Proyecto de Trabajo Profesional
Ciudad Autónoma de Buenos Aires, 18 de Junio de 2020

Sr. Director
Del Departamento de Computación
De la Facultad de Ingeniería de la Universidad de Buenos Aires
Lic. Pablo Cosso De mi mayor consideración:

Por la presente me dirijo a Usted a fin de elevar a su consideración el acta donde consta mi conformidad y la del estudiante Sbruzzi, José Ignacio (Padrón N°: 97452) en la que se acuerda el Tema de su Proyecto de Trabajo Profesional de Ingeniería Informática "CallCluster : Extracción, análisis y visualización de callgraphs".

Prof. Dr. Mariano Méndez

Sr. Sbruzzi, José Ignacio

Proyecto de Trabajo Profesional

Profesor Prof. Dr. Mariano Méndez deja constancia que en su opinión, al haber elegido el Sr. Sbruzzi, José Ignacio las asignaturas de la Orientación, de colocar la orientación de la Carrera de Ingeniería en informática y sus correspondientes electivas, se puede dar por cumplida la elaboración del Plan de Estudio Personal. Deja constancia que los contenidos de las asignaturas de la orientación abarcan los conocimientos necesarios para desarrollar satisfactoriamente el trabajo profesional. Sin más que tratar, se deja por concluida la reunión firmándose

tres ejemplares de la presente acta, uno para elevar a la Direcci´on del Departamento de Computaci´on, otro para los estudiantes y el tercero para el profesor.

Prof. Dr. Mariano Méndez

Sr. Sbruzzi, José Ignacio

CALLCLUSTER: EXTRACCIÓN, ANÁLISIS Y VISUALIZACIÓN DE CALLGRAPHS

Propuesta de TRABAJO PROFESIONAL

Autor:
SBRUZZI, JOSÉ IGNACIO - Padrón 97452

jose.sbru@gmail.com

Tutor:
PROF. DR. MARIANO MÉNDEZ

Facultad de Ingeniería - Universidad de Buenos Aires
Junio 2020

Acta Acuerdo para el desarrollo de Trabajo Profesional.

En la Ciudad Autónoma de Buenos Aires, al décimo octavo día del mes de Junio del año dos mil veinte se reúne en el Departamento de Computación de la Facultad de Ingeniería de la Universidad de Buenos Aires el profesor Dr. Mariano Méndez, con el estudiante de la carrera de Ingeniería en Informática Sr. Sbruzzi, José Ignacio (Padrón N°: 97452) para tratar la elección y acuerdo del tema de Trabajo Profesional para el Ciclo Superior de la carrera.

Teniendo en cuenta la propuesta presentada por los alumnos más las observaciones y mejoras propuestas por el profesor, se ha acordado el plan de trabajo para el desarrollo e implementación del proyecto "CallCluster: extracción, análisis y visualización de callgraphs" que figura en el documento adjuntado. El acuerdo consiste en las siguientes pautas:

1. El alumno Sr. Sbruzzi, José Ignacio realizará todas las etapas para la construcción e implementación del sistema.

2. El trabajo a realizar será presentado para cumplir los requisitos de la materia Trabajo Profesional.

3. El profesor Dr. Mariano Méndez acepta la función de tutor para dicho trabajo.

Dr. Méndez, Mariano

Sr. Sbruzzi, José Ignacio

Índice

1. Introducción **3**
1.1. Motivación 3
1.2. Estado del arte 3

2. Objetivo **6**
2.1. Subobjetivos 6

3. Diseño de la solución **7**
3.1. Arquitectura de la solución 7
3.2. Requerimientos funcionales de la solución 8
3.2.1. Requerimientos funcionales de los analizadores 8
3.2.2. Requerimientos funcionales del visualizador y analizador 8
3.3. Tecnologías y conocimientos a utilizar 9

4. Plan de trabajo **12**
4.1. Alcance 12
4.2. Metodología de trabajo 12
4.3. Estimación 13
4.4. Cronograma 16
4.5. Descripción de los entregables 17
4.5.1. Propuesta 17
4.5.2. Investigación sobre algoritmos de clustering 17
4.5.3. Investigación y relevamiento de librerías de visualización de grafos 17
4.5.4. Extractor C# 18
4.5.5. Extractor C 18
4.5.6. Visualizador 18
4.5.7. Documentación del proyecto 18
4.5.8. Presentación 18

5. Glosario **19**

6. Referencias **19**

1. Introducción

A partir de la aplicación de ciencia de datos y visualizaciones novedosas a temas de Ingeniería de Software, se ideó una nueva herramienta que será útil para juzgar y entender la estructura de proyectos de software en diversos niveles de abstracción.

1.1. Motivación

El hecho de que haya tan poco esfuerzo invertido en herramientas abiertas de análisis de código es un factor muy importante a tener en cuenta. Existen muy pocas herramientas de análisis de código fuente, y aún menos que sean abiertas. Las pocas que hay consisten en desarrollos pequeños y son poco versátiles. La mayoría están atadas a algún IDE. Ninguna permite el análisis del callgraph de proyectos grandes. Las herramientas de análisis de código existentes actualmente que se aplican en la indústria son cerradas y pagas. Entre las herramientas que existen que permiten visualizar grafos de dependencias o callgraphs, ninguna permite ejecutar ni visualizar una modularización automática.

