producto de software. Sin embargo, analizar el código requiere desarrollar métricas para cada lenguaje, ya que corresponden a la sintaxis específica de cada uno en particular. Por lo tanto, en el transcurso de esta investigación se realizó el diseño e implementación de una estructura universal basada en el estándar MOF 2.0.

Entre los resultados que no estaban contemplados en la propuesta original del proyecto se encuentra el proceso de transformación de código de un lenguaje específico a la estructura universal, lo cual permite representar múltiples árboles de sintaxis abstracta de lenguajes de programación específicos en una sola estructura. Esto permite contar con un framework escalable que permite homogeneizar el análisis de software. De esta manera, solo es necesario implementar nuevas métricas una vez para todos los lenguajes y, por lo tanto, con la adición de nuevos lenguajes a la estructura universal en el futuro, las métricas implementadas se pueden reutilizar, aumentando la extensibilidad.

Debe tenerse en cuenta que la transformación de un AST específico de un lenguaje al GAST depende estrictamente de la estructura de ese AST. Lo anterior implica que el proceso de mapeo es diferente para cada uno, y es necesario describir las reglas de equivalencia entre cada elemento sintáctico. Por lo tanto, la verificación del mapeo correcto de las reglas de equivalencia utiliza un método automático para asegurar la transformación de todos los elementos sintácticos en el GAST para cada archivo de proyecto.

Los resultados obtenidos mostraron que la validación del mapeo es una tarea que requiere tiempo. Sin embargo, representa una ventaja porque los módulos que usan el GAST no tienen que verificar la integridad de los elementos sintácticos debido a la verificación realizada al construir el transformador.

Además, esta investigación definió e implementó un conjunto de métricas utilizando grafos de acoplamiento indirecto con granularidad a nivel de método. Esta suite se basa en la teoría de la medición, así como en los puntos de vista de desarrolladores de software experimentados. También fue validado con criterios estándar y cumplió con todas las propiedades requeridas. Además de proporcionar métricas probadas y fundamentadas teóricamente, esta investigación también las aplicó a un conjunto de proyectos de software industrial y de código abierto escritos en C#. También brindó sugerencias sobre el uso de estas métricas para el mantenimiento del software.

El uso de este conjunto de métricas es de gran utilidad, por ejemplo, un gerente de proyecto experimentado podría usar la métrica FNOM para estimar el tiempo y el esfuerzo necesarios

para comprender y mantener la funcionalidad detrás de un método y comparar estos requisitos de recursos en todos los métodos e incluso en diferentes sistemas. También puede seleccionar aquellos métodos que tienen más probabilidades de verse afectados por errores o cambios introducidos en una de sus dependencias y asignar los recursos necesarios para probar la funcionalidad modificada.

Este análisis también podría ayudar a decidir si refactorizar o no después de investigar la posibilidad de deuda técnica. Las métricas FLOC y FCYC también podrían ayudar para propósitos similares, ya que producen clasificaciones diferentes para los mismos métodos en un sistema que FNOM. Podrían ayudar a identificar métodos más extensos o complejos que otros en términos de líneas de código o complejidad ciclomática.

El uso combinado de las tres métricas es el enfoque recomendado para aprovechar el análisis de la fragilidad en un sistema. La comparación de la fragilidad promedio de los métodos entre sistemas permite a los gerentes de proyecto decidir sobre la asignación de recursos a cada uno de ellos. Además, los valores altos de fragilidad también podrían servir como un indicador de menor reutilización, y la razón es que el código más prominente y complejo suele ser más específico de la aplicación. Además, el subgrafo de fragilidad de los métodos para cambiar podría ayudar a comprender la funcionalidad bajo mantenimiento.

Las métricas de riesgo están más relacionadas con medir el grado de reutilización y el tamaño y complejidad del subgrafo de riesgo de los métodos en un sistema. Las tres métricas RNOM, RLOC y RCYC clasifican los métodos de manera diferente y podrían combinarse para detectar aquellos con un código dependiente más extenso o complejo que otros. Los métodos de utilidad con valores de alto riesgo deben cambiarse y probarse con más cuidado que los de bajo riesgo.

