

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“ISHVEL: UN FRAMEWORK PARA LA ELABORACIÓN DE TAREAS EN CURSOS INTRODUCTORIOS DE PROGRAMACIÓN”

GONZALO ANDRÉS FERNÁNDEZ CARRILLO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Federico Meza
Profesora Correferente: Pedro Godoy

Junio - 2023

DEDICATORIA

Dedico este trabajo a mi madre, padre y hermano, quienes siempre han estado allí en los mejores y peores momentos, nunca me abandonaron cuando más los necesité.

AGRADECIMIENTOS

Agradezco a mi profesor guía y al equipo de coordinación de IWI-131 en la UTFSM, quienes me dieron la oportunidad de trabajar como ayudante coordinador del ramo durante todos estos años, y me han apoyado enormemente en desarrollar mi lado docente tanto en los talleres OCILabs, como en distintas otras instancias.

Agradezco también a mis amigas y amigos, quienes son y han sido pilares fundamentales a lo largo de mi vida desde que los conocí, siempre han estado allí y el que esté escribiendo estas últimas palabras en este documento son prueba irrefutable de que sin su apoyo no habría llegado tan lejos.

RESUMEN

Resumen— El curso de programación IWI-131 de la UTFSM es para muchos estudiantes, el primer acercamiento al mundo de la programación, y por lo mismo, es importante llevarlo a cabo de la mejor manera posible. Con el fin de lograr esto, se desarrolló el framework Ishvel, con el cual se busca apoyar el proceso de elaboración de las tareas de este curso, de modo tal de hacer de este instrumento formativo evaluativo, uno de calidad para los estudiantes.

Para esto, se proponen métricas de software con las cuales analizar la dificultad de una tarea en base a su solución, y se elaboró un editor de tareas con un sistema para comparar qué tan difícil es una tarea, con respecto a una tarea anterior llevada a cabo en el ramo.

Con este editor se pudo llevar a cabo el análisis de dos tareas en dos semestres distintos, determinando gracias a la información que entrega el mismo, cuales fueron los aspectos en que fallaron, cómo se podrían mejorar, y qué acciones se pudieron haber tomado de haber contado con la metodología de este framework a la hora de haber sido elaboradas.

Palabras Clave—Elaborar tareas, framework, curso introductorio de programación, educación

ABSTRACT

Abstract— The IWI-131 programming course of the UTFSM is for many students, the first approach to the world of programming, and therefore, it is important to carry it out in the best possible way. In order to achieve this, the Ishvel framework was developed, seeking to support the process of developing the homeworks of this course, so as to make an evaluative training tool as important as the homeworks, one of quality for students.

Based on this, software metrics are proposed to analyze the difficulty of a homework based on its solution, and a homework editor was developed integrating a system to compare how difficult a homework is based on its solution, regarding to a homework carried out in the course in a previous semester.

With this editor it was possible to analyze two homeworks from two different semesters, and by using the information provided by the tool, it was determined the aspects in which they failed, how they could be improved, and what actions could have been taken if the methodology of this framework had been used at the time they were formulated.

Keywords—Programming homework design, programming education, introductory programming course, CS1 homeworks

GLOSARIO

- DI: Departamento de Informática.
- IWI-131: Sigla del curso de programación de la UTFSM.
- LMS: Sistema de Gestión del Aprendizaje (por sus siglas en inglés, Learning Management System)
- UTFSM: Universidad Técnica Federico Santa María.
- S20XX-X: Semestre lectivo 20XX-X (Ejemplo: S2022-1 corresponde al primer semestre del 2022).
- UVA: Unidad Virtual de Aprendizaje.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	IX
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Objetivos	4
1.1.1 Objetivos Específicos	4
CAPÍTULO 2: MARCO CONCEPTUAL	5
2.1 Framework	5
2.2 Frameworks en el Apoyo a la Enseñanza	5
2.3 Curso Introductorio de Programación	8
2.4 Tarea de Programación	8
2.4.1 Dominio de un Problema en una Tarea	9
2.4.2 Factores de Importancia en una Tarea de Programación	9
2.4.3 Reutilización de Tareas de Programación	10
2.5 Métricas de Software	11
2.6 Complejidad Ciclomática	11
2.7 Métricas de Halstead	12
2.7.1 Volumen de un Programa	12
2.7.2 Volumen Potencial	13
2.7.3 Nivel de un Programa	14
2.7.4 Dificultad de un Programa	15
2.7.5 Esfuerzo de Programación	15
2.7.6 Tiempo Estimado de Programación	16
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	17
3.1 Herramientas y Tecnologías a Utilizar	17
3.1.1 Google Trends	17
3.1.2 WebAssembly	17
3.1.3 Python	17
3.1.4 Javascript	18
3.1.5 Multimetric	18
3.1.6 Markdown	18
3.1.7 GitHub Pages	19

3.1.8	Pyodide	19
3.1.9	React	19
3.2	Arquitectura de la Solución	19
3.3	Framework	21
3.3.1	Metodología	21
3.3.2	Cálculo de Métricas	28
3.3.3	Determinar la Diferencia de Dificultad entre Tareas	29
3.3.4	Determinar los Intervalos de Comparación de Dificultad entre Tareas	30
3.3.5	Sugerencias	37
3.3.6	Actualización de Métricas Semestrales	38
3.3.7	Actualización de Intervalos de Comparación de Dificultad	41
3.3.8	Análisis de Métricas entre Semestres	43
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		45
4.1	Validación del Framework	45
4.1.1	Cálculo de Métricas para Tareas del S2022-1 y S2022-2	45
4.1.2	Cálculo de los Intervalos de Dificultad	45
4.1.3	Diferencia de Dificultad entre Tareas del S2022-1 y S2022-2	48
4.1.4	Actualización de Intervalos de Comparación de Dificultad	49
4.1.5	Actualización de Métricas S2022-1 y S2022-2	50
4.1.6	Redacción de una Tarea con sus Respectivas Métricas	51
4.1.7	Comparación de las Tareas de Listas de IWI-131 del S2022-1 y S2022-2	55
4.1.8	Comparación de las Tareas de Archivos de IWI-131 del S2022-1 y S2022-2	57
CAPÍTULO 5: CONCLUSIONES		61
5.1	Trabajo a Futuro	62
REFERENCIAS BIBLIOGRÁFICAS		63

ÍNDICE DE FIGURAS

1	Unidad Virtual de Aprendizaje [Vásquez <i>et al.</i> , 2021].	3
2	Árbol del Problema.	4
3	Implementación de los patrones de un framework en un LMS [Derntl y Calvo, 2011]	6
4	Pantalla principal de la interfaz de usuario de Patman (abreviación de <i>pattern manager</i>) [Derntl y Calvo, 2011]	7
5	Nivel de importancia de los factores de una tarea, asignado por los mismos estudiantes según si les genera interés, o si les haría elegir desarrollar esa tarea por sobre otra [Torrey, 2011]	10
6	Arquitectura del editor de Ishvel.	20
7	Opciones de configuración del editor de Ishvel.	22
8	Sugerencia inicial del editor de Ishvel.	22
9	Formato de ejemplo de una tarea en el editor de Ishvel.	24
10	Métricas de una solución según Ishvel.	25
11	Sugerencias para una tarea según Ishvel.	26
12	Botón de descargar tarea en el editor de Ishvel.	27
13	Sección de métricas históricas del framework Ishvel.	27
14	Ejemplo de uso de la herramienta metrics-research con el parámetro -f.	28
15	Ejemplo de uso de la herramienta metrics-research con el parámetro -d.	28
16	Ejemplo de uso de la herramienta metrics-research con el parámetro -h.	29
17	Archivos de datos del editor.	38
18	Formato de los archivos de métricas del editor.	39
19	Ejemplo de como añadir las métricas de la tarea de archivos del semestre 2022-2 en el archivo de métricas del editor.	40
20	Ejemplo de como añadir las métricas de la tarea de archivos del semestre 2022-3 en el archivo de métricas del editor.	41
21	Contenido del archivo de intervalos de comparación de dificultad.	42
22	Sección de métricas históricas del framework Ishvel.	44
23	Presentación del formulario para la diferencia de dificultad entre tareas.	46
24	Ejemplo de las preguntas para comparar la dificultad entre 2 tareas de distintos semestres.	47
25	Intervalos de comparación de dificultad cargados en el framework.	49
26	Métricas por semestre de las soluciones de los estudiantes cargados en el framework.	50
27	Métricas por semestre de las soluciones de los profesores cargados en el framework.	51
28	Editor de Ishvel.	52
29	Tarea elaborada en el editor de Ishvel.	53
30	Métricas y sugerencias para una solución en base a una tarea redactada en el editor de Ishvel.	54
31	Métricas de una solución en base a una tarea redactada en el editor de Ishvel.	55
32	Métricas históricas del S2022-1 y S2022-2 para la tarea de listas en el editor.	55

33	Comparación de dificultad de la solución de la tarea de listas del S2022-2 en el editor de Ishvel	57
34	Comparación de dificultad de la solución de la tarea de archivos del S2022-2 en el editor de Ishvel	58

ÍNDICE DE TABLAS

1	Ejemplo de intervalos de comparación de dificultad para todas las métricas, a excepción del tiempo estimado.	30
2	Ejemplo de métricas de dos tareas del mismo contenido.	30
3	Ejemplo de la comparación de dificultad entre tareas, junto con las diferencias porcentuales de cada una de sus métricas.	32
4	Tabla base de intervalos por métrica para cada dificultad comparativa.	33
5	Tabla base de intervalos por métrica para cada dificultad comparativa con los límites máximos y mínimos del primer y último intervalo listos.	33
6	Tabla base de intervalos por métrica para cada dificultad comparativa con el primer intervalo listo.	34
7	Tabla base de intervalos por métrica para cada dificultad comparativa con el límite inferior del segundo intervalo.	34
8	Tabla base de intervalos por métrica para cada dificultad comparativa casi completa para una métrica.	35
9	Tabla base de intervalos por métrica para cada dificultad comparativa para la métrica de complejidad ciclomática.	36
10	Resultado de la encuesta a expertos sobre la diferencia de dificultad entre tareas.	47
11	Comparación de dificultad entre tareas, junto con las diferencias porcentuales de cada una de sus métricas	48
12	Intervalos de dificultad por métrica para cada dificultad comparativa.	48
13	Dificultad de las tareas según los intervalos de comparación	49

INTRODUCCIÓN

En la UTFSM se dicta el curso de introducción a la programación IWI-131 para todos los estudiantes de ingeniería en su primer año. Este es un momento crucial para su proceso de aprendizaje dado que para muchos, es el primer acercamiento que tienen con el mundo de la programación, y dependiendo de la experiencia educativa que tengan a lo largo del semestre, su motivación se verá directamente impactada de cara a adentrarse más en este mundo. Por lo mismo, es importante afinar todos los detalles posibles para hacer de su estadía por este ramo una experiencia amena y disfrutable.

Durante este primer acercamiento hacia el mundo de la computación, los estudiantes experimentarán diversas formas de aprender y ser evaluados, en particular, siguiendo la metodología de enseñanza del curso IWI-131 que se imparte desde el 2020, la cual separa los contenidos de las asignaturas en pequeñas unidades de aprendizaje llamadas Unidad Virtual de Aprendizaje (UVA), y donde cada una de estas contempla actividades con un rol tanto formativo como sumativo.

Las tareas forman una parte muy importante del proceso de aprendizaje y enseñanza, no sólo por el rol que toman, si no también para medir el nivel de aprendizaje general de los estudiantes, y brindarles a estos una retroalimentación significativa y en el momento adecuado. Por lo mismo, es necesario que las tareas sean redactadas de la mejor manera posible, centrándose en evaluar los contenidos de modo tal que el estudiante se motive a aprender, y que no generen efectos adversos en el mismo como lo pueden ser la frustración, falta de ganas de aprender y en el peor de los casos, la deserción.

En este contexto se desenvuelve esta memoria, buscando brindar al equipo docente del ramo de programación un framework que les ayude a elaborar sus tareas. Para lograr esto, se plantea una metodología para elaborar tareas en cursos introductorios de programación, acompañada de un editor de tareas que entrega métricas y sugerencias a los profesores respecto de la tarea que están elaborando.

Toda la investigación en este documento se centra en el uso de métricas de software para el análisis y la comparación de tareas de programación en base al código de sus soluciones. Estas métricas tienen un amplio uso en la industria para ver aspectos sobre la mantenibilidad de su software empresarial, sin embargo este trabajo presenta uno de sus primeros usos como índices sobre la dificultad de lo que se busca evaluar para el estudiante. Gracias a esto, se permite hacer un acercamiento hacia cómo poder determinar la dificultad de una tarea en base a su solución y en comparación a otras tareas. Para así lograr el objetivo de elaborar mejores tareas con la ayuda del framework Ishvel, el cual es una herramienta web que entrega métricas sobre las soluciones de las tareas en cursos introductorios de programación, y brinda un editor para redactar tareas de manera sencilla.

Este documento se separa en 5 capítulos: Definición del Problema (1), Marco Conceptual (2), Propuesta de Solución (3), Validación de la Solución (4) y Conclusiones (5).

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

Los cursos introductorios de programación son, en general, el primer acercamiento de un estudiante a los conceptos fundamentales de la computación [Omer U, 2021], y por lo tanto, se debe velar porque se desarrollen de la manera más prolija posible, considerando entre otras cosas, la elaboración y uso de tareas de calidad. Las tareas son parte importante del proceso de aprendizaje y enseñanza [Boye, 2020], pues permiten medir el aprendizaje de los estudiantes y proporcionarles una retroalimentación significativa [National Academy of Engineering, 2009]. Además, los cursos introductorios de programación, son la principal actividad en donde los estudiantes ponen en práctica lo que han aprendido sobre programación, lo que hace que las tareas jueguen un rol importante en su interés por aprender, pudiendo llevar al estudiante tanto a querer sobresalir resolviendo las evaluaciones del curso, como a desertar debido a su percepción de la programación por tareas de mala calidad [Torrey, 2011, Vivian *et al.*, 2013], entre otros efectos como los mencionados en el Árbol del Problema (ver Figura 2).

En la Universidad Técnica Federico Santa María (UTFSM), se dicta un curso introductorio de programación a lo largo de todos sus campus. Este recibe de forma masiva a todos los estudiantes de primer año de ingeniería, y se dicta de igual forma independiente del campus o carrera del estudiante. El curso, que se basa en una estrategia de aula invertida [Tucker, 2012], donde los estudiantes deben realizar un trabajo previo mirando videos y contestando cuestionarios asociados, está dividido en Unidades Virtuales de Aprendizaje (UVA), las cuales estructuran las actividades de cada semana (ver Figura 1), y su diseño se basa en el alineamiento constructivista [Biggs y Tang, 2011]. Este último plantea la necesidad de que las evaluaciones del curso reflejen los objetivos de aprendizaje del mismo y, que a su vez, sean estos los que definan las actividades y tareas a realizar. Esto brinda a las tareas un rol tanto formativo como sumativo de evaluar, y requiere que estas cumplan ciertos estándares para justificar que cumplen con los objetivos de aprendizaje [Biggs y Tang, 2011].

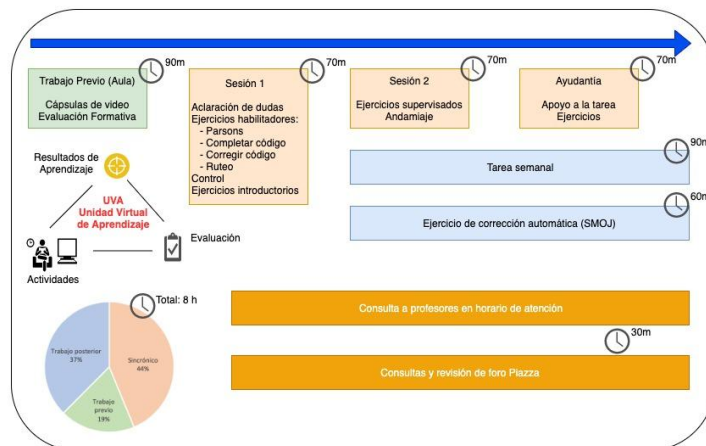


Figura 1: Unidad Virtual de Aprendizaje [Vásquez *et al.*, 2021].

Sin embargo, pese a que el curso cuenta con una gran cantidad de profesores, no son más de 5 los que se encargan de proponer tareas a la coordinación. Y, dado que cada UVA requiere una tarea, se vuelve complicado elaborar tareas de calidad a tiempo, que cumplan con el alineamiento constructivista del curso, y que cuya resolución no requiera más tiempo que el estipulado en la UVA. Además, la elaboración de tareas se lleva a cabo sin métricas que permitan medir la calidad de cada una, o una metodología de elaboración de tareas que garantice la calidad de las mismas en el tiempo.

Entre los problemas que puede conllevar el desarrollo de tareas que no son de calidad, se encuentran:

- Pérdida del interés en aprender a programar [Layman *et al.*, 2007], de modo tal que el estudiante deje de ver el ramo como un curso de aprendizaje, y su actitud hacia él sea nétamente para aprobar.
- Frustración a lo largo del ramo, al punto en que el estudiante puede decidir desertar del curso y de la programación en general [Vivian *et al.*, 2013].
- Aumentar las probabilidades de que los estudiantes realicen actos que falten a la honestidad académica para resolverla [Simon, 2017].
- Complicar la elaboración de la rúbrica con la cual la tarea será evaluada, lo que puede conllevar a entregar un feedback menos valioso al estudiante. Notar además, que el 20 % de la calificación final del estudiante corresponde a las tareas de cada UVA, por lo que es importante contar con un buen instrumento para revisar.

Es por esto que se requiere de un marco de trabajo que permita elaborar tareas de calidad, el cual brinde tanto una metodología como herramientas que faciliten a los profesores esta labor, ahorrándoles tiempo y garantizando a los estudiantes del curso una mejor experiencia de aprendizaje.

1.1. Objetivos

El objetivo general de esta memoria consiste en desarrollar un framework para la elaboración de tareas en cursos introductorios de programación.