La comprensión de la estructura del programa es un conocimiento clave para los programadores. Una interfaz basada en texto no revela facilmente la estructura de alto nivel del mismo, con lo cual una visualización como la que se propone es muy requerida. Poder ver y mejorar la arquitectura de un sistema de software puede mejorar de forma muy significativa su legibilidad, modificabilidad y mantenibilidad.

1.2. Estado del arte

La mayoría de las herramientas de análisis estático de código son pagas, con licencias pensadas para empresas grandes. Las herramientas de código abierto que se encuentran disponibles se centran principalmente en la detección automática de bugs o problemas de seguridad (DevSkim, RIPS, PMD, facebook infer, coccinelle, CPAchecker, Cppcheck, Frama-C, Sparse, Clang Static Analyzer), o en análisis que verifican el estilo visual del código. Se listan a continuación herramientas de código abierto que permiten análisis y visualización del código:

- Moose
- Proyecto Bauhaus
- ConQAT

- SoftVis3D

Proyecto Bauhaus y ConQAT son herramientas que inicialmente eran opensource pero gradualmente se dejaron de mantener. SoftVis3D es un plugin de visualizaciones para SonarQube, que no tiene un solo tipo de visualización. Moose es una plataforma desarrollada por investigadores para extraer modelos de algunos lenguajes de programación (entre los cuales se encuentran C y C#), permite desarrollar facilmente visualizaciones, y funciona en un entorno smalltalk "moldeable". Sin embargo, moose no incluye algoritmos de detección automática de comunidades. Si bien es una herramienta muy avanzada para hacer análisis de software, no extrae los datos necesarios para generar callgraphs con precisión, y tampoco extrae métricas de código.

Respecto de la extracción del callgraph, no se encontraron soluciones ya desarrolladas para C# que incluyan la recolección de métricas, y que utilicen las versiones actuales de Roslyn y C#. Para C, se encontraron dos proyectos de código abierto: Egypt y clang-callgraph, los cuales no extraen ningún tipo de métrica. Las versiones más recientes de gcc incluyen un flag para extraer el callgraph de un único archivo.

Para la implementación del extractor de callgraphs para el lenguaje C, se evaluaron tres alternativas distintas:

- Desarrollar un plugin de gcc
- Desarrollar una herramienta usando la biblioteca LibClang
- Utilizar antlr para extraer el AST y extraer el callgraph a partir del mismo

La opción de desarrollar un plugin de GCC se descartó debido a que existe poca documentación de la API de plugins de GCC.

Se escogió utilizar libclang porque existen muchos analizadores e IDEs que ya lo utilizan, con lo cual su interfaz es estable, confiable y está bien documentada. Además de analizar el árbol de sintaxis, libclang provee un análisis semántico del mismo, el cual es requerido para generar el callgraph.

Otra opción considerada fue utilizar antlr para analizar el AST (Abstract Syntax Tree) de los archivos C, ya que la existencia de gramáticas para muchos lenguajes de programación permitiría facilitar aún más la generación de extractores de callgraphs para los mismos. Sin embargo, se descartó porque, a diferencia de libclang, no facilita el análisis del proceso de compilación y, debido a que analiza el texto, no es una solución tan robusta como la brindada por libclang.

El objetivo principal del analizador es proponerle al usuario mejoras sobre la modularización del código fuente de un programa, a partir de su callgraph. Con este fin, se aplicará un algoritmo de detección de comunidades sobre ese grafo y se propondrá cada una de las comunidades obtenidas como un posible módulo (ver figura 1)

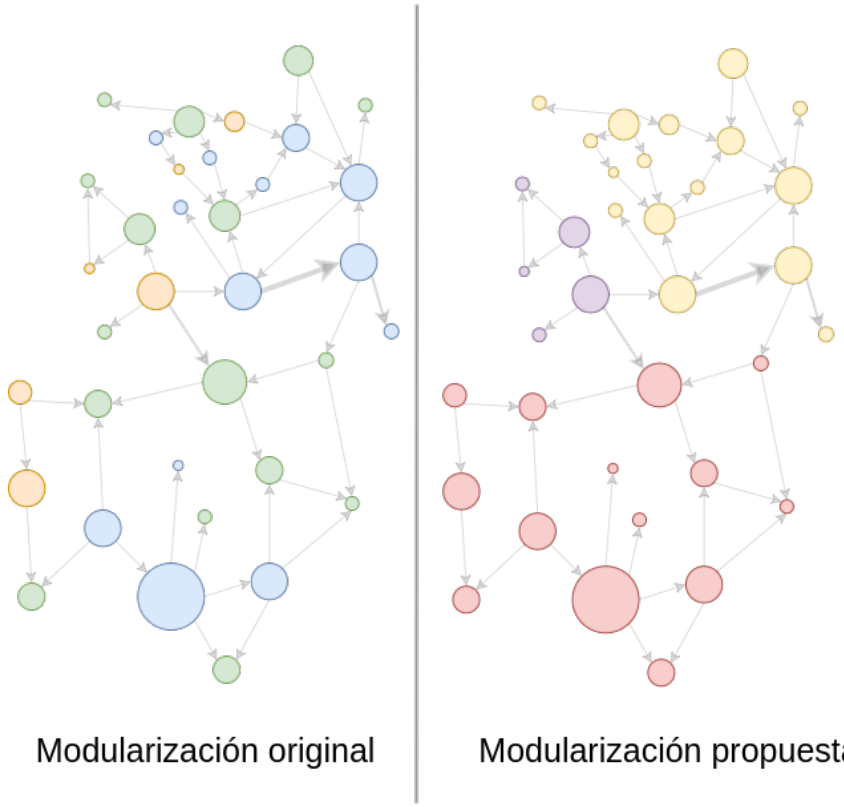


Figura 1: Ejemplo del funcionamiento del analizador. Los vértices representan funciones (o métodos) y las aristas, invocaciones. Los vértices están coloreados según el módulo al que pertenecen. El algoritmo de detección de comunidades propondrá módulos a partir del callgraph.