El esfuerzo requerido para resolver una solicitud de mantenimiento podría estimarse identificando los métodos objetivo y analizando sus métricas de fragilidad y riesgo. Las métricas de fragilidad miden cuánto código necesitan rastrear y comprender los programadores y qué tan complejo es. Por el contrario, las métricas de riesgo muestran el tamaño y la complejidad del código que el cambio podría tener un impacto potencial. Tanto los subgrafos de fragilidad como de riesgo de los métodos involucrados dan una idea cercana de los caminos que necesitarán ser evaluados. Además, los métodos con alta fragilidad y alto riesgo deben tratarse con la mayor precaución, ya que comprenderlos y actualizarlos podría requerir más esfuerzo, y el impacto potencial de este cambio y cualquier error introducido es significativo.

Es importante mencionar, que los resultados de esta investigación están siendo utilizados en la investigación en se encuentra curso como parte del proyecto “Metodología para el reconocimiento automático de patrones del Pensamiento Computacional en estudiantes de la educación general básica para mejorar los procesos de gestión”. Estos resultados, el conocimiento acumulado y la participación de estudiantes e investigadores que participaron durante el desarrollo del proyecto al que corresponde este informe final están contribuyendo a la producción de buenos resultados, que tendrán un notable impacto a nivel nacional.

A lo anterior se debe agregar que el profesor Antonio González Torres está desarrollando cinco proyectos con estudiantes de la maestría en computación que utilizan los resultados del proyecto. A esto se debe agregar que el profesor José Navas Sú de la Escuela de Computación se encuentra realizando su doctorado asociado a este proyecto con el Dr. González como su director de tesis, y que al menos dos candidatos están considerando ingresar al doctorado para realizar sus investigaciones asociadas a los resultados del proyecto trabajando con el mismo profesor.

Acompañando este informe se incluye el Anexo 1 – Resumen del Proyecto AVIB, en el cual se ofrecen información adicional a la aquí expuesta y en concordancia con la propuesta originalmente presentada y aceptada.

10 Recomendaciones

Los resultados de este proyecto pueden ser utilizados para ampliar el conjunto de métricas extraído con nuevas métricas más complejas derivadas de las anteriores y también para procesar sistemas de software escritos en otros lenguajes de programación además de C#. Por ejemplo, las métricas propuestas por Halstead podrían usarse para crear nuevas métricas de fragilidad y riesgo en cada método, por ejemplo, Halstead Fragility (FHAL) y Halstead Riskiness (RHAL). Esta extensión podría explorar modelos para combinar métricas que lleven a la creación de métricas compuestas, por ejemplo, la combinación lineal de varias métricas asignando pesos a cada una.

Además, otros trabajos futuros pueden incluir la posibilidad de realizar el proceso inverso: escribir código fuente a partir del árbol de sintaxis abstracta genérica y generar código en un lenguaje específico. Por se podría generar código en Java, C# y Python a partir de la estructura GAST. Esto permitiría la creación de un transpilador de idiomas de varios a varios.

Los resultados de esta investigación también pueden usarse para analizar código de malware. Por lo que se recomienda efectuar la descompilación de archivos binarios para obtener el código ensamblador correspondiente y transformarlo al GAST. Luego, la se puede aplicar un algoritmo de búsqueda para identificar patrones de código asociados a código malicioso para la detección temprana de malware.

11 Agradecimientos

Los investigadores agradecemos de forma especial el apoyo brindado por la Vicerrectora de Investigación y Extensión (VIE), así como al Área de Ingeniería en Computadores por la disposición para impulsar el trabajo de investigación realizado.

12 Referencias

[1] Arnaut, B. M., Ferrari, D. B., & e Souza, M. L. D. O. (2016, October). A requirement engineering and management process in concept phase of complex systems. In Systems Engineering (ISSE), 2016 IEEE International Symposium on (pp. 1-6). IEEE Press.