1.1.1. Objetivos Específicos

Con el fin de lograr el objetivo general de esta memoria, se definen los siguientes objetivos específicos:

- Identificar y definir criterios que puedan ser usados para evaluar una tarea.
- Definir una métrica que permita evaluar una tarea de acuerdo al cumplimiento de criterios.
- Definir una metodología para la elaboración de tareas utilizando el framework desarrollado.

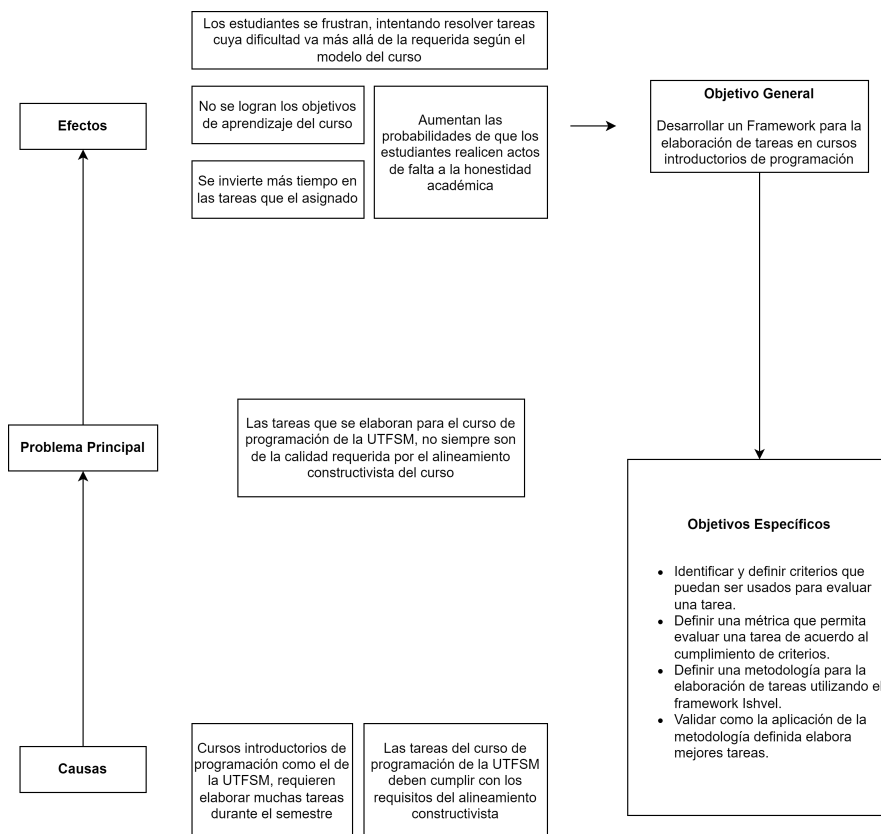


Figura 2: Árbol del Problema.
Fuente: elaboración propia.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Framework

Un framework es un marco de trabajo [La Red Martínez *et al.*, 2012], que plantea una metodología validada y un conjunto de herramientas, las cuales están diseñadas para la correcta aplicación de dicha metodología. Un framework se concibe en base a la acumulación de experiencia, buenas prácticas, patrones y soluciones validadas sobre el dominio de un problema recurrente, el cual ha sido abordado a través del tiempo y sus maneras de resolverse, han sido bien documentadas y mantenidas [Fayad *et al.*, 1999]. A modo de ejemplo, en el área de la ingeniería de software, un framework brinda al desarrollador un esqueleto base para el desarrollo de algún software, el cual tiene como base la aplicación de distintos patrones arquitectónicos que permiten resolver de forma eficiente algún problema en particular [Derntl y Calvo, 2011].

2.2. Frameworks en el Apoyo a la Enseñanza

En el contexto de apoyar la labor de enseñar, los frameworks de este dominio apuntan a mejorar la gestión del contenido educativo, siguiendo buenas prácticas rescatadas del área de la ingeniería de software, y aplicadas al dominio de la enseñanza. Estas prácticas permiten reutilizar aspectos prácticos de un curso previo, como lo es su organización y sus contenidos, y mejorarlos en el tiempo para así ahorrar tiempo y poder aplicar patrones de diseño que faciliten la labor de educar. A diferencia de los frameworks de ingeniería de software, estos frameworks están orientados a ser utilizados por un profesor, y en vez de entregar código, brindan herramientas y metodologías con las cuales el usuario puede gestionar de mejor manera su curso [Derntl y Calvo, 2011].

Por otro lado, al ser la enseñanza un dominio tan estudiado, existen muchos patrones que abordan diversas formas de resolver los problemas que ésta conlleva [Derntl y Calvo, 2011]. Esta realidad ha impulsado enfoques que complementen los Sistemas de Gestión del Aprendizaje (LMS) con frameworks que recopilen patrones que apoyen la enseñanza. De este modo, si ya se cuenta con un curso cuya gestión es a través de un LMS como Moodle, entonces se puede implementar un framework sobre el LMS que permita integrar patrones adecuados para el desarrollo del curso (ver Figura 3).

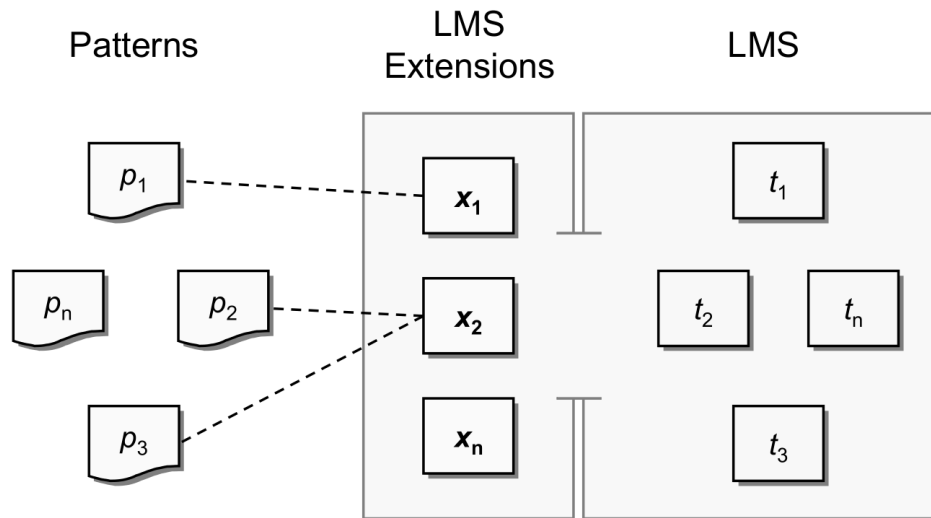


Figura 3: Implementación de los patrones de un framework en un LMS [Derntl y Calvo, 2011]

Sin embargo, también existen enfoques donde el framework permite al profesor adaptar un conjunto de patrones, los cuales son seleccionados en base a la metodología de trabajo que el framework propone, para luego así hacer uso de ellos en la organización de un curso de manera controlada, lo cual garantiza que la metodología validada del framework se lleve a cabo de la manera más prolija. Ejemplo de esto es la interfaz del manejador de patrones de la herramienta CEWebS [Derntl y Calvo, 2011] (ver Figura 4), la cual cuenta con una herramienta para que el profesor del curso configure qué componentes del LMS va a utilizar para llevar a cabo su curso. La idea detrás de esto es ocultar la parte técnica de la implementación de patrones en CEWebS, y abstraer al profesor de los mismos a través de una interfaz web que le permite seleccionar los patrones a usar tales como evaluación cruzada, discusión online, aprendizaje basado en proyectos, entre otros.

CEWebS

Reports Courses Services Patterns

Pattern Manager

Courses > Prototype Engineering

Currently Active Patterns

The following patterns are currently active in the this course. You may modify a pattern's configuration by clicking the *Edit* link of the pattern, which will redirect you to the pattern's configuration wizard. You can also view a summary of the current configuration of a pattern by clicking its *Info* link.

- Project-Based Learning Course [Edit] [Info]
- Preliminary Phases [Edit] [Info]
- Online Discussion [Edit] [Info]
- Project-Based Learning [Edit] [Info]
 - Team Building [Edit] [Info]
 - Team Workspaces [Edit] [Info]
 - Diary [Edit] [Info]
 - Project Milestone [Edit] [Info]
 - Project Milestone [Edit] [Info]
 - Project Milestone [Edit] [Info]
- Blended Evaluation [Edit] [Info]
 - Peer-Evaluation [Edit] [Info]
 - Self-Evaluation [Edit] [Info]
- Collect Feedback [Edit] [Info]
- Reaction Sheets [Edit] [Info]
- Questionnaire [Edit] [Info]

Employ a Pattern

Select an available pattern from the list box below and click on the Go button. You will then be redirected to the selected pattern's administration wizard.

--- select --- [Go]

Web Form Manager

Many patterns use web forms for user interaction (e.g. Questionnaire or Peer-Evaluation). Click the link below to manage the Web forms for this course.

→ Web Form Manager

Participants Data

Edit any of the course's participant data by clicking one of the links below.

→ Edit group information
→ Edit instructors
→ Edit participants

Figura 4: Pantalla principal de la interfaz de usuario de Patman (abreviación de *pattern manager*) [Derntl y Calvo, 2011]

Este último es la clase de framework que se implementará a lo largo de esta memoria, adaptándolo para la implementación de patrones adecuados y una metodología para la elaboración de tareas, de modo tal que guíe al profesor durante su uso, y garantice que éste siga una metodología adecuada para la tarea que está elaborando en el curso. A medida que redacta la tarea, el framework le brindará métricas al profesor, como una validación de que los objetivos de aprendizaje esperados para esa tarea se cumplan en base a diversas validaciones, como lo sería la revisión del código que resuelve el problema, entre otros.

2.3. Curso Introductorio de Programación

Un curso introductorio de programación es, en general, el primer acercamiento de un estudiante a los conceptos fundamentales de la computación [Omer U, 2021], tiene como enfoque entregar, a los estudiantes, los conceptos fundamentales de las ciencias de la computación, y son a su vez, cursos con altos niveles de reprobación, que pese a lo crucial que son en la formación del estudiante y de los futuros programadores, aún tienen muchos aspectos por mejorar, tanto en las tareas que tienen, como otros aspectos [Watson y Li, 2014]. También son conocidos en la literatura y en diversos currículos universitarios como CS0 [Lionelle *et al.*, 2020] y CS1 [Hertz, 2010], siendo CS2, CS3 y los que siguen, cursos que siguen la temática de las ciencias de la computación, pero que ya no tienen un carácter introductorio.

2.4. Tarea de Programación

Las tareas de programación son uno de los instrumentos con los cuales se evalúa, tanto los aprendizajes adquiridos de los estudiantes a lo largo del curso, como la puesta en práctica de los mismos. Generalmente involucran programar, sin embargo existen diversas adaptaciones según la forma en la que esté organizado el curso [Hertz, 2010]. Estas son el principal objeto de estudio de esta memoria, pues se busca lograr la elaboración de tareas de calidad, en base al entendimiento del impacto que estas pueden tener en el estudiante, y en los indicadores de si una tarea es apropiada o no para llevarse a cabo en un curso introductorio de programación, dado también su carácter formativo a lo largo de la experiencia educativa del estudiante.

Diversos autores han abordado la elaboración y medición de tareas de calidad, enfocándose tanto en factores emocionales del estudiante [Kinnunen y Simon, 2010, Layman *et al.*, 2007, Torrey, 2011, Vivian *et al.*, 2013], como en aspectos particulares de cualquier tarea en sí [Boye, 2020, Hundhausen *et al.*, 2015, Stevenson y Wagner, 2006]. En general, se destacan 3 aspectos principales:

- Las tareas deben tener alguna aplicación real, o bien, resolver un problema real que le dé sentido al tiempo que el estudiante invertirá en resolverla.
- Las tareas deben ser interesantes, un problema puede expresarse utilizando un contexto de la actualidad, de lo que los estudiantes en general considerarían interesante como lo son sus bandas musicales, juegos, tendencias, etc.
- Las tareas deben tener un nivel de dificultad adecuado, ni muy difíciles como para frustrar al estudiante, ni muy fáciles como para no cumplir con los objetivos de aprendizaje de la misma.

2.4.1. Dominio de un Problema en una Tarea

El dominio de un problema es un término de ingeniería para referirse a toda la información que define el problema, las restricciones de su solución, los objetivos que se desean lograr a la hora de abordarlo, el contexto donde el problema existe, y todas las reglas que definen la esencia del mismo. Este representa el entorno donde tanto el problema como sus soluciones propuestas se desenvuelven [Cunningham, 2014], y por lo tanto, engloba todo lo que se busca que el estudiante resuelva a lo largo del desarrollo de una tarea.

2.4.2. Factores de Importancia en una Tarea de Programación

Un estudio acerca de los factores que despiertan el interés e impulsan la elección de un estudiante en el desarrollo de una tarea [Torrey, 2011], indica que en el caso del interés, los factores que más interés generan de una tarea son (en orden decreciente de importancia):

- Que tenga gráficos
- Que tenga alguna utilidad en el mundo real
- Que sea entretenida
- Que sea desafiante
- Que sea fácil
- Que se relacione con algún hobby

Por otra parte, los factores que llevarían a un estudiante a elegir desarrollar una tarea por sobre otra, se encuentran (en orden decreciente de importancia):

- Que sea fácil
- Que tenga alguna utilidad en el mundo real
- Que sea entretenida
- Que sea desafiante
- Que se relacione con algún hobby
- Que tenga gráficos

Estos factores se resumen en el gráfico de la Figura 5.

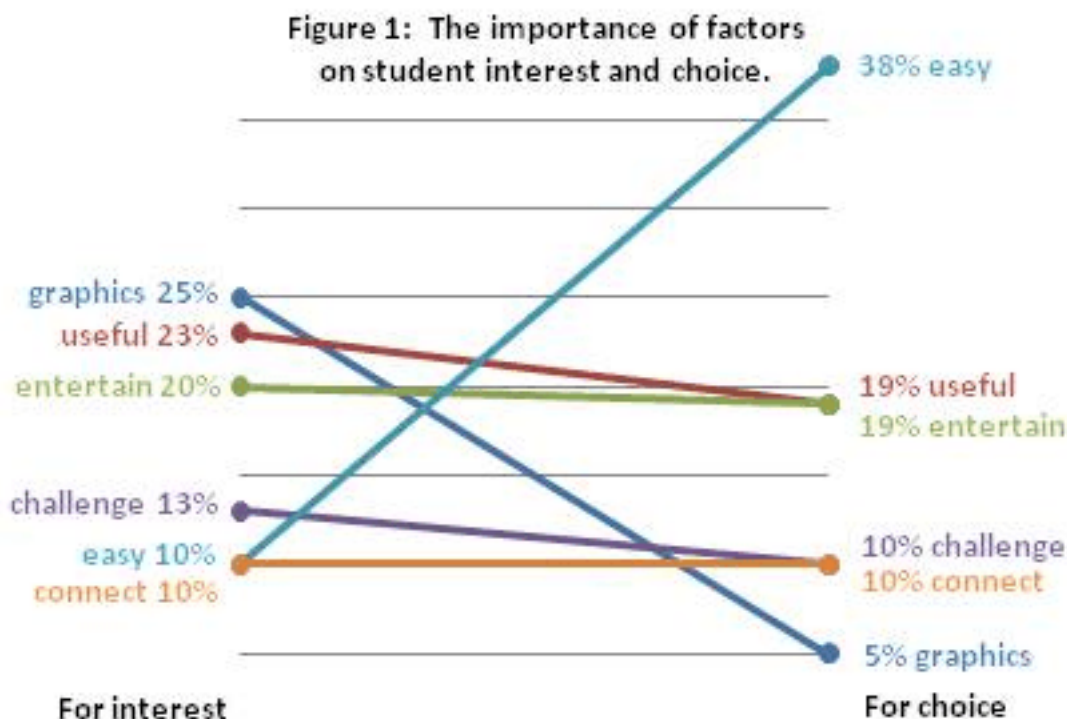


Figura 5: Nivel de importancia de los factores de una tarea, asignado por los mismos estudiantes según si les genera interés, o si les haría elegir desarrollar esa tarea por sobre otra [Torrey, 2011]

2.4.3. Reutilización de Tareas de Programación

La imperante necesidad de tener tareas nuevas cada semestre para un curso (pese a los cambios que la pandemia de COVID-19 ha provocado en algunos [Siegel *et al.*, 2021]), ha impulsado la investigación respecto a cómo reutilizar de forma eficiente las tareas de semestres anteriores [Craig y Morrison, 2021]. Sin embargo, proyectos como Moulinog [Yernaux *et al.*, 2020] han sido muy limitados en lograr esto, pues están limitados a, en base a una tarea, generar múltiples tareas que son en esencia lo mismo con la salvedad de cambiar ciertos valores y palabras, que no garantizan que una misma solución no sea esencialmente capaz de resolver dos de las versiones generadas.

Sin embargo, la idea de tener tareas base motiva a investigar acerca de si es buena idea reutilizar tareas, generando un pequeño banco de tareas de calidad, pero teniendo en cuenta que éstas deben ser del tipo *Tweakable assignments* [Craig y Morrison, 2021], es decir, la mayoría de sus partes pueden re-utilizarse, a excepción de algunas que, en base a ciertas especificaciones, deben modificarse en la nueva tarea de modo tal que una solución a la

tarea original, simplemente no funcione con la tarea nueva que se está elaborando, manteniendo así la calidad de la tarea original, y evitando llegar a aumentar las probabilidades de actos fraudulentos por parte de los estudiantes, quienes pueden hacerse de las soluciones de tareas anteriores [Simon, 2017].

2.5. Métricas de Software

Las métricas de software son valores computados con el objetivo de evaluar ciertas características del software desarrollado [Li y Cheung, 1987]. Existen diversas métricas de software las cuales se concentran en distintos aspectos. A continuación se explican las más relevantes para esta investigación.