Respecto del análisis del callgraph, existen dos cuestiones a tratar por separado:

- las métricas de calidad de una partición
- los algoritmos (heurísticas) que optimizan esas métricas

Existe cierta independencia entre las métricas y los algoritmos que las optimizan, siendo posible, con ciertas restricciones, variar una independientemente de la otra.

La novedad y rapidez del algoritmo Leiden respecto de Louvain y numerosos algoritmos preexistentes fueron puntos decisivo para adoptarlo. La mayoría de las métricas que cuantifican la calidad de la partición de una red social en comunidades son matematicamente complejas y guardan poca relación con las buenas prácticas de ingeniería de software, con lo cual se decidió utilizar la Modularidad.

2. Objetivo

Diseñar e implementar un conjunto de herramientas de extracción, visualización y análisis de callgraphs para facilitar la evaluación y mejora de la arquitectura de una aplicación programada en C o C#.

2.1. Subobjetivos

El sistema a desarrollar comprende los siguientes componentes:

- **Analizadores que obtienen el callgraph a analizar.** Como parte del presente trabajo se desarrollarán analizadores para programas escritos en C y en C#. Sin embargo, se utilizará una arquitectura extensible, que permitirá agregar más lenguajes de programación por parte de terceros.

- Una herramienta de visualización y análisis de callgraphs, compuesta por:

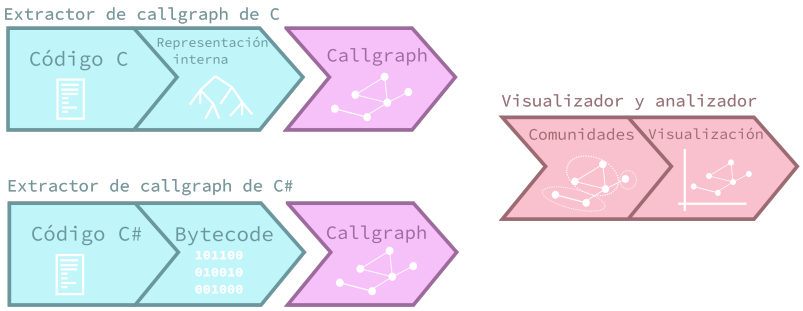
  - **Un módulo que proveerá diversas visualizaciones personalizables.** La herramienta de visualización permitirá aprovechar las conclusiones que brindan los algoritmos de clustering para que los desarrolladores o arquitectos puedan comprender y mejorar su software. Así, permitirá detectar automaticamente nuevos módulos o mejorar los ya existentes.
  - **Un módulo que permitirá aplicar sobre el callgraph un algoritmo de detección de comunidades llamado Leiden (publicado en Octubre de 2019).** El algoritmo Leiden ofrece una mejora clave respecto de las heurísticas de detección de

comunidades en redes sociales que se conocían hasta el momento: las particiones resultantes siempre son conexas. Otro factor muy importante es el hecho de que Leiden es mucho más rápido en la práctica que cualquier otra heurística. Estas características únicas posibilitan la aplicación de un algoritmo de detección de comunidades en un software de este tipo, ya que:

- No tiene sentido que un módulo de software tenga componentes no conexas.
- La detección de los módulos debe ser rápida para no degradar la usabilidad.

3. Diseño de la solución

3.1. Arquitectura de la solución



El proyecto propuesto estará conformado por tres componentes:

- Extractor de callgraphs de programas C
- Extractor de callgraphs de programas C#
- Visualizador y analizador

Se realiza esta división para facilitar la adición de nuevos analizadores por parte de terceros, y para que los analizadores puedan utilizar tecnologías distintas. La arquitectura está pensada para que se puedan agregar rapidamente más lenguajes de programación, además de los que forman parte del alcance de esta propuesta. Los dos analizadores que se proponen como parte del alcance usarán métodos similares, pero tecnologías muy distintas para obtener un callgraph: El analizador de C# utilizará componentes del compilador Roslyn, mientras que el de C utilizará de la biblioteca libclang.

- **Extractores de callgraphs** Su responsabilidad es obtener la información que será analizada por el otro componente. Además de la información del callgraph (es decir, qué función llama a qué otra función), estos analizadores extraen otros datos:

  - Nombre unívoco de la función y su ubicación en los archivos fuente.
  - Ubicación de la función en la jerarquía u organización del programa.
  - Otras características tales como el nivel de visibilidad de la función, o relacionadas al tamaño o complejidad de la misma.

  En el caso particular de C#, el analizador tendrá la responsabilidad de resolver casos de llamadas a interfaces o métodos sobrecargados de forma pesimista, es decir, vinculando a todas las funciones que podrían ser invocadas en ese punto.

- **Visualizador y analizador de callgraphs** Asiste al usuario en la obtención conclusiones a partir de la información recabada. Incluye visualizaciones personalizables que permiten cruzar los datos del callgraph y los obtenidos a partir de la ejecución del algoritmo Leiden.