[2] Coelho, J., & Valente, M. T. (2017). Why Modern Open Source Projects Fail. arXiv preprint arXiv:1707.02327.

[3] Cruz, A., Bastos, C., Afonso, P., & Costa, H. (2016). Software visualization tools and techniques: A systematic review of the literature. In Computer Science Society (SCCC), 2016 35th International Conference of the Chilean (pp. 1-12). IEEE.

[4] González-Torres, A., Garcia-Peñalvo, F. J., Therón Sánchez, R., & Colomo-Palacios R. (2016) Knowledge discovery in software teams by means of evolutionary visual software analytics, Journal of Science of Computer Programming, doi: 10.1016/j.scico.2015.09.005, ISSN: 0167-6423 (Impact Factor: 0.715).

[5] González-Torres, A. (2015). Evolutionary Visual Software Analytics (Ph.D. thesis, University of Salamanca, Spain, 2015) (pp. 1-294). University of Salamanca. url: http://hdl.handle.net/10366/128154

[6] González-Torres, A., Therón Sánchez, R. & García-Peñalvo, F.J. (2013). How evolutionary visual software analytics supports knowledge discovery, Journal of Information Science and Engineering, Institute of Information Science, Academia Sinica, Taiwan, ISSN: 1016-2364 (Impact Factor: 0.333)

[7] González-Torres, A., Therón Sánchez, R. & García-Peñalvo, F.J. (2013). Human–computer interaction in evolutionary visual software analytics, Computers in Human Behavior, Elsevier, The Netherlands, doi:10.1016/j.chb.2012.01.013, ISSN: 0747-5632 (Impact Factor: 2.273).

[8] Lovelock, J. D., Hahn, W. L., Atwal, R., Gupta, N., Blackmore, D., & O'Connell, A. (2017, julio). Forecast Alert: IT Spending, Worldwide, 2Q17 Update (Rep. No. G00324598). Retrieved July 05, 2017, from Gartner Inc. website: http://gtnr.it/2vbygQI

[9] Mohagheghi, P., & Jørgensen, M. (2017, mayo). What contributes to the success of IT projects?: success factors, challenges and lessons learned from an empirical study of software projects in the Norwegian public sector. In Proceedings of the 39th International Conference on Software Engineering Companion (pp. 371-373). IEEE Press.

[10] Pang, C., & Hindle, A. (2016, October). Continuous Maintenance. In Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on (pp. 458-462). IEEE.

[11] Robillard, M. P. (2016, November). Sustainable software design. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 920-923). ACM.

[12] Shmueli, O., Pliskin, N., & Fink, L. (2016). Can the outside-view approach improve planning decisions in software development projects?. Information Systems Journal, 26(4), 395-418.

[13] Salameh, H. B., Ahmad, A., & Aljammal, A. (2016). Software evolution visualization techniques and methods-a systematic review. In Computer Science and Information Technology (CSIT), 2016 7th International Conference on (pp. 1-6). IEEE.

[14] Taherdoost, H., & Keshavarzsaleh, A. (2015). A Theoretical Review on IT Project Success/Failure Factors and Evaluating the Associated Risks. In 14th International Conference on Telecommunications and Informatics, Sliema, Malta.

[15] Tavares, B. G., da Silva, C. E. S., & de Souza, A. D. (2017). Risk management analysis in Scrum software projects. International Transactions in Operational Research.

[16] The Standish Group. The Chaos Manifesto, 2016.

[17] Zylka, M. P., & Fischbach, K. (2017). Turning the Spotlight on the Consequences of Individual IT Turnover: A Literature Review and Research Agenda. ACM SIGMIS Database: the DATABASE for Advances in Information Systems, 48(2), 52-78.

[18] Kemmis, S & McTaggart, R. The sage handbook of qualitative research (3rd ed.), chapitre Participatory Action Research: Communicative Action and the Public Sphere., pages 559–603. Sage Publications Ltd, 2005.