2.6. Complejidad Ciclomática

La complejidad ciclomática es una métrica de software creada por Thomas McCabe, la cual mide el número de caminos linealmente independientes a través de una porción de código [Ebert *et al.*, 2016]. Hoy en día es una de las métricas más utilizadas en la industria para determinar la complejidad de entender y mantener un código, así como también la probabilidad de que éste tenga defectos. Un alto valor de esta métrica implica una densidad de defectos a lo largo de él, así como también, da cuenta de la cantidad de decisiones distintas que el programa debe tomar, y por tanto, lo complejo que es entender el problema subyacente que se intenta resolver.

La complejidad ciclomática de un programa se calcula mediante su grafo de ejecución, donde cada línea de código corresponde a un nodo del grafo, y cada nodo tendrá una arista hacia el nodo que sigue inmediatamente según la línea de ejecución del código. Una vez que está hecho el grafo de ejecución, la fórmula para la complejidad ciclomática es:

$$CC = E - N + 2P \quad (1)$$

Donde:

- CC : Complejidad Ciclomática
- E : Cantidad de aristas del grafo de ejecución
- N : Cantidad de nodos del grafo de ejecución
- P : Cantidad de componentes conexas del grafo

2.7. Métricas de Halstead

Maurice H. Halstead propuso una serie de métricas de software en su libro “Elements of Software Science” [Halstead, 1977], las cuales buscaban predecir de manera experimental cómo el hecho de escribir código también es una ciencia gobernada por leyes naturales. No obstante, posteriormente se realizaron experimentos que demuestran que el trabajo propuesto no representa realmente una ley natural [Hamer y Frewin, 1982]. Aún así, éstas métricas han sido ampliamente utilizadas en la industria para medir la mantenibilidad del software.

Halstead inicia su trabajo definiendo los siguientes parámetros para la implementación en código de cualquier algoritmo:

- η_1 : Número de *operadores* únicos que aparecen en la implementación
- η_2 : Número de *operandos* únicos que aparecen en la implementación
- N_1 : Uso total de todos los *operadores* que aparecen en la implementación
- N_2 : Uso total de todos los *operandos* que aparecen en la implementación
- $f_{1,j}$: Número de ocurrencias del j -ésimo *operador* más frecuente en la implementación, donde $j = 1, 2, \dots, \eta_1$
- $f_{2,j}$: Número de ocurrencias del j -ésimo *operando* más frecuente en la implementación, donde $j = 1, 2, \dots, \eta_2$

A partir de estos parámetros, se define el vocabulario η de la implementación como:

$$\eta = \eta_1 + \eta_2$$

Y el largo N de la implementación como:

$$N = N_1 + N_2$$

Basado en esto, Halstead plantea las métricas que se detallan a continuación:

2.7.1. Volumen de un Programa

Una característica importante de la implementación de cualquier algoritmo es su tamaño. Sin embargo, a medida que la implementación de éste es traducida de un lenguaje de programación a otro, su tamaño cambia. Por lo tanto, una métrica de tamaño que sólo considere la cantidad de caracteres de la implementación, no será suficientemente objetiva dado la diferencia de caracteres que pueden tener operandos u operadores similares en lenguajes de programación distintos.

Para abordar este problema, se debe notar que para cualquier implementación, existe un mínimo largo absoluto con el cual se puede representar el operador u operando más largo utilizado, éste mínimo es la representación en bits del mismo. El largo dependerá sólo de la cantidad de elementos en el vocabulario del programa, es decir, η . Por ejemplo, para $\eta = 8$ se requieren sólo 3 bits con los cuales se puedan representar todos los elementos del vocabulario, y a modo general, se requieren $\log_2 \eta$ bits para representar en su largo mínimo cualquiera de los elementos de un programa.

Esta interpretación entrega una dimensión en bits del volumen de cualquier programa, y permite medir el tamaño de cualquier implementación para cualquier algoritmo definiendo el volumen V como:

$$V = N \log_2 \eta \quad (2)$$

2.7.2. Volumen Potencial

Dado que traducir la implementación de un algoritmo de un lenguaje a otro implicará un cambio en el volumen del programa, se requiere otra métrica que de cuenta de la mínima forma en la que se puede expresar un algoritmo. Por ejemplo, si un lenguaje de programación ya cuenta con la subrutina o procedimiento que implementa un algoritmo, sólo se requiere llamarla y entregarle los parámetros correspondientes.

Para este caso, si se denotan los parámetros en su forma mínima absoluta, se puede definir que la mínima forma de implementación de cualquier algoritmo, nombrado de ahora en adelante como *volumen potencial* V^* , es:

$$V^* = (N_1^* + N_2^*) \cdot \log_2(\eta_1^* + \eta_2^*) \quad (3)$$

Donde:

- V^* : Volumen potencial
- N_1^* : Mínimo uso total de operadores de la implementación
- N_2^* : Mínimo uso total de operandos de la implementación
- η_1^* : Mínima cantidad de operadores únicos de la implementación
- η_2^* : Mínima cantidad de operandos únicos de la implementación

Dado que en la forma mínima, no se requiere repetición de operandos ni operadores, entonces:

$$N_1^* = \eta_1^*$$

$$N_2^* = \eta_2^*$$

Además, según lo planteado anteriormente, el mínimo número de operadores a utilizar serán sólo un operador para invocar a la subrutina o procedimiento, y otro para almacenar el resultado de la misma, por lo tanto:

$$\eta_1^* = 2$$

Con esto la ecuación 3 queda como:

$$V^* = (2 + \eta_2^*) \cdot \log_2(2 + \eta_2^*) \quad (4)$$

Donde η_2^* debería representar el número de distintos parámetros de input y output de la subrutina o procedimiento. Con esto, V^* es independiente del lenguaje en cual se exprese cualquier algoritmo, por lo tanto a diferencia de V , V^* no cambiará al traducir un algoritmo de un lenguaje a otro.

2.7.3. Nivel de un Programa

De manera intuitiva existe una idea del “nivel” que un programa tiene, el cual es determinado en base a la opinión de un grupo de expertos, basándose en que el nivel de un programa tiene un impacto en el esfuerzo de escribirlo, cometer errores en él y la facilidad con que puede entenderse. Sin embargo, esta métrica no puede quedar como una opinión, y con el fin de llevarla a algo cuantitativo, Halstead propone la siguiente definición para el *nivel de un programa* L como:

$$L = \frac{V^*}{V} \quad (5)$$

Lo que indica que la versión mínima en la que se puede escribir un algoritmo tendrá un nivel de 1, otras implementaciones con un mayor volumen un nivel menor, lo que implica que $L \leq 1$. Es importante notar que si se aplica esta métrica para evaluar qué tan fácil o difícil es entender un programa, ocurre que para una persona con un alto entendimiento del lenguaje de programación utilizado, será muy sencillo entender un programa con un bajo volumen, lo que conllevaría a que tenga un alto nivel. Por otro lado, para una persona menos fluída en el mismo lenguaje, será más sencillo entender un programa con un mayor volumen, lo que implicará un menor nivel de programa. Dicho esto, se plantea que la dificultad de entender un programa es inversamente proporcional al nivel del mismo.

Sin embargo, dada la ausencia de un valor conocido para el volumen potencial de una implementación (debido a que no existe un lenguaje de programación que implemente todos los algoritmos como subrutinas o procedimientos), es preferible obtener un cálculo del nivel del programa directamente de la implementación, sin hacer referencia a una posible subrutina que permita implementar el algoritmo en su forma mínima. Esto puede lograrse notando cómo impactan por separado los operadores y operandos en el nivel del programa.

El menor nivel de operadores que se puede utilizar es 2 (la invocación de una subrutina o procedimiento, y un operador para asignar su resultado en alguna variable). Por otro lado, no

hay un límite para la cantidad de operadores únicos que pueden haber. De aquí se desprende que:

$$L \approx \frac{\eta_1^*}{\eta_1} \quad (6)$$

Por otro lado, no existe un mínimo para los operandos: el que se repita mucho un mismo operando apoya a que el nivel del programa sea bajo. Este efecto puede medirse en base al radio del uso total de operandos únicos en la implementación, de donde se obtiene la segunda proporcionalidad:

$$L \approx \frac{\eta_2}{N_2} \quad (7)$$

Combinando las ecuaciones 6 y 7, se obtiene una versión alternativa para el cálculo del nivel de un programa.

$$\hat{L} = \frac{\eta_1^*}{\eta_1} \cdot \frac{\eta_2}{N_2} \quad (8)$$

Y sabiendo que η_1^* , la ecuación queda como:

$$\hat{L} = \frac{2}{\eta_1} \cdot \frac{\eta_2}{N_2} \quad (9)$$

Los experimentos de Halstead muestran que ambas ecuaciones son aceptables, sin embargo a lo largo de este documento se utilizará la ecuación 9.

2.7.4. Dificultad de un Programa

La dificultad de un programa según Halstead, es el inverso del nivel del mismo, entonces basándose en lo planteado en la sección 2.7.3, se define la dificultad de un programa D como:

$$D = \frac{1}{\hat{L}} = \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2} \quad (10)$$

2.7.5. Esfuerzo de Programación

El esfuerzo para desarrollar un programa particular corresponde, según Halstead, a la actividad mental requerida para escribirlo. Para calcularlo se debe considerar lo siguiente:

1. Se debe asumir que cualquier implementación de cualquier algoritmo, consiste en realizar N selecciones de un vocabulario de η elementos.
2. El cerebro es eficiente a la hora de realizar búsquedas, por lo tanto, se puede decir que es equivalente a hacer una búsqueda binaria, lo que implica que el cerebro realiza $\log_2 \eta$ comparaciones para la selección de cada uno de los elementos de la implementación [Halstead, 1977].

3. En base a los puntos anteriores, se puede determinar que un programa es generado mediante la realización de $N \cdot \log_2 \eta$ comparaciones mentales.
4. Recordando que el volumen de un programa se define como $V = N \log_2 \eta$, se determina que el volumen de un programa es también un contador del número de comparaciones mentales requeridas para generar un programa.
5. Cada comparación mental requiere un número de discriminaciones mentales, es decir, de descartar algunos elementos del vocabulario que no corresponden a lo que se busca implementar durante la búsqueda binaria del elemento correcto. Este número de discriminaciones mentales equivale a la dificultad de la tarea a realizar, y refuerza la idea de que el nivel de programación L es recíproco a la dificultad de programación.
6. Habiendo determinado que V equivale a la cantidad de comparaciones mentales a realizar, y el recíproco del nivel de programación $\frac{1}{L}$ es una medida del promedio de discriminaciones mentales requeridas para cada comparación, es posible plantear que el número total de discriminaciones mentales E requeridas para generar un programa es:

$$E = \frac{V}{L} \quad (11)$$

7. Recordando de la ecuación 5, si se reemplaza L por $\frac{V^*}{V}$, la fórmula del esfuerzo de programación se puede ver como:

$$E = \frac{V^2}{V^*} \quad (12)$$

Con esto se obtiene que el esfuerzo mental requerido para la implementación de cualquier algoritmo, está cuadráticamente relacionado con el volumen del mismo.

2.7.6. Tiempo Estimado de Programación

Según el trabajo de John Stroud en su obra "The Fine Structure of Psychological Time" [Stroud, 1967], existe una cantidad de tiempo requerido por el cerebro humano para realizar una discriminación de elementos, por lo que existe una cantidad de discriminaciones que se puede hacer por segundo. De aquí nace el número de Stroud S , el cual indica los límites para la cantidad de discriminaciones por segundo que puede hacer el cerebro humano, cuyo valor está acotado por $5 \leq S \leq 20$, y que según los experimentos de Halstead, su valor será considerado 18 a lo largo de este documento.

Dado que la cantidad de discriminaciones que el cerebro humano puede hacer está limitada por los límites del número de Stroud, y que la ecuación 12 del esfuerzo de programación tiene dimensiones de dígitos binarios por cantidad de discriminaciones, es posible estimar el tiempo que un ser humano tardaría en realizar la implementación de un algoritmo como:

$$T = \frac{E}{S} \quad (13)$$

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

3.1. Herramientas y Tecnologías a Utilizar

El framework consiste de una serie de tecnologías y herramientas utilizadas tanto para la aplicación de la metodología, como para el desarrollo y funcionamiento de la aplicación para editar y evaluar tareas. Estas herramientas y tecnologías se describen a continuación:

3.1.1. Google Trends

Es una herramienta proporcionada por Google que permite a sus usuarios conocer las tendencias de búsqueda en Internet. Además, se puede utilizar para comparar la popularidad relativa de un término de búsqueda a lo largo del tiempo y según diferentes localizaciones geográficas. Se basa en datos recopilados por el motor de búsqueda de Google, presentados en forma de gráficos y diagramas, para facilitar la comprensión de la popularidad de sus términos de búsqueda a lo largo del tiempo.

3.1.2. WebAssembly

Es un formato binario de instrucciones para una máquina virtual basada en pilas, diseñado para ser un punto de compilación portable para distintos lenguajes de programación. Esta tecnología permite desplegar programas en diferentes lenguajes tanto en aplicaciones de cliente como de servidor [W3C, 2022].

Gracias a WebAssembly, es posible ejecutar una biblioteca de Python en una aplicación Javascript para ser utilizada en el navegador, sin tener que instalar programas adicionales ni realizar configuraciones en el navegador del usuario.

3.1.3. Python

Es un lenguaje de programación interpretado fácil de aprender, con eficientes estructuras de datos de alto nivel y un enfoque simple pero efectivo hacia la programación orientada a objetos [Python Software Foundation, 2023]. Consta de múltiples módulos y es ampliamente utilizado en la industria para el desarrollo de aplicaciones.

Las tareas del curso IWI-131 de programación de la Universidad Técnica Federico Santa María deben ser resueltas en este lenguaje, y es a su vez el lenguaje principal que se estudia en el

presente trabajo.

3.1.4. Javascript

Es un lenguaje de programación interpretado basado en prototipos, multiparadigma y de un único hilo de ejecución. Es uno de los lenguajes de scripting más conocidos para el desarrollo web, pero también es utilizado ampliamente en la industria para otro tipo de aplicaciones y ambientes [Mozilla Contributors, 2021].

Su uso en conjunto con WebAssembly permite el desarrollo de aplicaciones de navegador que utilizan distintos lenguajes de programación a la vez, como por ejemplo, Python con Javascript.

3.1.5. Multimetrix

Es una biblioteca de Python para calcular métricas de código para programas en distintos lenguajes de programación [Weihmann, 2021]. Entre las métricas que calcula esta biblioteca se encuentran:

- Complejidad ciclomática
- Dificultad según Halstead
- Esfuerzo según Halstead
- Tiempo estimado de programación según Halstead
- Volumen según Halstead

3.1.6. Markdown

Es una herramienta de conversión de texto plano a HTML para creadores de contenido web. Permite crear textos en un formato sencillo de leer y escribir, para luego ser convertido en un código estructuradamente válido de HTML [Gruber, 2021].

Con Markdown es posible estructurar la redacción de distintos tipos de artículos, entre éstos, tareas de programación, lo que facilita su creación, distribución y estandarización a lo largo de las distintas personas que trabajan en la redacción de las mismas.

3.1.7. GitHub Pages

Es un ambiente provisto por GithubTM para desarrollar y publicar sitios web, haciendo posible alojar un sitio web estático de forma fácil, rápida y gratuita [Utomo y Falahah, 2020]. Con esto, es posible delegar la responsabilidad de alojar una aplicación web a GithubTM, y así no tener que contar con un servidor propio que deba ser manualmente mantenido.

3.1.8. Pyodide

Es una versión de Python y una colección de distintos módulos de python compilados a WebAssembly. Este proyecto ha llevado exitosamente el intérprete de Python a un módulo de WebAssembly, permitiendo así que un código de Python sea ejecutado sin problemas en un navegador web [Huffman, 2023]. Multimetric es una de las bibliotecas que puede ser ejecutada en un navegador gracias a este módulo.

3.1.9. React

Es un framework de Javascript, originalmente creado por FacebookTM para resolver los problemas de desarrollar interfaces de usuario complejas con datos que cambian en el tiempo. React cambió la forma en que las aplicaciones eran creadas logrando enormes avances en cómo se lleva a cabo el desarrollo web [Gackenheim, 2015].

3.2. Arquitectura de la Solución

El framework cuenta con un editor el cual tiene como objetivo permitir al profesor llevar a cabo el proceso de elaboración de una tarea de inicio a fin, aplicando la metodología de Ishvel. Este editor se encuentra funcionando es <https://vadokdev.github.io/Ishvel/>, y cuenta con la siguiente arquitectura:

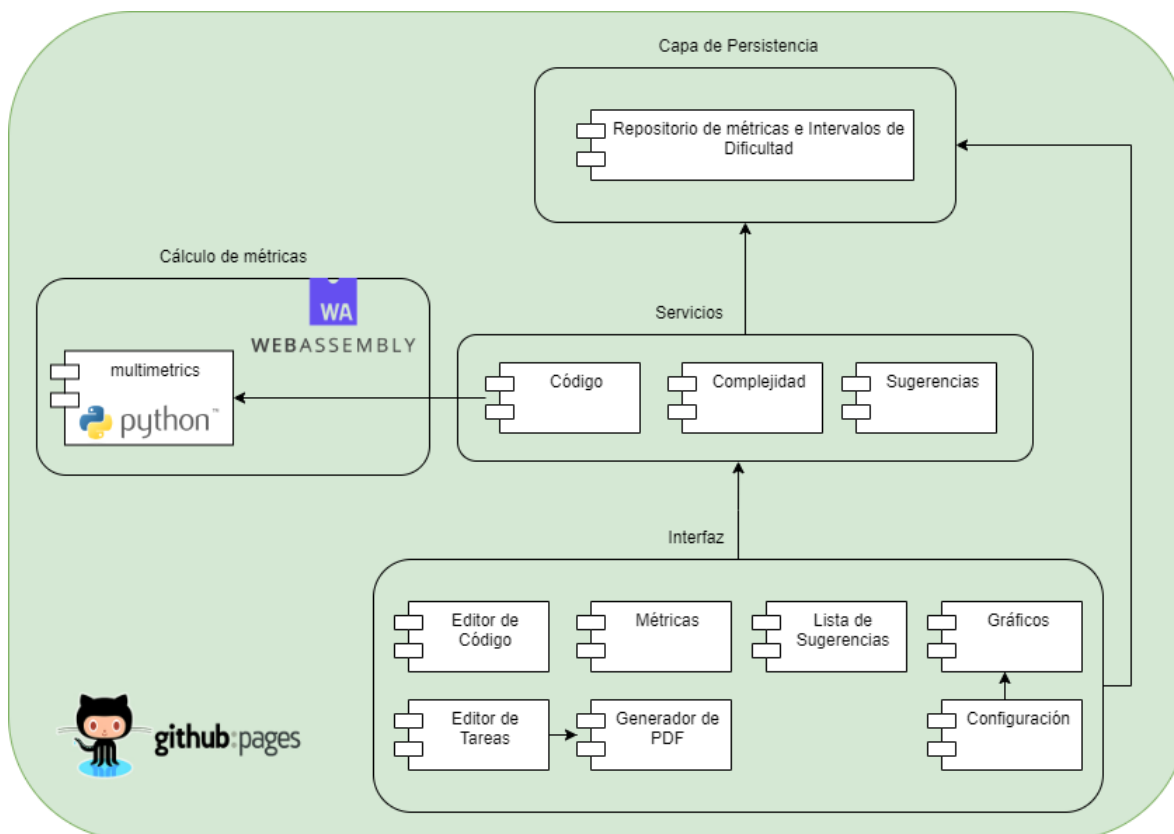


Figura 6: Arquitectura del editor de Ishvel.
Fuente: elaboración propia.