3.2. Requerimientos funcionales de la solución

3.2.1. Requerimientos funcionales de los analizadores

Los analizadores serán herramientas de línea de comandos, para que sea sencillo integrarlos a la configuración de proyectos de software preexistentes. El resultado de una ejecución de cualquiera de los analizadores será un único archivo cuyo formato será independiente del lenguaje analizado. Para facilitar la adición de nuevos analizadores, el archivo con el resultado del análisis tendrá un formato basado en texto tal como json o xml.

3.2.2. Requerimientos funcionales del visualizador y analizador

El visualizador será una aplicación gráfica que permitirá adjuntar a un callgraph dos tipos de entidades:

- visualizaciones
- comunidades

El asistente de ejecución de Leiden permitirá configurar los hiperparámetros pertinentes y establecer criterios que permitan ejecutar la detección de comunidades en distintos niveles de jerarquía (por ejemplo para clusterizar clases en vez de funciones) o en un subconjunto de los datos. Así, la herramienta permitiría realizar análisis a distintos niveles: puede permitir reubicar los métodos de un pequeño conjunto de clases, o reorganizar los namespaces de una solución de millones de líneas.

El asistente de visualizaciones será similar al de herramientas como Microsoft Excel o Google Sheets (ver figura 2), brindando un conjunto de visualizaciones que se pueden parametrizar y permiten cierto nivel de personalización. A continuación se brinda un listado de visualizaciones a incluir y una breve descripción de cada una:

- **Grafo jerárquico** (ver figura 4): Es un callgraph a cierto nivel de la jerarquía (por ejemplo, namespaces, clases o métodos) y permite al usuario navegar por la jerarquía haciendo click en los vértices. Así, por ejemplo, al hacer click en un namespace puede observarse un callgraph donde los vértices son sus clases, y luego al hacer click sobre una clase se exhiben los métodos de la misma.

- **Grafo jerárquico coloreado por comunidad** (ver figura 5): Es un grafo jerárquico cuyos vértices están coloreados según la comunidad a la que pertenecen. Si el vértice no representa una función sino un conjunto de funciones con distintos colores asignados, se representa como un gráfico de torta.

- **Grafo diff** (ver figura 3): Exhibe el mismo grafo en dos ventanas: en una ventana las funciones están coloreadas según su ubicación en la jerarquía extraída (por ejemplo, según su namespace); en la otra ventana, están coloreadas según la comunidad asignada por Leiden.

- **Histograma**: Reporta una propiedad representativa del tamaño o complejidad como histograma.

- **Treemap**: Reporta alguna propiedad representativa del tamaño como treemap utilizando como jerarquía la extraída del código analizado o el resultado de una ejecución de Leiden.

3.3. Tecnologías y conocimientos a utilizar

Para la construcción del extractor de callgraphs de programas en C# se utilizará C# y la librería Roslyn. Para el extractor de callgraphs de progra-

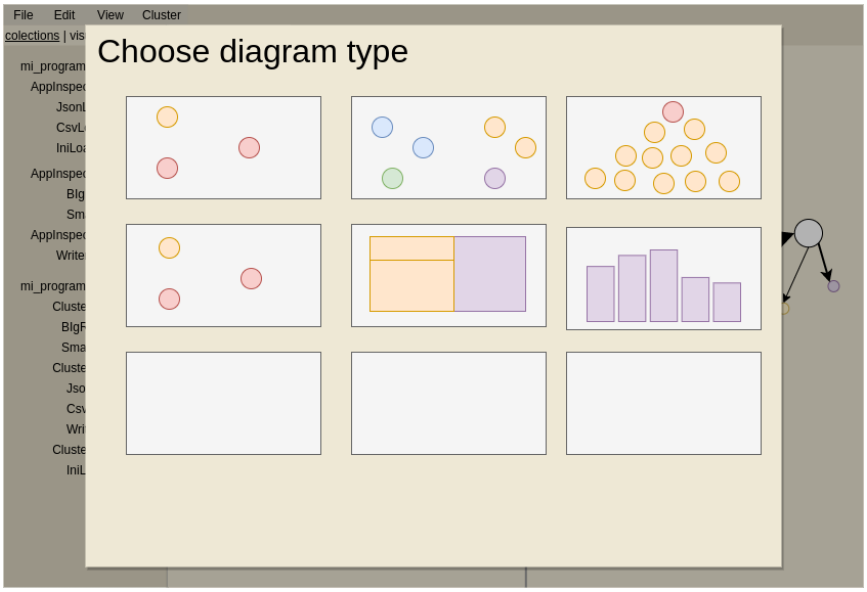


Figura 2: Mockup del selector de tipo de diagrama

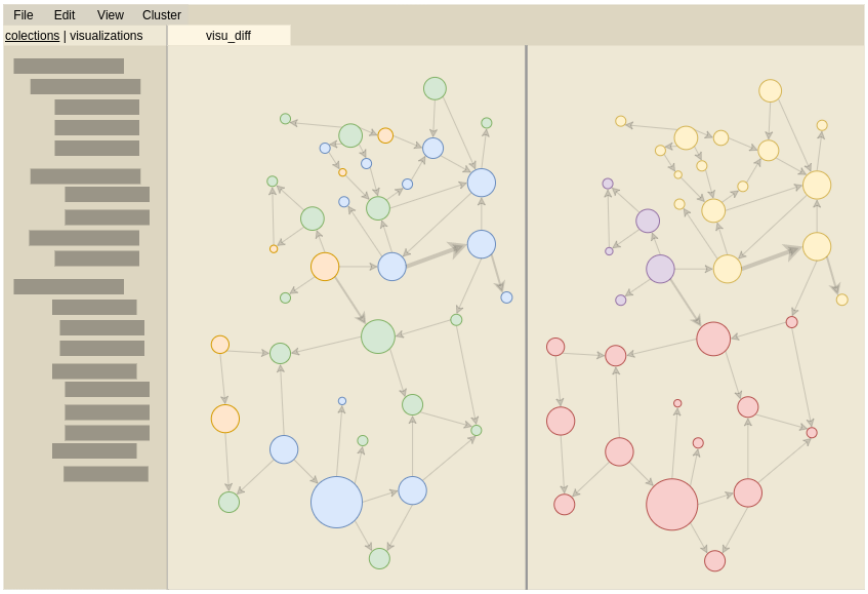


Figura 3: Mockup de la vista diff

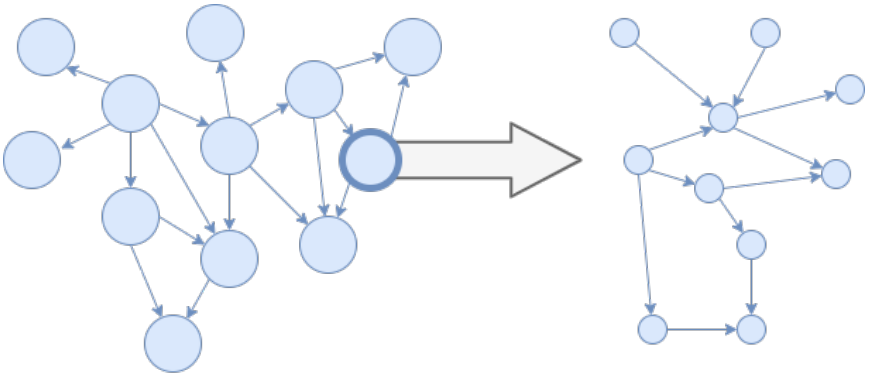


Figura 4: Ilustración que representa al grafo jerárquico, al interactuar con un nodo se visualiza su contenido (por ejemplo, al seleccionar una clase se exhiben sus métodos)

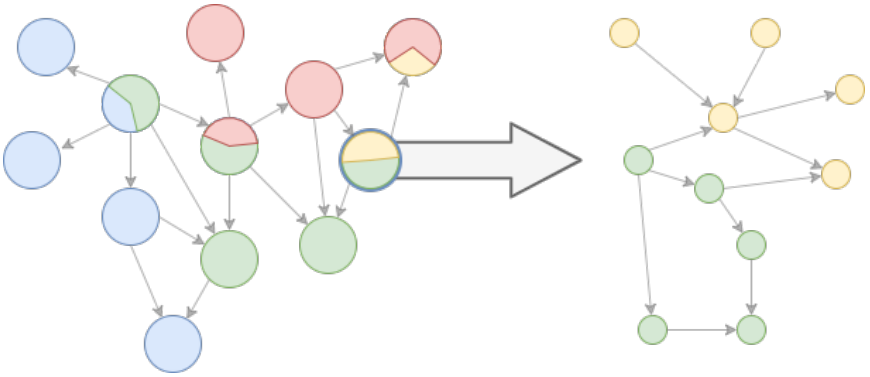


Figura 5: Ilustración representativa del "grafo jerárquico coloreado por comunidad", donde los vértices que agrupan funciones de varias comunidades se representan como un gráfico de torta.

mas C se usará libclang y el lenguaje C. El visualizador se desarrollará en electronjs y vuejs. El algoritmo de clustering incluído estará escrito en C++ y se incluirá como un módulo de nodejs.

4. Plan de trabajo

4.1. Alcance

- Desarrollo de un programa que obtiene el callgraph de un programa C# utilizando roslyn
- Desarrollo de un programa que obtiene el callgraph de un programa C utilizando libclang
- Desarrollo de un visualizador de callgraphs (vue + electron) que:
  - Brinda diversas visualizaciones del callgraph (Vis.js)
  - Permite generar nuevas modularizaciones a partir del algoritmo leiden
  - Incluye la implementación nativa del algoritmo leiden desarrollada por los investigadores que crearon el algoritmo
- Desarrollar un módulo nativo para nodejs que implemente leiden, adaptando la implementación original.
- Relevamiento teórico y técnico de algoritmos de detección de comunidades
- Relevamiento de algoritmos de visualización de grafos
- Relevamiento de librerías y herramientas que facilitarían la extracción de callgraphs

4.2. Metodología de trabajo

El proyecto se llevará adelante por medio de un proceso iterativo incremental. Se realizarán reuniones cada 2 semanas para verificar el avance del proyecto y ajustar objetivos de corto plazo. Cada reunión quedará evidenciada por medio de una minuta, donde quedarán registrados los objetivos acordados. La comunicación entre los miembros se realizará a través de telegram, email y google hangouts. Todos los entregables se disponibilizarán en repositorios de github. El seguimiento del proyecto se realizará utilizando un archivo org-mode disponibilizado en uno de los repositorios.