El software en sí está diseñado para funcionar perfectamente sin la necesidad de tener un servidor externo, es decir, la aplicación corre completamente en el navegador del usuario. Para lograr esto, hace uso de WebAssembly y Github Pages TM, donde el primero permite ejecutar código nativo de Python desde el navegador del usuario, y el segundo permite al usuario acceder al sitio web desde cualquier dispositivo con acceso a Internet.

Con respecto a los módulos del editor, éste se compone de 4 capas:

- **Cálculo de métricas:** Esta capa consiste de una implementación en Python gracias a WebAssembly que permite hacer uso del módulo `multimetrics`[Konrad Weihmann, 2023], de manera tal que al entregarle un código en texto plano, éste es capaz de calcular sus métricas y entregarlas al resto de capas de la aplicación
- **Servicios:** La capa de servicios es la encargada de llevar a cabo toda la lógica de la aplicación, orquestando las solicitudes de la interfaz para entregarle la información que ésta necesita para renderizar en la vista del cliente.

- **Capa de Persistencia:** Esta capa se encarga de almacenar los datos de las métricas históricas e intervalos de comparación de dificultades, con el fin de servirlos a la capa de servicios y que ésta haga su trabajo de cara a las solicitudes de la interfaz.
- **Interfaz:** Esta es la capa visual del software, la cual se encarga de tomar la información que obtiene desde la capa de servicios y mostrarla en el sitio web para ser visualizada por el usuario.

3.3. Framework

El marco de trabajo de Ishvel se propone con el fin de entregar una herramienta docente, la cual permite obtener una noción de la dificultad de una tarea antes de ser entregada al estudiantado para su resolución. Para ello, Ishvel se basa en el código que resuelve la tarea de manera correcta, extrayendo métricas del mismo y comparándolas con las métricas de tareas anteriores.

3.3.1. Metodología

Ishvel ofrece un acercamiento a entender la dificultad de una tarea en comparación a tareas anteriores, antes de publicarla a los estudiantes. Para esto, se propone la siguiente metodología de trabajo:

- Desde el editor de Ishvel, seleccionar las configuraciones adecuadas para el contenido de la tarea que se va a desarrollar, y el semestre y tipo de soluciones con la cual compararla. Para esto, se ofrecen las siguientes opciones:
 - **Contenido de la tarea:** Opción para determinar de qué contenido será la tarea a elaborar, lo que configura el editor para utilizar las métricas de tareas anteriores del contenido seleccionado.
 - **Semestre a comparar:** Opción para determinar de qué semestre serán las tareas del contenido seleccionado que se utilizarán para comparar.
 - **Tareas a comparar:** Opción para determinar si las métricas de la tarea a elaborar se compararán con las métricas de las soluciones de los profesores o de los estudiantes, del semestre seleccionado previamente.

Configuración

Contenido de la tarea

Secuenciales ▼

Semestre a comparar

2022-2 ▼

Tareas a comparar

Estudiantes ▼

Figura 7: Opciones de configuración del editor de Ishvel.

Fuente: elaboración propia.

- Una vez configurado el editor, iniciar por el título de la tarea, añadir alguna imagen representativa, y proceder a seguir el formato de ejemplo del editor:
 - **Contexto:** Contextualización corta respecto al problema que se está enfrentando, buscando que sea real, concisa e interesante. En caso de necesitar ideas para contextualizar el problema, se puede ver la *sugerencia por defecto* que el framework ofrece, la cual invita al docente a abrir Google Trends™ y ver situaciones de actualidad que pueden ser interesantes para el contexto de un problema.

Sugerencias

Inspiración

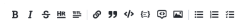
¿Necesitas ideas?, ¡Visita el sitio de tendencias de Google Trends para inspirarte!

[CERRAR](#) [VER MÁS](#)

Figura 8: Sugerencia inicial del editor de Ishvel.

Fuente: elaboración propia.

- **Instrucciones:** Sección con la lista de instrucciones de lo que el estudiante debe realizar, concentrándose en que éste pueda relacionar lo que el texto dice con lo que él debe escribir en su programa. La idea de separar la sección de instrucciones de la sección de contextualización, es poder tener un espacio centrado netamente en cómo el estudiante traducirá instrucciones en lenguaje humano a lenguaje máquina, sin tener distractores o tener que pasar por algún proceso cognitivo previo del cual desprender lo que se debe hacer, como lo sería por ejemplo mezclando el contexto con las instrucciones en un sólo texto.
- **Ejemplos:** Sección con ejemplos de la salida completa del programa. De esta manera, el estudiante tiene una guía visual de cómo debería ser su programa, y no tiene que imaginar por completo los detalles visuales de la salida del programa, para así concentrarse en programar.
- **Recomendaciones:** Sección extra con las recomendaciones que se pueden dar al estudiante para que realice un mejor trabajo, así como también, restricciones al mismo con las cuales el estudiante evite utilizar herramientas o bibliotecas que no están consideradas como parte de lo que se busca evaluar en la tarea.

B I 

Tarea N° X: Título de la Tarea

En este párrafo se debería dar el contexto del problema que buscamos resolver, se sugiere que éste sea corto para no aburrir al estudiante, interesante para captar su atención y que refleje un problema real, para que así el estudiante le encuentre sentido a resolver un problema utilizando la programación, de esta manera el proceso de aprendizaje y desarrollo que el estudiante vivirá gracias a esta tarea será mucho mejor. En el caso de no tener alguna idea sobre el contexto en que se puede basar esta tarea, se recomienda utilizar la sugerencia de Google Trends que está en la esquina inferior derecha de este editor, es una sugerencia por defecto y con ella se pueden sacar muy buenas ideas!

El formato de escritura de este editor es **Markdown**, el cual con la barra de herramientas de arriba, permite estilizar fácilmente la tarea, desde aspectos en la tipografía, hasta fórmulas matemáticas:

```
""KaTeX
d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
```

...

Objetivos

Aquí se deben listar los objetivos de aprendizaje que esta tarea busca, se recomienda utilizar una lista no enumerada para ello.

- * Utilizar los operadores aritméticos para realizar cálculos complejos
- * Aplicar los métodos de entrada y salida para pedir y mostrar datos al usuario

Instrucciones

En esta sección se deberían dar las instrucciones de lo que se busca desarrollar, se recomienda utilizar una lista enumerada, para facilitar el entendimiento al estudiante sobre lo que tiene que convertir de lenguaje humano a lenguaje máquina:

1. Solicite la coordenada x de la posición del camión
2. Solicite la coordenada y de la posición del camión
3. Calcule la distancia del camión con respecto a la coordenada (123, 456)
4. Finalmente, muestre en pantalla la distancia calculada

Ejemplos

...

Holal, por favor ingresa la posición del camión:

Coordenada X: 0
Coordenada Y: 25

...

La distancia al camión es de: 448.20754121277344

...

Recomendaciones

En esta sección se pueden dar indicios o hints a los estudiantes, que puedan ayudar en su proceso de desarrollo de la tarea, así como también, a evitar que utilicen herramientas o librerías que no son parte de los objetivos que la tarea busca evaluar.

- * Recuerda utilizar sólo los contenidos vistos en clase
- * El ejemplo es sólo explicativo, y no requiere que la salida de la tarea sea exactamente igual
- * Recuerda que la raíz de un número es equivalente a elevar ese número a 1/2



Tarea N° X: Título de la Tarea



En este párrafo se debería dar el contexto del problema que buscamos resolver, se sugiere que éste sea corto para no aburrir al estudiante, interesante para captar su atención y que refleje un problema real, para que así el estudiante le encuentre sentido a resolver un problema utilizando la programación, de esta manera el proceso de aprendizaje y desarrollo que el estudiante vivirá gracias a esta tarea será mucho mejor. En el caso de no tener alguna idea sobre el contexto en que se puede basar esta tarea, se recomienda utilizar la sugerencia de Google Trends que está en la esquina inferior derecha de este editor, es una sugerencia por defecto y con ella se pueden sacar muy buenas ideas!

El formato de escritura de este editor es **Markdown**, el cual con la barra de herramientas de arriba, permite estilizar fácilmente la tarea, desde aspectos en la tipografía, hasta fórmulas matemáticas:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Objetivos

Aquí se deben listar los objetivos de aprendizaje que esta tarea busca, se recomienda utilizar una lista no enumerada para ello.

- Utilizar los operadores aritméticos para realizar cálculos complejos
- Aplicar los métodos de entrada y salida para pedir y mostrar datos al usuario

Instrucciones

Figura 9: Formato de ejemplo de una tarea en el editor de Ishvel.
Fuente: elaboración propia.

- Una vez redactada la tarea, ésta se debe resolver en cualquier editor de código. Luego, la solución desarrollada debe copiarse y pegarse en el recuadro de Resolver Tarea. Una vez ingresada la solución, el editor automáticamente calculará las métricas de esa solución, y las comparará con la solución configurada previamente. Después de este paso, se mostrará en la sección Métricas cuál es la dificultad de esta solución en comparación a la tarea configurada por cada una de las métricas, así como también el tiempo estimado de resolución, y la dificultad promedio en comparación a la tarea configurada.

<> Resolver tarea

```
#Determinación de la posición de impacto
(cuando t=t_impacto) y cálculo de error
x_impacto=xi+vx*t_impacto
print('Proyectil impacta en coordenada x:',
round(x_impacto,2))
print('Se falló al Correcaminos por:',
round(x_impacto-x_coyote,2), 'm')
```

✎ Configuración

Contenido de la tarea

Secuenciales

Semestre a comparar

2022-2

Tareas a comparar

Estudiantes

📊 Métricas

⚠ Complejidad Ciclomática

Ligeramente más fácil

?

⚠ Esfuerzo

Mucho más fácil

?

⚠ Dificultad

Dificultad similar

?

⚠ Volumen

Mucho más difícil

?

⚠ Tiempo

90m

?

⚠ Promedio

Dificultad similar

?

Figura 10: Métricas de una solución según Ishvel.
Fuente: elaboración propia.

- Por último, para iterar sobre el enunciado y poder abordar así las métricas de la tarea que se está elaborando, se muestran en la sección Sugerencias una serie de sugerencias que se pueden aplicar, para mejorar la diferencia de dificultad entre la tarea que se está elaborando y la tarea seleccionada para comparar.

 **Sugerencias**

Esfuerzo de la tarea

La complejidad ciclomática en la práctica, es directamente proporcional a la cantidad de condiciones que debe evaluar el programa, es decir, cuando la solución de una tarea tiene una complejidad ciclomática demasiado alta, se sugiere modificar el enunciado de modo tal de remover las secciones que impliquen añadir más condiciones. Se pueden evaluar contenidos sin la necesidad de generar tantas condiciones específicas que lleven a bifurcaciones en el código, similarmente, en el caso de que la complejidad ciclomática sea demasiado baja, se pueden añadir casos específicos en el enunciado que requieran que el estudiante añada más bifurcaciones en su código a través de condiciones

CERRAR

Volumen de la Tarea

El volumen de un programa es directamente proporcional a la cantidad de operadores y operandos utilizados, así como también, al logaritmo en base 2 de la cantidad de operadores y operandos únicos que la implementación requiere. Cuando el volumen de una implementación es demasiado alto, se recomienda modificar aspectos del enunciado que requieran de demasiadas operaciones aritméticas distintas, así como también, que hagan uso de demasiadas variables, pues esto aumenta el vocabulario de la implementación y así la dimensión del volumen de la implementación resultante, del mismo modo, reducir la cantidad de cosas que exige el enunciado para que el código requerido como solución sea más corto, y así reducir el volumen en general. Para el caso en que el volumen sea demasiado bajo, añadir párrafos en el enunciado que requieran de diferentes operaciones aritméticas, y más datos para almacenar y tener que crear más variables, favorecerá mucho el aumento en el valor de esta métrica en la implementación final

CERRAR

Complejidad Ciclométrica

La complejidad ciclométrica en la práctica, es directamente proporcional a la cantidad de condiciones que debe evaluar el programa, es decir, cuando la solución de una tarea tiene una complejidad ciclométrica demasiado alta, se sugiere modificar el enunciado de modo tal de remover las secciones que impliquen añadir más condiciones. Se pueden evaluar contenidos sin la necesidad de generar tantas condiciones específicas que lleven a bifurcaciones en el código, similarmente, en el caso de que la complejidad ciclométrica sea demasiado baja, se pueden añadir casos específicos en el enunciado que requieran que el estudiante añada más bifurcaciones en su código a través de condiciones

CERRAR

Tiempo de la Tarea

El tiempo estimado de una implementación es directamente proporcional al esfuerzo requerido por la misma, en este caso, lo ideal es que el tiempo estimado de resolución no pase de 1 hora, para ello en el caso de tener una tarea que requiera demasiado tiempo, se debe ajustar el esfuerzo de la misma, éste crece exponencialmente al aumentar el volumen, y disminuye cuando aumenta el volumen potencial. Por lo mismo, para ajustar el esfuerzo de una implementación según si es muy grande o muy pequeño, se pueden aplicar las sugerencias anteriores para manejar el valor del volumen, y para el caso del volumen potencial, se sugiere jugar en el enunciado con la cantidad de elementos únicos que se trabajan, es decir, la cantidad de valores distintos que se necesitan almacenar, principalmente las entradas del problema a resolver. A medida que los inputs del problema aumentan, el volumen potencial aumentará, y a medida que estos disminuyan, el volumen potencial también lo hará y por consiguiente el esfuerzo aumentará

CERRAR

Figura 11: Sugerencias para una tarea según Ishvel.
Fuente: elaboración propia.

- Para descargar la tarea en PDF, se presiona el botón Descargar Tarea

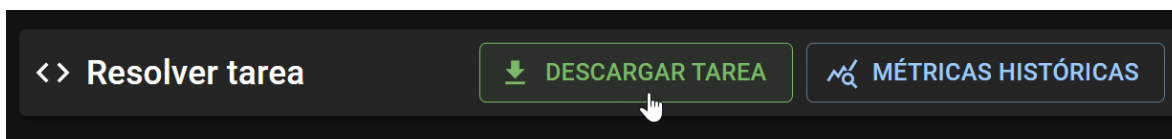


Figura 12: Botón de descargar tarea en el editor de Ishvel.

Fuente: elaboración propia.

Realizados estos pasos, se puede elaborar una tarea de un curso introductorio de programación teniendo una idea sobre su dificultad antes de publicarla a los estudiantes. Por otro lado, el framework busca también ofrecer métricas históricas para analizar el comportamiento de las tareas en el tiempo. Para esto se presiona el botón Métricas Históricas al lado del botón Descargar Tarea como se ve en la Figura 12, lo que desplegará las métricas históricas que el framework tiene cargadas.

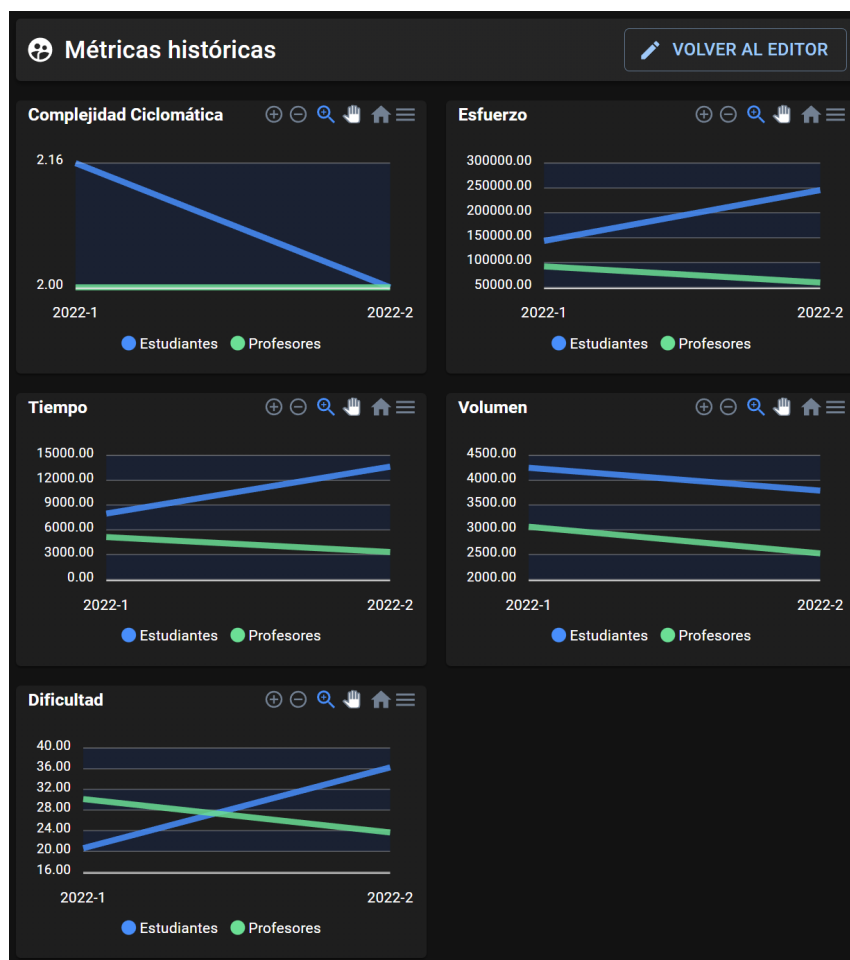


Figura 13: Sección de métricas históricas del framework Ishvel.

Fuente: elaboración propia.

3.3.2. Cálculo de Métricas

El framework trae consigo una herramienta llamada *metrics-research*, la cual hace uso del módulo *multimetricprog* [Konrad Weihmann, 2023] para calcular de manera programática las métricas tanto de una solución, como el promedio de métricas de varias soluciones al mismo tiempo.

Para utilizar la herramienta, se debe navegar en una terminal hasta el directorio que la contiene, y luego ejecutar el programa *main.py* que se encuentra en la carpeta *metrics-research*, utilizando alguno de los siguientes parámetros:

- **-f FILE:** Indica en FILE la ruta del archivo cuyas métricas se desean calcular.

```
>> python main.py -f .\2Secuenciales.py
{
  "cyclomatic_complexity": 2,
  "halstead_difficulty": 30.11764705882353,
  "halstead_effort": 92245.4225660487,
  "halstead_timerequired": 5124.745698113817,
  "halstead_volume": 3062.836296138336
}
```

Figura 14: Ejemplo de uso de la herramienta *metrics-research* con el parámetro *-f*.

Fuente: elaboración propia.

- **-d DIR:** Indica en DIR el directorio de la carpeta con las soluciones cuyo promedio de métricas se desea calcular.

```
>> python main.py -d .\assignments\teachers\2022-1\
.\assignments\teachers\2022-1\2Secuenciales.py
.\assignments\teachers\2022-1\3 Condicionales.py
.\assignments\teachers\2022-1\4 Ciclos.py
.\assignments\teachers\2022-1\5 Funciones.py
.\assignments\teachers\2022-1\6 Strings.py
.\assignments\teachers\2022-1\7 Listas.py
.\assignments\teachers\2022-1\8 Diccionesarios.py
.\assignments\teachers\2022-1\9 Archivos.py
{
  "cyclomatic_complexity": 15.25,
  "halstead_difficulty": 33.960229021981554,
  "halstead_effort": 105476.20398288252,
  "halstead_timerequired": 5859.78911016014,
  "halstead_volume": 3123.400548322863
}
```

Figura 15: Ejemplo de uso de la herramienta *metrics-research* con el parámetro *-d*.

Fuente: elaboración propia.

- **-h:** Se consulta la documentación de los parámetros del programa.

```
>> python main.py -h
usage: main.py [-h] [-f FILE] [-d DIR]

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  directory of the homework whose metrics you want to calculate
  -d DIR, --dir DIR     folder with the homeworks whose average metrics you want to calculate (deep search)
```

Figura 16: Ejemplo de uso de la herramienta metrics-research con el parámetro -h.
Fuente: elaboración propia.

3.3.3. Determinar la Diferencia de Dificultad entre Tareas

La diferencia de dificultad entre tareas, se calcula utilizando el promedio de los intervalos a los que pertenezca la diferencia porcentual de cada métrica, sin considerar la métrica de **tiempo de resolución**. Para determinar esto, se realiza el siguiente procedimiento para cada métrica entre 2 tareas:

1. Se define un conjunto de **intervalos de comparación de dificultad**. Estos corresponden a 5 intervalos que categorizan la diferencia de dificultad entre 2 tareas, utilizando las diferencias porcentuales de las métricas de las mismas.
2. Se calcula la diferencia porcentual entre los valores de la métrica con la que se esté trabajando para ambas tareas.
3. Se determina el intervalo de comparación de dificultad al que pertenece la diferencia porcentual calculada

Una vez que se obtiene el intervalo de comparación de dificultad al que pertenecen cada una de las métricas de ambas tareas, se promedian los resultados y se redondea al entero más cercano. El valor de este resultado indicará la diferencia de dificultad entre ambas tareas, según la siguiente clasificación:

1. Mucho más fácil
2. Más fácil
3. Similar
4. Más difícil
5. Mucho más difícil

Para entender mejor esta parte de la metodología, a continuación se presenta un ejemplo. Sean los intervalos de comparación de dificultad para las métricas consideradas:

Intervalo	Complejidad Ciclomática	Dificultad H.	Esfuerzo	Volumen
1 - mucho más fácil	-inf %, -50 %	-inf %, -25 %	-inf %, -75 %	-inf %, -35 %
2 - más fácil	-49 %, -5 %	-24 %, -15 %	-74 %, -25 %	-34 %, 5 %
3 - similar	-4 %, 25 %	-14 %, 5 %	-24 %, 15 %	6 %, 30 %
4 - más difícil	26 %, 70 %	6 %, 50 %	16 %, 65 %	31 %, 55 %
5 - mucho más difícil	71 %, inf %	51 %, inf %	66 %, inf %	56 %, inf %

Tabla 1: Ejemplo de intervalos de comparación de dificultad para todas las métricas, a excepción del tiempo estimado.

Fuente: elaboración propia

Y las métricas de 2 tareas del mismo contenido, junto con sus respectivas diferencias porcentuales:

	Complejidad Ciclomática	Dificultad H.	Esfuerzo	Volumen
	16	30	2255	730
	17	44	1520	342
Diferencia Porcentual	6.25 %	46.66 %	-32.59 %	-53.15

Tabla 2: Ejemplo de métricas de dos tareas del mismo contenido.

Fuente: elaboración propia

Se determina a qué intervalo pertenece la diferencia porcentual de cada una de las métricas, obteniendo los siguientes valores:

- Complejidad Ciclomática: **3 - similar**
- Dificultad H.: **4 - más difícil**
- Esfuerzo: **2 - más fácil**
- Volumen: **1 - mucho más fácil**

Finalmente, se calcula la diferencia de dificultad entre ambas tareas como el promedio redondeado de las categorías de las métricas, es decir, 3. Por lo tanto, aplicando la metodología del framework Ishvel, ambas tareas tienen una dificultad similar, o dicho de otra manera, la resolución de la tarea que se está elaborando, tiene una dificultad similar a la de una tarea desarrollada previamente del mismo contenido.

3.3.4. Determinar los Intervalos de Comparación de Dificultad entre Tareas

Los intervalos de comparación de dificultad mencionados anteriormente, son los que se utilizan para poder extraer información sobre la diferencia porcentual de una métrica entre

2 tareas. Para determinarlos, se requiere primero plantear qué dificultades comparativas y métricas se utilizarán en el proceso:

Sean las dificultades comparativas:

1. Mucho más fácil
2. Más fácil
3. Similar
4. Más difícil
5. Mucho más difícil

Y las métricas contempladas para determinar la dificultad entre 2 tareas:

1. Complejidad ciclomática
2. Dificultad según Halstead (Dificultad H.)
3. Esfuerzo
4. Volumen

Lo que se busca es definir:

- LI_{dm} : Límite **inferior** de la dificultad comparativa d asociado a la métrica m , con $d = \{1, 2, 3, 4, 5\}$ para cada dificultad comparativa respectivamente, y $m = \{1, 2, 3, 4\}$ para cada métrica, sin contar la métrica de tiempo estimado.
- LS_{dm} : Límite **superior** de la dificultad comparativa d asociado a la métrica m , con $d = \{1, 2, 3, 4, 5\}$ para cada dificultad comparativa respectivamente, y $m = \{1, 2, 3, 4\}$ para cada métrica, sin contar la métrica de tiempo estimado.

Para determinar los intervalos de comparación de dificultad, se requiere de una evaluación por un conjunto de expertos en programación, que pueden ser docentes, ayudantes docentes o personas con experiencia en el área de la informática. Una vez que se cuenta con el conjunto de expertos, se les debe entregar 2 tareas que evalúen contenidos similares, pero que fueron desarrolladas durante periodos lectivos distintos. El panel de expertos tomará ambas tareas, y evaluará según su juicio la dificultad de una tarea en comparación con la otra, siendo las posibles respuestas:

1. La segunda tarea es mucho más fácil que la primera tarea
2. La segunda tarea es más fácil que la primera tarea
3. La segunda tarea tiene una dificultad similar a la primera tarea
4. La segunda tarea es más difícil que la primera tarea
5. La segunda tarea es mucho más difícil que la primera tarea

Una vez obtenidas las respuestas del conjunto de expertos, se debe calcular la diferencia porcentual entre cada una de las métricas de ambas tareas, sin considerar la métrica de tiempo estimado, puesto que ésta es directamente proporcional a la métrica de esfuerzo. Se debe repetir el proceso con varias tareas para obtener la mayor información posible, especialmente con tareas cuya dificultad comparativa sea distinta a la de las otras tareas ya evaluadas. El objetivo de obtener esta información, es relacionar las diferencias porcentuales con los niveles de diferencias de dificultad entre tareas, de manera de tener para cada métrica distintos intervalos de diferencias porcentuales, a partir de los cuales determinar la dificultad de una tarea en comparación con otra.

A continuación se muestra un ejemplo de la información resultante al término de este paso:

Tareas	Comparación de dificultad	Complejidad ciclomática (dif %)	Dificultad H. (dif %)	Esfuerzo (dif %)	Volumen (dif %)
1 - 2	1 - mucho más fácil	6,25 %	43,69 %	-32,61 %	-32,61 %
3 - 4	2 - más fácil	-37,50 %	-2,86 %	-32,65 %	-32,65 %
5 - 6	3 - similar	-18,75 %	-54,06 %	-76,65 %	-76,65 %
7 - 8	1 - mucho más fácil	0,00 %	-21,46 %	-35,27 %	-35,27 %
9 - 10	5 - mucho más difícil	54,55 %	66,39 %	250,00 %	515,61 %
11 - 12	1 - mucho más fácil	50,00 %	79,59 %	-2,34 %	-2,34 %
13 - 14	2 - más fácil	40,00 %	0,00 %	-15,12 %	-15,12 %
15 - 16	3 - similar	26,32 %	54,55 %	3,46 %	3,46 %
17 - 18	1 - mucho más fácil	43,69 %	50,00 %	-32,61 %	66,39 %
19 - 20	5 - mucho más difícil	-2,86 %	40,00 %	-32,65 %	79,59 %
21 - 22	1 - mucho más fácil	-54,06 %	26,32 %	-76,65 %	0,00 %
23 - 24	2 - más fácil	-21,46 %	43,69 %	-35,27 %	54,55 %
25 - 26	3 - similar	66,39 %	-2,86 %	515,61 %	50,00 %
27 - 28	1 - mucho más fácil	79,59 %	-54,06 %	-2,34 %	40,00 %
29 - 30	5 - mucho más difícil	3,87 %	-21,46 %	-15,12 %	26,32 %

Tabla 3: Ejemplo de la comparación de dificultad entre tareas, junto con las diferencias porcentuales de cada una de sus métricas.

Fuente: elaboración propia

Ahora, lo que se busca es determinar 5 intervalos de diferencias porcentuales para cada métrica, donde cada intervalo corresponde a una de las 5 dificultades comparativas posibles, para así poder comparar las métricas de distintas tareas entre sí. Para esto, se debe seguir el siguiente procedimiento para cada una de las métricas:

1. Se crea una tabla con 5 filas y 5 columnas, donde las filas corresponden a los intervalos de las dificultades comparativas, y las columnas corresponden a las métricas asociadas a cada intervalo.

Comparación de Dificultad	Complejidad Ciclomática	Dificultad H.	Esfuerzo	Volumen
1 - mucho más fácil	LI_{11}	LI_{12}	LI_{13}	LI_{14}
	LS_{11}	LS_{12}	LS_{13}	LS_{14}
2 - más fácil	LI_{21}	LI_{22}	LI_{23}	LI_{24}
	LS_{21}	LS_{22}	LS_{23}	LS_{24}
3 - similar	LI_{31}	LI_{32}	LI_{33}	LI_{34}
	LS_{31}	LS_{32}	LS_{33}	LS_{34}
4 - más difícil	LI_{41}	LI_{42}	LI_{43}	LI_{44}
	LS_{41}	LS_{42}	LS_{43}	LS_{44}
5 - mucho más difícil	LI_{51}	LI_{52}	LI_{53}	LI_{54}
	LS_{51}	LS_{52}	LS_{53}	LS_{54}

Tabla 4: Tabla base de intervalos por métrica para cada dificultad comparativa.

Fuente: elaboración propia

2. En el primer y último intervalo de la métrica, colocar -inf % e inf % respectivamente.

Comparación de Dificultad	Complejidad Ciclomática	Dificultad H.	Esfuerzo	Volumen
1 - mucho más fácil	-inf %			
2 - más fácil				
3 - similar				
4 - más difícil				
5 - mucho más difícil				
	inf %			

Tabla 5: Tabla base de intervalos por métrica para cada dificultad comparativa con los límites máximos y mínimos del primer y último intervalo listos.

Fuente: elaboración propia

3. A partir del intervalo de la primera dificultad comparativa “mucho más fácil”, se calcula el límite superior restante como el mínimo entre la máxima diferencia porcentual de

esa dificultad comparativa, y la mínima diferencia porcentual de la dificultad comparativa siguiente, en este caso -37.50 %.

Comparación de Dificultad	Complejidad Ciclomática	Dificultad H.	Esfuerzo	Volumen
1 - mucho más fácil	-inf %	-inf %	-inf %	-inf %
	-37,50 %			
2 - más fácil				
3 - similar				
4 - más difícil				
5 - mucho más difícil				
	inf %	inf %	inf %	inf %

Tabla 6: Tabla base de intervalos por métrica para cada dificultad comparativa con el primer intervalo listo.

Fuente: elaboración propia

- Se continúa con el intervalo de la siguiente dificultad comparativa, donde su límite inferior se calcula como el límite superior anterior + 0.01 %, lo que en este caso resulta en -37.49 %.

Comparación de Dificultad	Complejidad Ciclomática	Dificultad H.	Esfuerzo	Volumen
1 - mucho más fácil	-inf %	-inf %	-inf %	-inf %
	-37,50 %			
2 - más fácil	-37,49 %			
3 - similar				
4 - más difícil				
5 - mucho más difícil				
	inf %	inf %	inf %	inf %

Tabla 7: Tabla base de intervalos por métrica para cada dificultad comparativa con el límite inferior del segundo intervalo.

Fuente: elaboración propia

- Se repiten los pasos 3 y 4 para los intervalos posteriores.

Comparación de Dificultad	Complejidad Ciclomática	Dificultad H.	Esfuerzo	Volumen
1 - mucho más fácil	-inf %	-inf %	-inf %	-inf %
	-37,50 %			
2 - más fácil	-37,49 %			
	-18,75 %			
3 - similar	-18,74 %			
	66,39 %			
4 - más difícil				
5 - mucho más difícil				
	inf %	inf %	inf %	inf %

Tabla 8: Tabla base de intervalos por métrica para cada dificultad comparativa casi completa para una métrica.

Fuente: elaboración propia

6. En caso de que falte información sobre alguna de las dificultades comparativas, posiblemente porque de la muestra de tareas para el conjunto de expertos, ningún par de tareas fue considerado dentro de alguna dificultad comparativa específica, se debe seguir el siguiente procedimiento:

- Si falta información para la dificultad comparativa 1:
 - Se utiliza el límite inferior de la dificultad comparativa 2 menos 0.01 %, es decir: $LI_{2m} - 0,01 \%$, como límite superior. Si no está definido LI_{2m} , se considera la menor diferencia porcentual de la dificultad comparativa 2 menos 0.01 % como límite superior.
- Si falta información para la dificultad comparativa 2:
 - Se calcula el límite inferior como el límite superior de la dificultad comparativa 1 más 0.01 %, y el límite superior como el límite superior de la dificultad comparativa 1 más 0.01 %, sumado a la distancia entre el límite inferior y superior de la dificultad comparativa 4, es decir:

$$LI_{2m} = LS_{1m} + 0,01 \%$$

$$LS_{2m} = LS_{1m} + 0,01 \% + abs(LI_{4m} - LS_{4m})$$

- Si falta información para la dificultad comparativa 3:
 - Se utiliza el límite superior de la dificultad comparativa 2 más 0.01 %, es decir: $LS_{2m} + 0,01 \%$, como límite inferior.
 - Se utiliza el límite inferior de la dificultad comparativa 4 menos 0.01 %, es decir: $LI_{4m} - 0,01 \%$, como límite superior.
- Si falta información para la dificultad comparativa 4:

- Se calcula el límite inferior como el límite superior de la dificultad comparativa 3 más 0.01 %, y el límite superior como el límite superior de la dificultad comparativa 3 más 0.01 %, sumado a la distancia entre el límite inferior y superior de la dificultad comparativa 2, es decir:

$$LI_{4m} = LS_{3m} + 0,01 \%$$

$$LS_{4m} = LS_{3m} + 0,01 \% + abs(LI_{2m} - LS_{2m})$$

- Si falta información para la dificultad comparativa 5:
 - Se utiliza el límite superior de la dificultad comparativa 4 más 0.01 %, es decir: $LS_{4m} + 0,01 \%$, como límite inferior. Si no está definido LS_{4m} , se considera la mayor diferencia porcentual más 0.01 % como límite inferior.
- Si ya se definió el límite inferior de la dificultad comparativa d , pero no hay información para definir el superior:
 - Se toma 20 como el tamaño del intervalo, es decir, se calcula el límite superior como $LS_{dm} = LI_{dm} + 20$, dado que 20 sería el tamaño del intervalo si todos los intervalos fuesen iguales.
- Si no hay información para definir los límites de la dificultad comparativa d , se debe continuar aplicando la heurística al resto de dificultades comparativas, hasta que hayan suficientes límites definidos como para definir los de la dificultad comparativa d .