4.3. Estimación

Tarea	*Descripción*	*Estimación (horas)*
Gestión del proyecto		
Reuniones	Tiempo en reuniones de seguimiento	14
Elaboración de minutas	Elaboración de minutas de reunión	7
		Total: 21
Propuesta de TPP		
Planificación inicial	Construir un detalle y estimación del alcance	10
Elaboración de mockups	Construir mockups para ilustrar mejor el alcance	5
Elaboración del texto	Construir la primera versión del texto de la propuesta	10
Revisión del texto	Revisar la propuesta según comentarios del tutor	5
		Total: 30
Extractor C#		
Relevar antlr, libclang, GCC	Construcción de prototipos utilizando antlr y libclang, construir un plugin de GCC	10
Pruebas	Construcción de pruebas automáticas	10
Extracción del callgraph	El extractor puede obtener el callgraph de una solución	20
Extracción de métricas	El extractor puede obtener métricas sobre la dimensión y complejidad de cada función	20
		Total:60
Extractor C		
Investigar Roslyn y alternativas a Roslyn	Desarrollar un prototipo utilizando roslyn e investigar alternativas	10
Pruebas	Construcción de pruebas automáticas	10
Extracción del callgraph	El extractor puede obtener el callgraph de una compilación	20

Continúa en la siguiente página

Continúa de la página anterior

Tarea	***Descripción***	***Estimación (horas)***
Extracción de metadatos	El extractor puede obtener métricas sobre la dimensión y complejidad de una función	20
		Total: 60
Visualizador - etapa 1	La primera entrega tiene como objetivo resolver custiones tecnológicas	
Configuración	Parametrizar las herramientas para el uso de vuejs y electronjs con módulos nativos	15
Adaptar Leiden	Adaptar la implementación C++ de Leiden para publicarla como un módulo de nodejs	35
Integración con extractores de callgraphs	Posibilidad de ejecutar los extractores desde el visualizador	10
		Total: 60
Visualizador - etapa 2	Funcionalidad mínima	
Panel jerárquico	Panel que permite gestionar una jerarquía, muestra la jerarquía minada.	5
Panel de visualizaciones	Panel que permite gestionar las visualizaciones creadas, y crear nuevas	5
Asistente de creación de visualizaciones	Diálogo que permite crear visualizaciones	10
Visualización jerárquica	Visualización jerárquica	20
Asistente de creación de clusterings	Diálogo que permite parametrizar y ejecutar Leiden	5
Visualización jerárquica por comunidad	Visualización jerárquica coloreada por comunidad	15
		Total: 60
Visualizador - etapa 3		
Visualización de histograma	Visualización de histograma	10

Continúa en la siguiente página

Continúa de la página anterior

Tarea	*Descripción*	*Estimación (horas)*
Visualización de treemap	Visualización de treemap	10
Operaciones sobre jerarquías	Agregar operaciones que permiten modificar un clustering generado	10
Gráfico diff	Agregar la visualización diff	20
		Total: 50
Relevamiento sobre algoritmos de clustering de grafos		
Investigación	Comprende la búsqueda, síntesis y análisis de bibliografía	20
Redacción y revisión	Comprende la redacción del texto definitivo a ser presentado	10
		Total: 30
Relevamiento sobre librerías de visualización de grafos		
Investigación y recolección de datos	Se lista un conjunto de librerías a evaluar	8
Prototipos	Elaboración de prototipos con un subconjunto de las librerías relevadas	4
Redacción y revisión	Redacción del texto definitivo a ser presentado	3
		Total: 15
Presentación		**Total: 20**
Documentación del proyecto		
Especificación del formato de callgraph	Se elabora un documento público que especifica el formato del callgraph	5
Documentación del visualizador	Generación de documentos públicos que explican cómo se utiliza el visualizador	10

Continúa en la siguiente página

Continúa de la página anterior

Tarea	*Descripción*	*Estimación (horas)*
Documentación del extractor C	Generación de documentos públicos que explican cómo se utiliza el extractor de C	10
Documentación del extractor C#	Generación de documentos públicos que explican cómo se utiliza el extractor de C#	10
		Total: 35
Suma general del esfuerzo	Totalidad del Trabajo Práctico Profesional	**Total: 441**

4.4. Cronograma

Cada iteración comprende 60 horas de trabajo en entregables, y 3 en tareas de gestión (reuniones y elaboración de minutas). En el caso de los entregables cuyo esfuerzo es menor a 60 horas, se destinarán las horas restantes a trabajar sobre el entregable de la iteración siguiente. La última iteración (séptima) comprenderá 33 horas de trabajo. Cada iteración constará de 4 semanas de trabajo, a un ritmo de 15 horas semanales de trabajo sobre entregables, además de 3 horas de gestión del proyecto. La última iteración tomará 2 semanas.

Iteración	Sem. Inicio	Sem. Fin	**Entregables**	Esfuerzo entregado (horas)	Esfuerzo entregado total (horas)	Esfuerzo invertido (horas)
1	1	4	Investigación sobre librerías de visualización de grafos Investigación sobre algoritmos de clustering Minutas y reuniones	15 30 3	48	63
2	5	8	Extractor C# Minutas y reuniones	60 3	63	63
3	9	12	Extractor C Minutas y reuniones	60 3	63	63
4	13	16	Visualizador - etapa 1 Minutas y reuniones	60 3	63	63
5	17	20	Visualizador - etapa 2 Minutas y reuniones	60 3	63	63
6	21	24	Visualizador - etapa 3 Minutas y reuniones	50 3	53	63
7	25	26	Documentación del proyecto Presentación Minutas y reuniones	35 20 3	58	33
Total			Totalidad del Trabajo Práctico Profesional	411	411	411

Propuesta	Entregables	**Total**
30 horas	411 horas	441 horas

4.5. Descripción de los entregables

4.5.1. Propuesta

Consiste en el desarrollo de la presente propuesta. Las tareas involucradas en este entregable fueron:

- Investigar herramientas existentes de análisis y visualización de código
- Buscar herramientas y librerías que permitan extraer el callgraph de distintos lenguajes
- Determinar el alcance y la arquitectura del TP
- Redacción y revisión de la propuesta

4.5.2. Investigación sobre algoritmos de clustering

Comprende la búsqueda de trabajos de investigación que describan algoritmos de clustering de grafos que puedan ser utilizados para detectar módulos, la elaboración de un informe que resume los hallazgos y la elección del algoritmo a utilizar.