7. Se aplican los procedimientos del paso 6 para terminar de rellenar la tabla, y con esto entonces se obtiene la tabla de intervalos de comparación de dificultad para la métrica de complejidad ciclomática.

Comparación de Dificultad	Complejidad Ciclométrica	Dificultad H.	Esfuerzo	Volumen
1 - mucho más fácil	-inf %	-inf %	-inf %	-inf %
	-37,50 %			
2 - más fácil	-37,49 %			
	-18,75 %			
3 - similar	-18,74 %			
	66,39 %			
4 - más difícil	66,40 %			
	85,14 %			
5 - mucho más difícil	85,15 %			
	inf %	inf %	inf %	inf %

Tabla 9: Tabla base de intervalos por métrica para cada dificultad comparativa para la métrica de complejidad ciclométrica.

Fuente: elaboración propia

3.3.5. Sugerencias

Dado que el framework da indicios de la dificultad de la tarea en base a su solución, a continuación se presentan estrategias para interpretar cada una de las métricas planteadas en este documento y tomar acciones para ajustarlas:

- **Complejidad ciclomática:** En la práctica, la complejidad ciclomática, es directamente proporcional a la cantidad de condiciones que debe evaluar el programa, es decir, cuando la solución de una tarea tiene una complejidad ciclomática demasiado alta, se sugiere modificar el enunciado de modo tal de remover las secciones que impliquen añadir más condiciones. Se pueden evaluar contenidos sin la necesidad de generar tantas condiciones específicas que lleven a bifurcaciones en el código. Similarmente, en el caso de que la complejidad ciclomática sea demasiado baja, se pueden añadir casos específicos en el enunciado que requieran que el estudiante añada más bifurcaciones en su código a través de condiciones.
- **Volumen:** El volumen de un programa es directamente proporcional a la cantidad de operadores y operandos utilizados, así como también, al logaritmo en base 2 de la cantidad de operadores y operandos únicos que la implementación requiere. Cuando el volumen de una implementación es demasiado alto, se recomienda modificar aspectos del enunciado que requieran de demasiadas operaciones aritméticas distintas, así como también, que hagan uso de demasiadas variables, pues esto aumenta el vocabulario de la implementación y así la dimensión del volumen de la implementación resultante. Del mismo modo, reducir la cantidad de exigencias del enunciado para que el código requerido como solución sea más corto, y así reducir el volumen en general. Para el caso en que el volumen sea demasiado bajo, añadir requerimientos en el enunciado que requieran de diferentes operaciones aritméticas, y más datos para almacenar y tener que crear más variables, favorecerá mucho el aumento en el valor de esta métrica en la implementación final.
- **Esfuerzo:** El esfuerzo crece exponencialmente al aumentar el volumen, y disminuye cuando aumenta el volumen potencial. Por lo mismo, para ajustar el esfuerzo de una implementación según si es muy grande o muy pequeño, se pueden aplicar las sugerencias anteriores para manejar el valor del volumen. Para el caso del volumen potencial, se sugiere modificar el enunciado para variar la cantidad de elementos únicos que se trabajan, es decir, la cantidad de valores distintos que se necesitan almacenar, principalmente las entradas del problema a resolver. A medida que los inputs del problema aumentan, el volumen potencial aumentará, y a medida que estos disminuyan, el volumen potencial también lo hará y por consiguiente el esfuerzo aumentará.
- **Dificultad:** La dificultad de un programa se maneja según el vocabulario η y el largo N del mismo. Cuando la cantidad de operadores únicos aumenta, o cuando aumenta el total de operadores únicos utilizados, la dificultad del programa aumenta, mientras que cuando la cantidad de operandos únicos aumenta, la dificultad disminuye. Para

ajustar el valor de la dificultad de una implementación según el enunciado de una tarea, se sugiere incluir en los párrafos aspectos del enunciado que requieran cálculos con operadores aritméticos variados, o bien, hacer uso de distintos valores a considerar a la hora de realizar el programa. Esto puede llevarse a cabo con distintos inputs, o bien, con distintos valores a considerar a la hora de realizar condiciones, poner límites, precios en el caso de programas que requieran tiendas, etc.

- **Tiempo:** El tiempo estimado de una implementación es directamente proporcional al esfuerzo requerido por la misma. En este caso, lo idóneo es que el tiempo estimado de resolución no pase de una hora. Para ello en el caso de tener una tarea que requiera demasiado tiempo, se recomienda aplicar las sugerencias de la métrica de esfuerzo para hacer los ajustes necesarios.

3.3.6. Actualización de Métricas Semestrales

En el editor del framework, se encuentra el directorio `src/core/data`, que contiene un conjunto de archivos encargados de mantener los datos con los que funciona el software.

```
>> ls .\src\core\data\  
  
Length Name  
-----  
    619 comparisionIntervals.json  
   4346 studentAssignments.json  
   4100 teacherAssignments.json
```

Figura 17: Archivos de datos del editor.
Fuente: elaboración propia.

En particular, los archivos `studentAssignments.json` y `teacherAssignments.json` contienen, respectivamente, las métricas promedio de las soluciones de todos los estudiantes, y las métricas individuales de las soluciones de los profesores a las tareas de los semestres estudiados.

Para actualizar las métricas semestrales, se debe primero calcular el promedio de métricas de las tareas de los estudiantes, así como también las métricas de las soluciones de los profesores a las mismas, siguiendo los pasos de la sección 3.3.2. Luego se deben añadir estos datos al archivo respectivo para que sean desplegadas en las métricas históricas del editor.

Ambos archivos comparten la siguiente estructura:

```
[
  {
    "semester": "2022-1",
    "contents": {
      "cyclomatic_complexity": 1,
      "halstead_difficulty": 2,
      "halstead_effort": 3,
      "halstead_timerequired": 4,
      "halstead_volume": 5,
    }
    ...
  }
  ...
]
```

Figura 18: Formato de los archivos de métricas del editor.
Fuente: elaboración propia.

La cual se compone de una lista de objetos con las siguientes propiedades:

- **semester:** Semestre al cual corresponden las métricas
- **contents:** Objeto cuyo nombre debe ser el contenido de la tarea, y que sus propiedades serán las métricas de la misma. Para trabajar con los 8 contenidos evaluados en los semestres 2022-1 y 2022-2, se utilizaron como contenidos:

- **dicts:** Diccionarios
 - **files:** Archivos
 - **lists:** Listas
 - **functions:** Funciones
 - **strings:** Strings
 - **loops:** Ciclos
 - **conditionals:** Condicionales
 - **sequential:** Secuenciales
- **cyclomatic_complexity:** Complejidad ciclomática de la tarea del contenido correspondiente
 - **halstead_difficulty:** Dificultad según Halstead de la tarea del contenido correspondiente
 - **halstead_effort:** Esfuerzo de la tarea del contenido correspondiente
 - **halstead_timerequired:** Tiempo estimado requerido para desarrollar la tarea del contenido correspondiente
 - **halstead_volume:** Volumen de la tarea del contenido correspondiente

Para añadir las métricas de una nueva tarea, ya sea el promedio de las tareas de los estudiantes, o de la solución de los profesores, se debe añadir una nueva llave debajo de las métricas de la última tarea del semestre en curso, esta llave debe tener el nombre del contenido cuyas métricas fueron calculadas, y dentro de esta llave, debe pegarse el resultado de las métricas calculadas con la herramienta *metrics-research*.

```
1  [
2
3
4
5  {
6    "semester": "2022-2",
7    "dicts": {
8      "cyclomatic_complexity": 17,
9      "halstead_difficulty": 44.36666666666667,
10     "halstead_effort": 152032.16753603518,
11     "halstead_timerequired": 8446.231529779732,
12     "halstead_volume": 3426.720530489147
13   }
14 }
15 ]
16
17
18
19
```

```
1  [
2  {
3    "semester": "2022-2",
4    "dicts": {
5      "cyclomatic_complexity": 17,
6      "halstead_difficulty": 44.36666666666667,
7      "halstead_effort": 152032.16753603518,
8      "halstead_timerequired": 8446.231529779732,
9      "halstead_volume": 3426.720530489147
10    },
11    "files": {
12      "cyclomatic_complexity": 15,
13      "halstead_difficulty": 40.73684210526316,
14      "halstead_effort": 97468.19980974238,
15      "halstead_timerequired": 5414.899989430132,
16      "halstead_volume": 2392.630227887733
17    }
18  }
19 ]
```

Figura 19: Ejemplo de como añadir las métricas de la tarea de archivos del semestre 2022-2 en el archivo de métricas del editor.

Fuente: elaboración propia.

En el caso de necesitar añadir un nuevo semestre, se debe añadir un objeto nuevo a la lista, el cual siga el formato de la figura 18, y entonces añadir allí las métricas según el contenido del que se disponga información.

```
1  [
2
3
4
5  {
6    "semester": "2022-2",
7    "dicts": {
8      "cyclomatic_complexity": 17,
9      "halstead_difficulty": 44.36666666666667,
10     "halstead_effort": 152032.16753603518,
11     "halstead_timerequired": 8446.231529779732,
12     "halstead_volume": 3426.720530489147
13   }
14 }
15 ]

1  [
2  {
3    "semester": "2022-2",
4    "dicts": {
5      "cyclomatic_complexity": 17,
6      "halstead_difficulty": 44.36666666666667,
7      "halstead_effort": 152032.16753603518,
8      "halstead_timerequired": 8446.231529779732,
9      "halstead_volume": 3426.720530489147
10    }
11  },
12  {
13    "semester": "2022-1",
14    "files": {
15      "cyclomatic_complexity": 15,
16      "halstead_difficulty": 40.73684210526316,
17      "halstead_effort": 97468.19980974238,
18      "halstead_timerequired": 5414.899989430132,
19      "halstead_volume": 2392.630227887733
20    }
21  }
22 ]
```

Figura 20: Ejemplo de como añadir las métricas de la tarea de archivos del semestre 2022-3 en el archivo de métricas del editor.

Fuente: elaboración propia.

3.3.7. Actualización de Intervalos de Comparación de Dificultad

En el editor del framework se encuentra el directorio `src/core/data`, el cual contiene el archivo `comparisionIntervals.json` con el siguiente contenido:

```
1  {
2    "cc": {
3      "1": ["-inf", -37.51],
4      "2": [-37.5, 6.25],
5      "3": [6.26, 26.26],
6      "4": [26.27, 70.03],
7      "5": [70.04, "inf"]
8    },
9    "hDifficult": {
10     "1": ["-inf", -76.66],
11     "2": [-76.65, -32.61],
12     "3": [-32.6, -12.6],
13     "4": [-12.59, 31.46],
14     "5": [31.47, "inf"]
15   },
16   "hEffort": {
17     "1": ["-inf", -54.07],
18     "2": [-54.06, 3.87],
19     "3": [3.88, 23.88],
20     "4": [23.89, 81.83],
21     "5": [81.84, "inf"]
22   },
23   "hVolume": {
24     "1": ["-inf", -53.11],
25     "2": [-53.1, -45.62],
26     "3": [-45.61, -25.61],
27     "4": [-25.6, -18.11],
28     "5": [-18.1, "inf"]
29   }
30 }
31
```

Figura 21: Contenido del archivo de intervalos de comparación de dificultad.
Fuente: elaboración propia.

Cuya estructura consta de:

- **Métrica:** El nombre de la métrica que contendrá los intervalos de comparación de dificultad.
 - “1”: Lista con el límite inferior y superior del intervalo de comparación para la dificultad “mucho más fácil”.
 - “2”: Lista con el límite inferior y superior del intervalo de comparación para la dificultad “más fácil”.
 - “3”: Lista con el límite inferior y superior del intervalo de comparación para la dificultad “similar”.
 - “4”: Lista con el límite inferior y superior del intervalo de comparación para la dificultad “mucho difícil”.
 - “5”: Lista con el límite inferior y superior del intervalo de comparación para la dificultad “mucho más difícil”.

Siendo los nombres de métricas válidas:

- **cc:** Complejidad ciclomática.
- **hDifficult:** Dificultad según Halstead.
- **hEffort:** Esfuerzo.
- **hVolume:** Volumen

Para modificar los intervalos de comparación de dificultad, basta con modificar los valores de los límites superiores e inferiores de las métricas.

3.3.8. Análisis de Métricas entre Semestres

Para analizar cómo han ido variando las métricas a lo largo de los semestres, el framework cuenta con la sección *Métricas Históricas*, donde se pueden ver gráficos que comparan las métricas de complejidad ciclomática, dificultad, tiempo, esfuerzo y volumen a lo largo de los semestres.

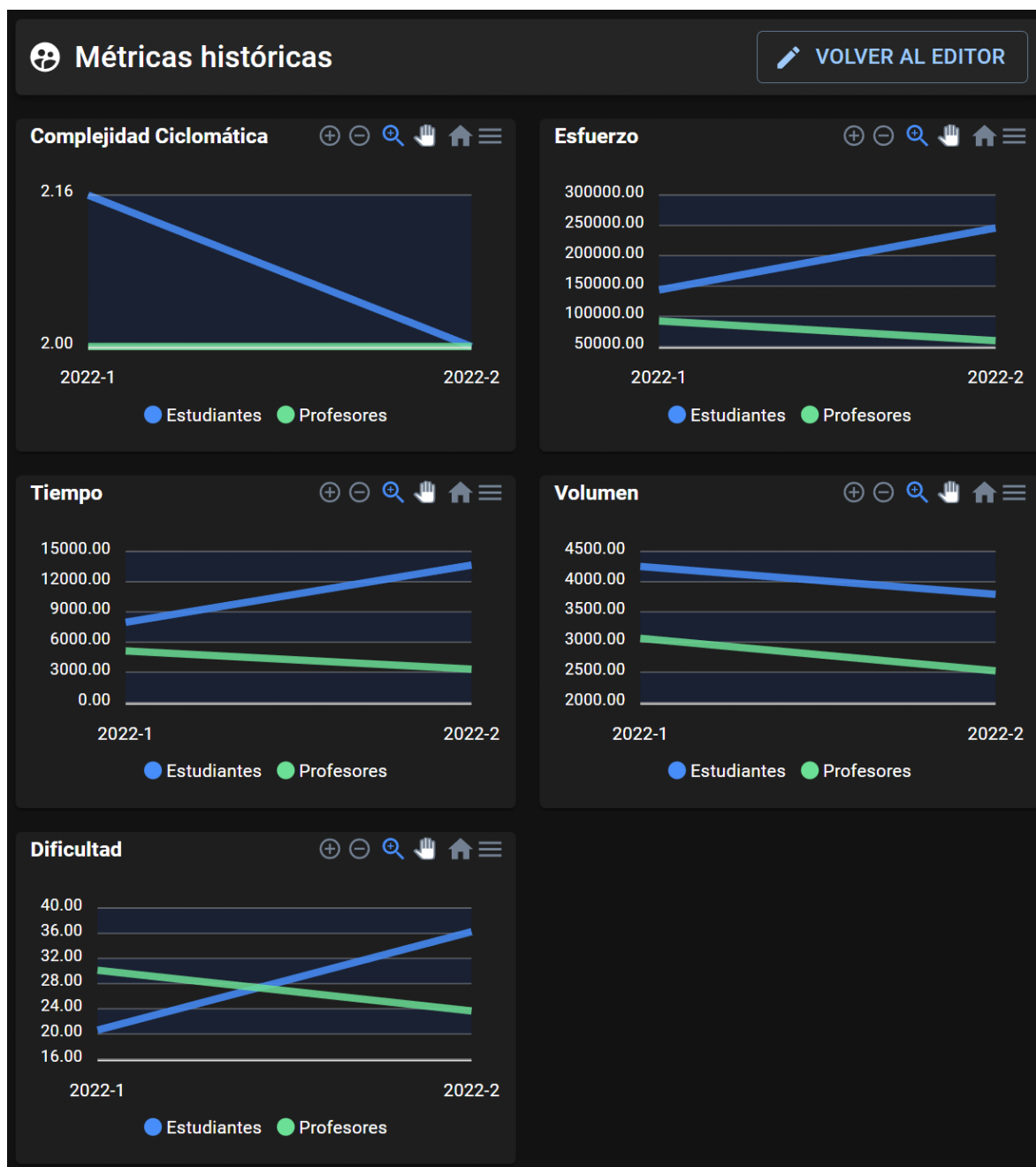


Figura 22: Sección de métricas históricas del framework Ishvel.
Fuente: elaboración propia.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

4.1. Validación del Framework

4.1.1. Cálculo de Métricas para Tareas del S2022-1 y S2022-2

Con el fin de recopilar las métricas de las soluciones entregadas a las tareas del S2022-1 y S2022-2, se recopilaron las soluciones desarrolladas por los profesores, así como el conglomerado de soluciones de los estudiantes para cada una de las tareas.

Luego, utilizando la herramienta *metrics-research*, se calcularon las métricas para cada una de las soluciones propuestas por los profesores. Para esto se hizo uso del parámetro `-f` dado que las métricas se calcularon de manera individual.

En cambio, para los estudiantes, se agruparon las soluciones en carpetas por tarea y semestre, de modo tal de ejecutar la herramienta *metrics-research* con el parámetro `-d`, para así obtener el promedio de las métricas de todas las soluciones correspondientes a una tarea.

4.1.2. Cálculo de los Intervalos de Dificultad

Utilizando la herramienta Google FormsTM, se creó un formulario con el cual evaluar la diferencia de dificultad entre las tareas del S2022-1 y el S2022-2. El formulario constaba de una descripción de lo que hay que hacer, una entrada para ingresar un correo de contacto, una pregunta para determinar el grado de expertíz del encuestado, y en adelante, una pregunta de comparación de dificultad por cada uno de los contenidos evaluados en ambos semestres:

- Tarea 1: Programas secuenciales
- Tarea 2: Condicionales
- Tarea 3: Ciclos
- Tarea 4: Funciones
- Tarea 5: Strings
- Tarea 6: Listas
- Tarea 7: Diccionesarios
- Tarea 8: Archivos

IRR Tareas de Programación

El siguiente formulario busca determinar la diferencia de dificultad entre tareas del primer y segundo semestre del 2022, del curso IWI-131 Programación de la UTFSM. Para responder, se debe hacer un juicio de valor en cuanto a qué tan difícil es una tarea en comparación a otra. La idea es comparar cada tarea del 2do semestre (2022-2) con su tarea respectiva del semestre anterior (2022-1), usando la escala explicada a continuación:

- 1 - La tarea del 2do semestre es mucho más fácil que la tarea del primer semestre del mismo contenido
- 2 - La tarea del 2do semestre es más fácil que la tarea del primer semestre del mismo contenido
- 3 - La tarea del 2do semestre tiene una dificultad similar a la del primer semestre del mismo contenido
- 4 - La tarea del 2do semestre es más difícil que la tarea del primer semestre del mismo contenido
- 5 - La tarea del 2do semestre es mucho más difícil que la tarea del primer semestre del mismo contenido

Las tareas se encuentran en este link: https://drive.google.com/file/d/1P-_cZ4Cj8q9v-B2aF5X6BPD9ZpiFyRSk/view?usp=share_link

gonzalo.fdez06@gmail.com [Cambiar de cuenta](#)

* Indica que la pregunta es obligatoria

Correo *

Tu dirección de correo electrónico

¿Cual es tu rol de cara al ramo de Programación de la UTFSM? *

- ☐ Persona que aprobó el ramo
- ☐ Estudiante cursando el ramo
- ☐ Ayudante (o ex-ayudante) del ramo
- ☐ Docente del ramo
- ☐ Otro: _____

Figura 23: Presentación del formulario para la diferencia de dificultad entre tareas.
Fuente: elaboración propia.

Tarea 2: Programas secuenciales *

1 2 3 4 5

mucho más fácil ☐ ☐ ☐ ☐ ☐ mucho más difícil

Tarea 3: Condicionales *

1 2 3 4 5

mucho más fácil ☐ ☐ ☐ ☐ ☐ mucho más difícil

Figura 24: Ejemplo de las preguntas para comparar la dificultad entre 2 tareas de distintos semestres.

Fuente: elaboración propia.

La encuesta fue respondida por 2 profesores con años de experiencia en el ramo IWI-131 de la UTFSM como docentes y como coordinadores, y por 3 ayudantes con años de experiencia apoyando el proceso docente del mismo. A continuación se resumen las respuestas:

Rol	Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5	Tarea 6	Tarea 7	Tarea 8
Ayudante	1	4	2	2	3	4	1	2
Ayudante	1	3	2	3	3	3	2	3
Ayudante	2	4	4	3	2	4	2	2
Docente	2	3	2	3	3	3	3	3
Docente	2	3	2	2	2	2	3	2
Promedio	2	3	2	3	3	3	2	2

Tabla 10: Resultado de la encuesta a expertos sobre la diferencia de dificultad entre tareas.

Fuente: elaboración propia

Luego, se calcula la diferencia porcentual entre las métricas de cada una de las tareas utilizando los datos calculados en 4.1.1, obteniendo los siguientes resultados:

Tareas	Comparación de dificultad	Complejidad ciclométrica (dif %)	Dificultad H. (dif %)	Esfuerzo (dif %)	Volumen (dif %)
Secuenciales	2 - más fácil	0,00 %	-21,46 %	-35,27 %	-17,58 %
Condicionales	3 - similar	26,32 %	12,12 %	3,46 %	-7,72 %
Ciclos	2 - más fácil	-18,75 %	-54,06 %	-76,65 %	-49,17 %
Funciones	3 - similar	50,00 %	79,59 %	-2,34 %	-45,62 %
Strings	3 - similar	40,00 %	3,87 %	-15,12 %	-18,28 %
Listas	3 - similar	54,55 %	66,39 %	515,61 %	269,97 %
Archivos	2 - más fácil	-37,50 %	-2,86 %	-32,65 %	-30,67 %
Diccionarios	2 - más fácil	6,25 %	43,69 %	-32,61 %	-53,10 %

Tabla 11: Comparación de dificultad entre tareas, junto con las diferencias porcentuales de cada una de sus métricas

Fuente: elaboración propia

Finalmente, aplicando la heurística de 4.1.1, se obtuvieron los siguientes intervalos de comparación de dificultad:

Comparación de Dificultad	Complejidad Ciclométrica	Dificultad H.	Esfuerzo	Volumen
1 - mucho más fácil	-inf %	-inf %	-inf %	-inf %
	-37,51 %	-54,07 %	-76,66 %	-53,11 %
2 - más fácil	-37,50 %	-54,06 %	-76,65 %	-53,10 %
	6,25 %	3,87 %	-32,61 %	-45,62 %
3 - similar	6,26 %	3,88 %	-32,60 %	-45,61 %
	26,26 %	23,88 %	-12,60 %	-25,61 %
4 - más difícil	26,27 %	23,89 %	-12,59 %	-25,60 %
	70,03 %	81,83 %	31,46 %	-18,11 %
5 - mucho más difícil	70,04 %	81,84 %	31,47 %	-18,10 %
	inf %	inf %	inf %	inf %

Tabla 12: Intervalos de dificultad por métrica para cada dificultad comparativa.

Fuente: elaboración propia

4.1.3. Diferencia de Dificultad entre Tareas del S2022-1 y S2022-2

Utilizando la tabla 12, se aplican estos intervalos a las diferencias porcentuales de la tabla 11, obteniendo los siguientes resultados por métrica para cada tarea:

Tareas	Complejidad ciclomática	Dificultad H.	Esfuerzo	Volumen	Promedio
Secuenciales	2	4	2	2	3
Condicionales	4	2	2	3	3
Ciclos	2	2	2	2	2
Funciones	2	2	2	5	3
Strings	4	4	5	5	5
Listas	4	4	3	2	3
Archivos	4	2	3	4	3
Diccionarios	4	3	4	5	4

Tabla 13: Dificultad de las tareas según los intervalos de comparación
Fuente: elaboración propia

Al comparar los resultados con las respuestas de la tabla 10, se determina que la heurística aplicada para determinar los intervalos de diferencias porcentuales, coincidió con la opinión de los expertos en 3 pares de tareas: Ciclos, Funciones y Strings, mientras que para el resto, los resultados no se alejaron demasiado de los resultados de la encuesta.

4.1.4. Actualización de Intervalos de Comparación de Dificultad

Para cargar los intervalos de dificultad previamente calculados en el framework, se modifica el archivo `comparisionIntervals.json` según la sección 3.3.7, resultando:

```
editor > src > core > data > {} comparisionIntervals.json > ...
1  {
2    "cc": {
3      "1": ["-inf", -37.51],
4      "2": [-37.5, 6.25],
5      "3": [6.26, 26.26],
6      "4": [26.27, 70.03],
7      "5": [70.04, "inf"]
8    },
9    "hDifficult": {
10     "1": ["-inf", -76.66],
11     "2": [-76.65, -32.61],
12     "3": [-32.6, -12.6],
13     "4": [-12.59, 31.46],
14     "5": [31.47, "inf"]
15   },
16   "hEffort": {
17     "1": ["-inf", -54.07],
18     "2": [-54.06, 3.87],
19     "3": [3.88, 23.88],
20     "4": [23.89, 81.83],
21     "5": [81.84, "inf"]
22   },
23   "hVolume": {
24     "1": ["-inf", -53.11],
25     "2": [-53.1, -45.62],
26     "3": [-45.61, -25.61],
27     "4": [-25.6, -18.11],
28     "5": [-18.1, "inf"]
29   }
30 }
31
```

Figura 25: Intervalos de comparación de dificultad cargados en el framework.
Fuente: elaboración propia.

4.1.5. Actualización de Métricas S2022-1 y S2022-2

Para cargar las métricas previamente calculadas en el framework, se modifican los archivos `studentAssignments.json` y `teacherAssignments.json` según lo que indica la sección 3.3.6, resultando:

```
editor > src > core > data > {} studentAssignments.json > {} 0 > {} strings
1  ✓ [
2  ✓ {
3    "semester": "2022-1",
4    > "dicts": { ...
10   },
11   > "files": { ...
17   },
18   > "lists": { ...
24   },
25   > "functions": { ...
31   },
32   > "strings": [ ...
38   ],
39   > "loops": { ...
45   },
46   > "conditionals": { ...
52   },
53   > "sequential": { ...
59   }
60 },
61 ✓ {
62   "semester": "2022-2",
63   > "dicts": { ...
69   },
70   > "files": { ...
76   },
77   > "lists": { ...
83   },
84   > "functions": { ...
90   },
91   > "strings": { ...
97   },
98   > "loops": { ...
104  },
105  > "conditionals": { ...
111  },
112  ✓ "sequential": {
113    "cyclomatic_complexity": 2.0,
114    "halstead_difficulty": 36.24925768120358,
115    "halstead_effort": 245707.91024898444,
116    "halstead_timerequired": 13650.439458276913,
117    "halstead_volume": 3790.8910712734883
118  }
119 }
120 ]
```

Figura 26: Métricas por semestre de las soluciones de los estudiantes cargados en el framework.

(algunas métricas fueron ocultas para efectos de espacio) Fuente: elaboración propia.

```
editor > src > core > data > {} teacherAssignments.json > ...
1  [
2    {
3      "semester": "2022-1",
4      "dicts": { ...
10    },
11    "files": { ...
17    },
18    "lists": { ...
24    },
25    "functions": { ...
31    },
32    "strings": { ...
38    },
39    "loops": { ...
45    },
46    "conditionals": { ...
52    },
53    "sequential": { ...
59    }
60  },
61  {
62    "semester": "2022-2",
63    "dicts": { ...
69    },
70    "files": { ...
76    },
77    "lists": { ...
83    },
84    "functions": { ...
90    },
91    "strings": { ...
97    },
98    "loops": { ...
104    },
105    "conditionals": { ...
111    },
112    "sequential": {
113      "cyclomatic_complexity": 2,
114      "halstead_difficulty": 23.653846153846153,
115      "halstead_effort": 59711.84138271026,
116      "halstead_timerequired": 3317.324521261681,
117      "halstead_volume": 2524.4030503259623
118    }
119  }
120 ]
121
```

Figura 27: Métricas por semestre de las soluciones de los profesores cargados en el framework.

(algunas métricas fueron ocultas para efectos de espacio) Fuente: elaboración propia.

4.1.6. Redacción de una Tarea con sus Respectivas Métricas

Siguiendo los pasos de la metodología, se abre el editor de ishvel desde <https://vadokdev.github.io/Ishvel/> y se configuran las opciones de comparación. Para este caso, se utilizarán las tareas de los estudiantes para el contenido de Strings del semestre 2022-2.

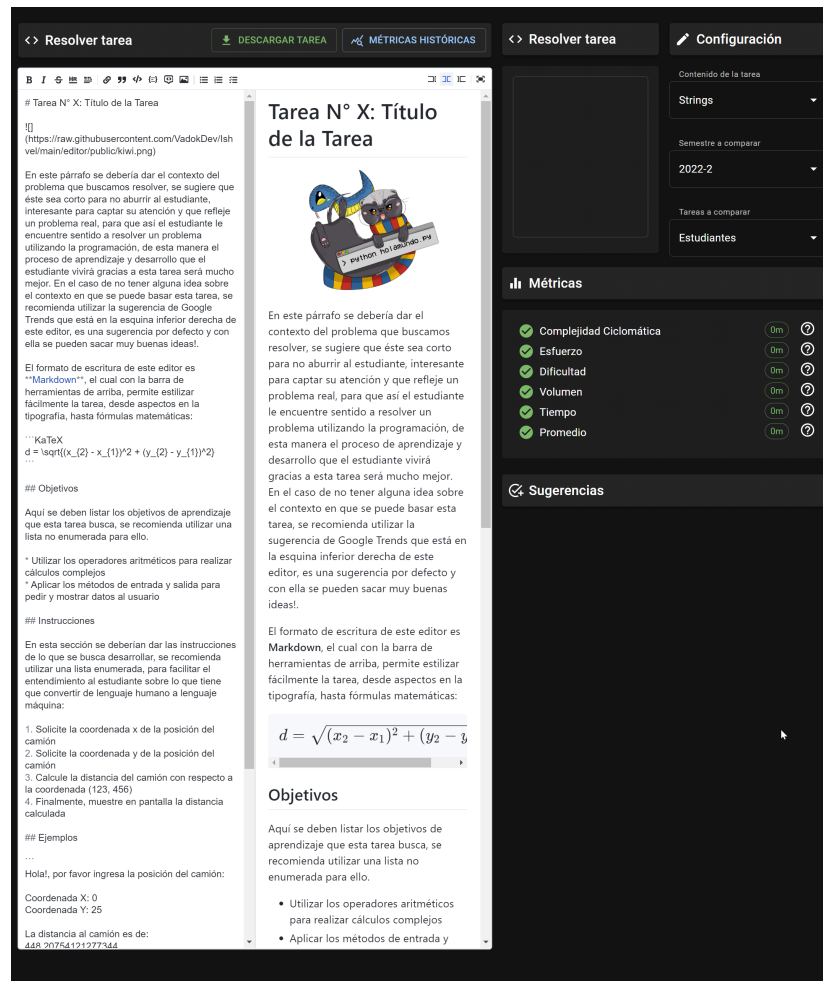


Figura 28: Editor de Ishvel.
Fuente: elaboración propia.

Luego, se utilizan las herramientas del editor de Ishvel para redactar la tarea, añadiendo imágenes, fórmulas, y completando el contexto, instrucciones, ejemplos y recomendaciones de la tarea, así como cualquier otro aspecto que se considere relevante para el estudiante.

Fuente: elaboración propia.

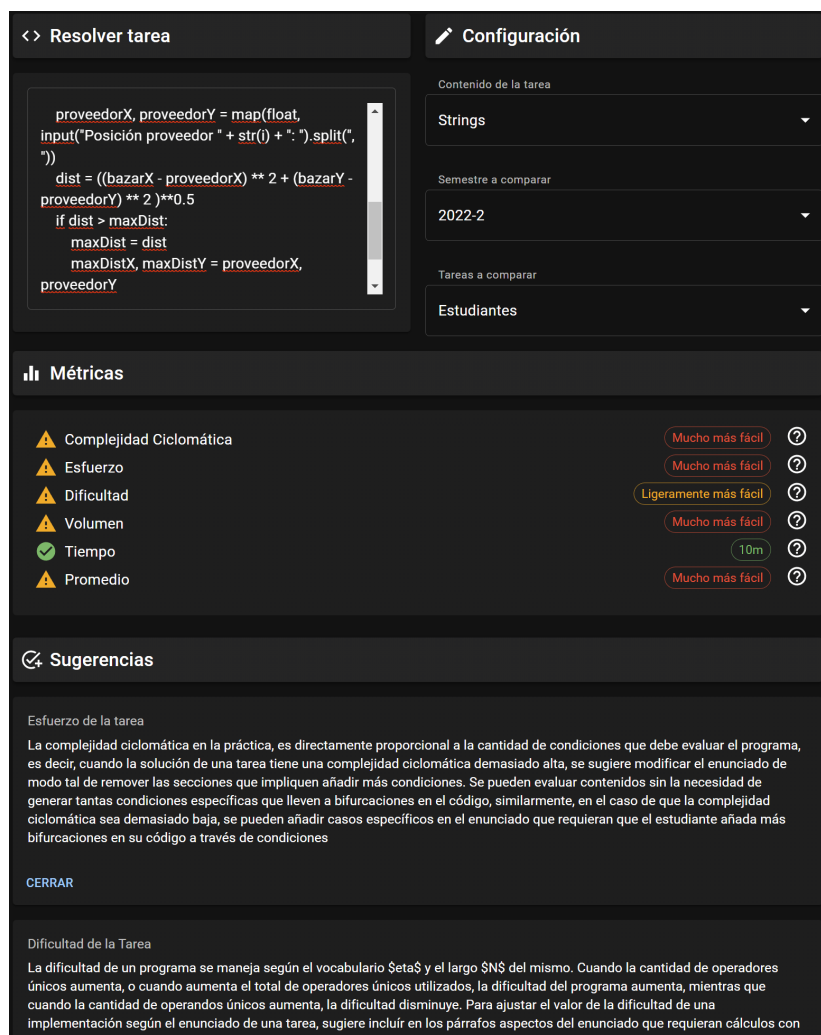


Figura 30: Métricas y sugerencias para una solución en base a una tarea redactada en el editor de Ishvel.

Fuente: elaboración propia.

Según el editor, se presume entonces que la tarea elaborada es mucho más fácil que la tarea de strings del S2022-2, en base a las soluciones de los estudiantes. Además, el editor presenta sugerencias que se pueden aplicar para abordar este resultado.

Por otro lado, el editor estima que la solución a esta tarea se podría desarrollar en 10 minutos, lo cual es bueno dado que se busca que las tareas no requieran más de una hora para ser resueltas por los estudiantes. Luego de utilizar estos consejos, se logra llegar a una tarea cuya dificultad en base a las métricas es en promedio similar a la de la tarea del S2022-2, por lo que con esto se tiene una tarea cuya dificultad no escapa a la de semestres anteriores. Sin embargo, ésta practicamente duplica el tiempo requerido para desarrollarse en comparación a lo esperado, pero en términos de dificultad la tarea ya es aceptable.

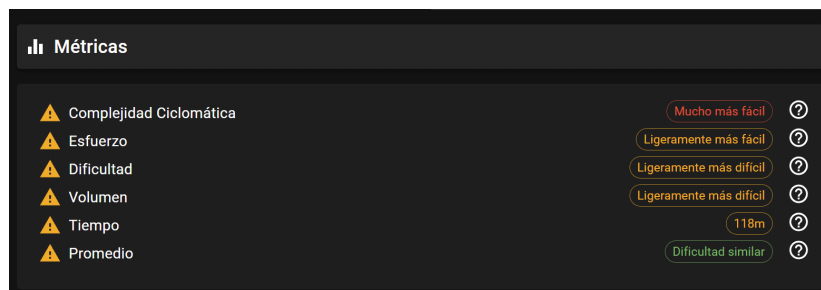


Figura 31: Métricas de una solución en base a una tarea redactada en el editor de Ishvel.
Fuente: elaboración propia.