4.5.3. Investigación y relevamiento de librerías de visualización de grafos

Consiste en:

- Análisis de un conjunto de librerías de visualización de grafos para javascript
- Selección de un subconjunto de menos de 5 librerías
- Desarrollo de prototipos utilizando las librerías seleccionadas
- Desarrollo de un documento que compara todas las librerías halladas y describe los prototipos realizados.

4.5.4. Extractor C#

Consiste en el desarrollo de una herramienta de software que extrae el callgraph de un programa C# realizando un análisis estático de su código fuente. Está comprendido en este entregable, además del código fuente de la herramienta, un conjunto de pruebas automáticas. La documentación asociada es parte de otro entregable. La herramienta se desarrollará en el lenguaje C# utilizando Roslyn.

4.5.5. Extractor C

Comprende las mismas tareas que para el extractor C#, pero para el lenguaje C. En este caso, la herramienta se desarrollará utilizando C, y en vez de Roslyn se utilizará libclang.

4.5.6. Visualizador

Consiste en el desarrollo de un visualizador que permita a un usuario ver los callgraphs extraídos. Este componente se desarrollará utilizando electronjs y vuejs. Además de la herramienta misma y su código fuente, este entregable comprende la adaptación del algoritmo Leiden para que pueda ser ejecutado por un proceso nodejs.

1. **Etapa 1** La etapa 1 comprenderá la resolución de diversas cuestiones tecnológicas. El resultado no tendrá demasiada utilidad: debe mostrar el resultado de la ejecución de Leiden como texto y permitir la ejecución de los extractores desde la misma herramienta.

2. **Etapa 2** La etapa 2 comprende una parte de las visualizaciones y el front-end

3. **Etapa 3** En la etapa 3 se agregan las visualizaciones faltantes en las otras dos etapas.

4.5.7. Documentación del proyecto

Comprende la documentación técnica y manuales de usuario correspondientes a cada uno de los componentes de software.

4.5.8. Presentación

Preparación de una presentación oral

5. Glosario

- Callgraph: Es un grafo dirigido en el que cada vértice representa una función, y las aristas (dirigidas) representan el hecho de que una función invoca a otra. La noción de callgraph y control flow graph se desarrollan en (Nielson, Nielson, y Hankin, 2015).

- Algoritmo de clustering de grafos: Es un algoritmo que encuentra conjuntos de vértices fuertemente relacionados en un grafo. La bibliografía no es unánime respecto de qué condiciones deben reunir tales conjuntos. Existen diversas métricas de calidad, descriptas en: (Newman y Girvan, 2004) (Reichardt y Bornholdt, 2006) (Traag, Van Dooren, y Nesterov, 2011) (Traag, Aldecoa, y Delvenne, 2015) (Traag, Krings, y Van Dooren, 2013).

- Algoritmo de Detección de comunidades: Si bien una red social es un grafo con propiedades particulares, en este documento se utilizan los términos çlustering de grafosz "detección de comunidadesçomo sinónimos.

6. Referencias

Appel, A. W. (2004). *Modern compiler implementation in c*. Cambridge university press.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., y Lefebvre, E. (2008, Oct). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, *2008*(10), P10008. Descargado de `http://dx.doi.org/10.1088/1742-5468/2008/10/P10008` doi: 10.1088/1742-5468/2008/10/p10008

Diehl, S. (2007). *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media.

DOMINGUE, J. A. (1998). *Software visualization: Programming as a multimedia experience*. MIT press.

Newman, M. E. J., y Girvan, M. (2004, Feb). Finding and evaluating community structure in networks. *Phys. Rev. E*, *69*, 026113. Descargado de `https://link.aps.org/doi/10.1103/PhysRevE.69.026113` doi: 10.1103/PhysRevE.69.026113

Nielson, F., Nielson, H. R., y Hankin, C. (2015). *Principles of program analysis*. Springer.

Parés, F., Gasulla, D. G., Vilalta, A., Moreno, J., Ayguadé, E., Labarta, J., ... Suzumura, T. (2017). Fluid communities: a competitive, scalable and diverse community detection algorithm. En *International conference on complex networks and their applications* (pp. 229–240).

Reichardt, J., y Bornholdt, S. (2006, 08). Bornholdt, s.: Statistical mechanics of community detection. physics review e 74(1), 016110. *Physical review. E, Statistical, nonlinear, and soft matter physics*, *74*, 016110. doi: 10.1103/PhysRevE.74.016110

Schaeffer, S. (2007, 08). Graph clustering. *Computer Science Review*, *1*, 27-64. doi: 10.1016/j.cosrev.2007.05.001

Traag, V. A., Aldecoa, R., y Delvenne, J.-C. (2015, Aug). Detecting communities using asymptotical surprise. *Phys. Rev. E*, *92*, 022816. Descargado de `https://link.aps.org/doi/10.1103/PhysRevE.92.022816` doi: 10.1103/PhysRevE.92.022816

Traag, V. A., Krings, G., y Van Dooren, P. (2013). Significant scales in community structure. *Scientific reports*, *3*(1), 1–10.

Traag, V. A., Van Dooren, P., y Nesterov, Y. (2011, Jul). Narrow scope for resolution-limit-free community detection. *Phys. Rev. E*, *84*, 016114. Descargado de `https://link.aps.org/doi/10.1103/PhysRevE.84.016114` doi: 10.1103/PhysRevE.84.016114

Traag, V. A., Waltman, L., y van Eck, N. J. (2019). From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*, *9*(1), 1–12.

Currículum Vitae

- **Información personal**
  - **Nombre:** Sbruzzi, José Ignacio
  - **Nacionalidad:** Argentina
  - **Fecha de nacimiento:** 5 de Octubre de 1995
  - **Dirección:** Patagones 167, Avellaneda, Provincia de Buenos Aires
  - **Teléfono:** +54 11 3007 0139
  - **E-Mail:** jose.sbru@gmail.com
- **Educación**
  - 2011,2012: **Pueri Domus unidade Verbo Divino**, São Paulo, SP, Brasil
  - 2009,2010,2013: **Escuela Media N° 2: "Juan María Gutierrez"**. Bachiller en Ciencias Naturales.
  - 2014 - actualidad: **Facultad de Ingeniería - Universidad de Buenos Aires**. Ingeniería en Informática.
- **Idiomas**
  - Español: nativo
  - Inglés: proficent (Nivel C1 según el Common European Framework of Reference)
  - Portugués: avanzado
- **Experiencia Laboral**
  - Abril 2018 - Octubre 2018: **Investigación y Desarrollo.** *Ayi y Asociados*
  - Noviembre 2018 - Actualidad: **Desarrollador de Software.** *CyS Informática*
- **Habilidades técnicas**
  - python
  - nodejs
  - jquery

- bootstrap
- asp.net webforms
- asp.net MVC
- NHibernate
- webgl
- reactjs
- C/C++
- docker
- bash
- visual studio
- linux
- SQL
- Mongo DB
- Ocaml
- lisp (clojure y clisp)
- coq
- java
- Amazon Web Services
- git

Historial académico

Materias del Ciclo Básico Común

Código	Nombre	Nota
28	Análisis matemático	10
3	Física	9
24	Introducción al Conocimiento de la Sociedad y el Estado	9
40	Introducción al Pensamiento Científico	10
5	Química	9
27	Álgebra	8

Materias obligatorias comunes

Código	Nombre	Nota	Créditos
6201	FISICA I A	8	8
7540	ALGORITMOS Y PROGRAMACION I	4	6
6103	ANALISIS MATEMATICO II A	9	8
6108	ALGEBRA II A	9	8
6203	FISICA II A	8	8
6301	QUIMICA	9	6
7541	ALGORITMOS Y PROGRAMACION II	9	6
6215	FISICA III D	8	4
6109	PROBABILIDAD Y ESTADISTICA B	7	6
6670	ESTRUCTURA DEL COMPUTADOR	9	6
6602	LABORATORIO	8	6
7512	ANALISIS NUMERICO I	9	6
7506	ORGANIZACION DE DATOS	10	6
7507	ALGORITMOS Y PROGRAMACION III	9	6
7542	TALLER DE PROGRAMACION I	9	4
6110	ANÁLISIS MATEMÁTICO III A	4	6
7112	ESTRUCTURA DE LAS ORGANIZACIONES	9	6
6620	ORGANIZACION DE COMPUTADORAS	8	6
7508	SISTEMAS OPERATIVOS	8	6
7114	MODELOS Y OPTIMIZACION I	10	6
7509	ANALISIS DE LA INFORMACION	8	6
7510	TECNICAS DE DISEÑO	7	6
7543	INTRODUCCION A LOS SISTEMAS DISTRIBUIDOS	8	6
7515	BASE DE DATOS	9	6
7552	TALLER DE PROGRAMACION II	8	4
7140	LEG. Y EJ. PROF. DE LA ING. EN INFORMAT.	8	4
			156

Materias electivas

Código	Nombre	Nota	Créditos
6107	MATEMATICA DISCRETA	10	6
7531	TEORIA DE LENGUAJE	10	4
6671	SISTEMAS GRÁFICOS	10	6
7529	TEORIA DE ALGORITMOS I	10	6
7573	ARQUITECTURA DEL SOFTWARE	9	4
7514	LENGUAJES FORMALES	9	6
7516	LENGUAJES DE PROGRAMACION	10	6
7118	ESTRUCTURA ECONOMICA ARGENTINA	9	4
-	Aprendizaje Estadístico: Teoría y Aplicaciones	10	4
			46

Materias obligatorias de la orientación

Código	Nombre	Nota	Créditos
7113	INFORMACION EN LAS ORGANIZACIONES	8	6
7544	ADM. Y CONTROL DE PROY. INFORMATICOS I	9	6
7545	TALLER DE DESARROLLO DE PROYECTOS I	9	6
7546	ADM. Y CONTROL DE PROY. INFORMATICOS II	7	6
7547	TALLER DE DESARROLLO DE PROYECTOS II	8	6
7548	CALIDAD EN DESARROLLO DE SISTEMAS	10	6
			36

Resumen

- Promedio general de la carrera (sin tener en cuenta el CBC): 8.50
- Total de créditos: 238
- Faltante de créditos: 10 (exceso de 2 créditos)