4.1.7. Comparación de las Tareas de Listas de IWI-131 del S2022-1 y S2022-2

Utilizando el editor del framework Ishvel, se visita la sección de métricas históricas y se configura el editor para observar las métricas de la tarea de listas, siendo estos los resultados obtenidos por métrica:



Figura 32: Métricas históricas del S2022-1 y S2022-2 para la tarea de listas en el editor.
Fuente: elaboración propia.

Al analizar en detalle la situación, es posible notar que en la tarea de listas del S2022-1, las soluciones de los estudiantes se acercaron bastante a las soluciones propuestas por los profesores. Entre los distintos aspectos que pudieron llevar a este resultado, se destacan los siguientes:

- La tarea del S2022-1 constaba de 2 partes, una donde se realizaba la contextualización del problema y luego se resolvía un cuestionario en AULA, y otra en la que se debía programar. Esta introducción previa al problema debió ayudar a los estudiantes a tener un buen entendimiento de lo que se necesitaba hacer, así como también, familiarizarse con el contexto del problema el cual representaba una situación cercana a la realidad, que además es interesante, lo que cumple con los requisitos de una buena tarea planteados al inicio de este documento.
- El formato de la tarea del S2022-1 es bastante cercano al propuesto por la metodología de este framework, es decir, el hecho de comenzar con una contextualización interesante y útil, seguir con una serie de instrucciones en una lista ordenada, entregar ejemplos y una lista de recomendaciones, ayudó a que los estudiantes tuviesen muy claro qué tenían que hacer y cómo llevarlo a cabo. Especialmente en la parte de la complejidad ciclomática, se puede notar que la cantidad de bifurcaciones en el código que los estudiantes utilizaron para desarrollar sus tareas, es prácticamente igual a la de los profesores. Esto significa que estaba claro qué casos había que evaluar en base a lo que el enunciado entregaba.
- Los contenidos evaluados se limitaban a lo enseñado en la UVA, y al separar el desarrollo de la tarea del S2022-1 en 2 semanas de aprendizaje continuo sobre listas, le permitieron al estudiante mantener los contenidos frescos para el posterior desarrollo de su tarea.
- El tiempo estimado requerido para desarrollar esta tarea según la solución de los profesores se encontraba por debajo de una hora, lo que significa que era posible saber de antemano que la tarea podía resolverse en ese plazo. Además, las soluciones de los estudiantes reflejan esto, dado que sólo están aproximadamente 7 minutos por sobre el tiempo de resolución de los profesores. El hecho de elaborar tareas que requieran menos de una hora para su resolución, apoya enormemente a los estudiantes con la gestión de sus horarios académicos y les facilita llevar a cabo sus responsabilidades con otros ramos dado que se respeta la exigencia horaria de IWI-131.

En el caso de la tarea de listas del S2022-2, las métricas de las soluciones de los estudiantes difieren significativamente de las de los profesores en comparación con las métricas de la tarea del S2022-1, lo cual puede deberse a diversos factores:

- La tarea del S2022-2 tiene un tiempo estimado de resolución de cinco horas, lo cual de antemano entrega indicios de que el tiempo que los estudiantes debieron dedicarle al

desarrollo de su solución, fue más que el esperado y por tanto el añadir la complejidad de gestionar sus horarios académicos y el tiempo que consumían otros ramos, la situación en la que desarrollaron la tarea pudo no ser la más adecuada.

- A excepción de la métrica de complejidad ciclomática, todas las métricas promedio de las soluciones de los estudiantes están considerablemente por debajo de las métricas de las soluciones de los profesores. El hecho de que la complejidad ciclomática estuviese en promedio por sobre la de los profesores, indica que la forma en que el enunciado entregaba a los estudiantes las instrucciones, no fue suficientemente claro como para que éstos tuviesen claro los casos a evaluar. A diferencia de la tarea del S2022-1 que seguía en gran parte el formato de la metodología de Ishvel, la tarea del S2022-2 no lo sigue y como resultado los estudiantes desarrollaron programas que evaluaban más casos bordes que los que el enunciado solicitaba en realidad.
- Según el análisis de dificultad en base a métricas del editor de Ishvel, la tarea del S2022-2 es **mucho más difícil** que la tarea del S2022-1.

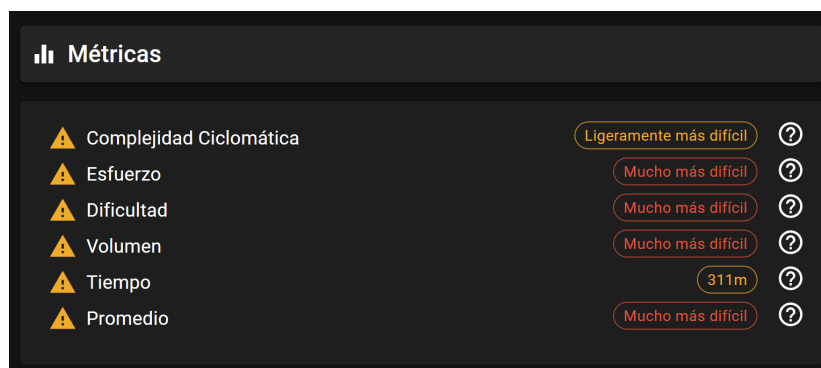


Figura 33: Comparación de dificultad de la solución de la tarea de listas del S2022-2 en el editor de Ishvel

Fuente: elaboración propia.

4.1.8. Comparación de las Tareas de Archivos de IWI-131 del S2022-1 y S2022-2

Utilizando el editor del framework Ishvel, se visita la sección de métricas históricas y se configura el editor para observar las métricas de la tarea de archivos, siendo estos los resultados por métrica:



Figura 34: Comparación de dificultad de la solución de la tarea de archivos del S2022-2 en el editor de Ishvel

Fuente: elaboración propia.

Para el caso de la tarea de archivos en ambos semestres, ocurre un fenómeno poco usual en el S2022-1: las soluciones de los estudiantes se acercan significativamente a las soluciones entregadas por los profesores en casi todas las métricas, excepto en la métrica de complejidad ciclomática. Hay diversos factores que impactan en esta situación:

- En la solución propuesta de los profesores, se hace uso de funciones utilitarias, las cuales aumentan la complejidad ciclomática del programa en sí dado que son bloques de código independiente. Sin embargo, los estudiantes en general no hicieron uso de funciones, dado que no fueron solicitadas explícitamente en el enunciado a diferencia de la tarea de archivos del S2022-2, y por lo tanto resolvieron sus programas sin hacer

reutilización de código mediante funciones, lo que explica por qué el volumen de sus soluciones es mayor al volumen de la solución de los profesores.

- Dado que la tarea de archivos es de las últimas tareas del semestre, existe un incentivo menor a desarrollarla dado que su impacto en la nota final puede no ser demasiado relevante para los alumnos. En particular, en el S2022-1 de 1341 estudiantes, 784 no hicieron entrega de su tarea de archivos, mientras que en el S2022-2 de un total de 1149 estudiantes, 696 no hicieron entrega de esta tarea. Esta falta de incentivo lleva también a algunos estudiantes a enviar un programa incompleto o que saben que no funcionará correctamente, con un vocabulario de implementación menor que hace que la dificultad de sus soluciones sea menor a la solución de los profesores en el S2022-1.
- El enunciado de la tarea del S2022-1 sigue de forma similar el formato planteado por la metodología del framework. Además, es corto dado que la mayoría de su contenido son ejemplos, y muy conciso a la hora de explicar todas las condiciones que se tienen que trabajar, por lo que no hay que inferir condiciones del texto, lo que facilita el entendimiento de lo que hay que hacer para los estudiantes, y apoya el hecho de que éstos generaran programas similares al que los profesores entregaron como solución de la tarea.

Por otro lado, para el caso de la tarea de archivos del S2022-2, se puede apreciar que pese a que las métricas de la solución de los profesores, son similares a las de la solución de los profesores del S2022-1, éstas difieren significativamente de las métricas de las soluciones entregadas por los estudiantes. Además, que la complejidad ciclomática promedio de las soluciones de los estudiantes, es mucho mayor a la complejidad ciclomática de la tarea de los profesores. Entre los factores que pueden explicar esta situación, además del hecho de que la tarea de archivos es de las últimas del semestre, se encuentran:

- El enunciado de la tarea requiere inferir ciertas condiciones que se deben implementar en el código, en particular, con respecto a cómo extraer información de un archivo cuyo formato no es constante. La carga cognitiva añadida de tener que descifrar exactamente cómo llevar a cabo esto, lleva a los estudiantes a añadir muchas condiciones en su código para poder resolver esta problemática, las cuales podrían llevarse a cabo de una manera más estructurada si las instrucciones dijeran explícitamente qué es lo que se debe traducir de lenguaje humano a código.
- En las exigencias de la tarea se encuentran no sólo el manejo de archivos, sino también la elaboración de diccionarios con cierta estructura en base a la lectura de un archivo. Al mezclar estos 2 contenidos de esta forma, se aglomeran los contenidos que los estudiantes deben utilizar, llevándoles a elaborar programas con un mayor volumen, lo que lleva a una mayor dificultad y un mayor esfuerzo en las métricas de sus soluciones. Se entiende que el objetivo de las UVAs en el ramo es que sea un aprendizaje incremental, y que cada UVA puede hacer uso de los contenidos de la UVA anterior,

sin embargo, en este caso lo que más complicó a los estudiantes fue el desarrollo de la función `procesar_planilla` solicitada, pues la mayoría del código se concentra aquí, para lograr generar el diccionario con la estructura deseada. De haber concentrado la tarea en los contenidos de archivos, la métrica de volumen disminuiría y consigo disminuirían todas las demás, acercándose a la solución propuesta por los profesores.

En ambos casos las soluciones de los profesores tienen un tiempo estimado de solución mayor a una hora, lo cual al impactar negativamente en el manejo del horario académico de los estudiantes, refuerza también la idea de que entreguen tareas a medio hacer o no entreguen, y que las métricas de sus soluciones difieran mucho más a las métricas de las soluciones de los profesores.

CAPÍTULO 5

CONCLUSIONES

El curso de programación IWI131 de la Universidad Técnica Federico Santa María está muy bien estructurado y hace un gran trabajo con la enseñanza a través de UVAs. Sin embargo, luego del estudio llevado a cabo en este documento, es posible concluir que hay muchas mejoras que se pueden llevar a cabo en un instrumento formativo y evaluador tan importante como son las tareas. Las recomendaciones y párrafos respecto a qué es una buena tarea planteadas en este documento, servirán como una guía para la elaboración de tareas de programación a futuro, pues se trabajaron distintos aspectos desde qué genera interés en los estudiantes, hasta qué causa frustración en ellos a la hora de desarrollar sus tareas.

Por otro lado, la aplicación de métricas de software como herramienta de medición para la dificultad de una tarea, y para el estudio de la experiencia de los estudiantes a la hora de desarrollarla, fue de gran ayuda para poder entender ciertos aspectos en los que debemos prestar más atención a la hora de elaborar una tarea, y cómo la solución de la misma nos entrega información importante al respecto.

Si bien las métricas de Halstead datan de hace más de 50 años, aún así es posible extraer información valiosa de ellas, especialmente cuando son aplicadas a las soluciones de una tarea de programación. Este estudio demuestra el potencial que tienen para analizar la dificultad de una tarea antes de ser publicada. Sin embargo, se requiere una mayor cantidad de datos respecto a más semestres de tareas y el cálculo de sus métricas, para poder así extraer mejores conclusiones. Del mismo modo, la métrica de complejidad ciclomática nos da una interesante aproximación respecto a cómo se traduce el enunciado de una tarea a la cantidad de condiciones que el estudiante tendrá que programar, y cómo éstas complicarán el proceso de desarrollo de su solución impactando en otras métricas.

El editor de Ishvel es una gran herramienta para aplicar la metodología del framework. Entrega de antemano una tarea de ejemplo con la cual es posible iniciar la elaboración de una tarea para cualquier contenido. Sin embargo, la información que puede aportar con respecto a la dificultad de una tarea en comparación a tareas anteriores, está limitada por la eficacia de la heurística planteada en este documento para determinar los intervalos de comparación de dificultad. Dado que ésta no tiene suficiente afinación todavía, se requiere de mayor trabajo en ella para lograr una heurística más eficaz.

Del mismo modo, el uso de tecnologías web que permitan ejecutar el editor de Ishvel en cualquier navegador, sin la necesidad de un servidor externo el cual se necesite mantener, o una base de datos externa en la cual cargar la data de las métricas de cada semestre, facilita mucho la distribución y el uso del framework a gran escala, pues todo queda en manos de Github Pages TM. Por esta misma razón, es que el framework quedó de código abierto, de manera tal que cualquier profesor en cualquier universidad puede descargarlo, añadir las métricas de los contenidos de sus propias tareas, y hacer un análisis sobre el desarrollo de

su curso de programación a lo largo del semestre en su propia versión del editor. Con una arquitectura así, se podrá distribuir el framework mucho más fácilmente y no sólo limitarlo a cursos de la Universidad Técnica Federico Santa María.

Por lo demás, en base a la validación del framework, es posible determinar que fue posible cumplir con el objetivo de crear un framework capaz de apoyar en la elaboración de tareas para cursos introductorios de programación, en base a una metodología y la definición de métricas que entreguen información respecto a la dificultad de las mismas. Con estas métricas, se pueden evaluar las soluciones de las tareas y así estudiar el efecto que éstos tienen a medida que varían en las mismas, como se plantea en la sección de validación.

5.1. Trabajo a Futuro

A futuro es necesario determinar una mejor heurística para la creación de intervalos de comparación de dificultad, ya que la propuesta en este documento no está lo suficientemente afinada, y requiere de mayor información. Del mismo modo, se debe continuar analizando las tareas de los próximos semestres en el curso de programación, para así acumular más material de estudio, más métricas con las cuales poder hacer un análisis estadístico más concreto, y más respuestas sobre la comparación de dificultad entre tareas por un conjunto mayor de expertos que el que participó de este trabajo. Con esta información, se puede determinar cómo es la distribución de las métricas de las tareas y relacionar los intervalos de confianza de la misma con los intervalos de comparación de dificultad, lo que daría un respaldo estadístico mayor a las mismas, y afinaría mucho más la metodología del framework para el análisis de dificultad de las tareas.

También es posible hacer un análisis de experiencia de usuario con respecto al editor, probarlo en un laboratorio y obtener datos sobre la experiencia de un profesor elaborando una tarea en él, para así corregir aspectos de la interfaz, ver qué otras opciones podrían añadirse al editor, y hacer mejoras en la metodología. Junto con esto, se puede disminuir la carga cognitiva entregada por los valores de tantas métricas en el sitio mediante la creación de una única métrica, que resulte de alguna ponderación realizada en base a las ya expuestas en este documento, y que entregue un valor final con el cual categorizar la tarea en cuestión. Para crear esta métrica que pondere las demás, se deja como trabajo a futuro la experimentación de cómo distintos valores de las métricas en distintas soluciones de tareas, impactan en la visión del estudiante hacia las mismas, y así recabar información mediante encuestas para poder agrupar los valores de las métricas, ponderarlos y finalmente extraer una única métrica de evaluación para las tareas.

REFERENCIAS BIBLIOGRÁFICAS

- [Biggs y Tang, 2011] Biggs, J. y Tang, C. (2011). *Teaching For Quality Learning At University*. SRHE and Open University Press Imprint. McGraw-Hill Education.
- [Boye, 2020] Boye, A. (2020). How do i create meaningful and effective assignments? https://www.depts.ttu.edu/tlpdc/Resources/Teaching_resources/TLPDC_teaching_resources/CreatingEffectiveAssignments.php. [Online; consultado el 19 de julio del 2021 a las 23:00hrs].
- [Craig y Morrison, 2021] Craig, M. y Morrison, B. B. (2021). Engagecse-
du

assignments: To re-use or not re-use? that is the question. *ACM Inroads*, 12(3):18–20.
- [Cunningham, 2014] Cunningham, W. (2014). Problem domain.
- [Derntl y Calvo, 2011] Derntl, M. y Calvo, R. (2011). E-learning frameworks: Facilitating the implementation of educational design patterns. *International Journal of Technology Enhanced Learning*, 3:284–296.
- [Ebert et al., 2016] Ebert, C., Cain, J., Antoniol, G., Counsell, S., y Laplante, P. (2016). Cyclo-matic complexity. *IEEE Software*, 33(06):27–29.
- [Fayad et al., 1999] Fayad, M. E., Schmidt, D. C., y Johnson, R. E. (1999). *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. John Wiley & Sons, Inc., USA.
- [Gackenhaimer, 2015] Gackenhaimer, C. (2015). *What Is React?*, pp. 1–20. Apress, Berkeley, CA.
- [Gruber, 2021] Gruber, J. (Última actualización en 2021). Markdown. <https://daringfireball.net/projects/markdown/>. Accedido el 1 de mayo de 2023.
- [Halstead, 1977] Halstead, M. H. (1977). *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., USA.
- [Hamer y Frewin, 1982] Hamer, P. G. y Frewin, G. D. (1982). M.h. halstead's software science - a critical examination. En *Proceedings of the 6th International Conference on Software Engineering*, ICSE '82, Washington, DC, USA. IEEE Computer Society Press.
- [Hertz, 2010] Hertz, M. (2010). What do ς_1 and ς_2 mean? investigating differences in the early courses. En *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, p. 199–203, New York, NY, USA. Association for Computing Machinery.
- [Huffman, 2023] Huffman, R. M. (2023). *Julia in WebAssembly*. Tesis doctoral, Massachusetts Institute of Technology.

- [Hundhausen *et al.*, 2015] Hundhausen, C. D., Carter, A. S., y Adesope, O. (2015). Supporting programming assignments with activity streams: An empirical study. En *Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15*, pp. 320–325, New York, NY, USA. Association for Computing Machinery.
- [Kinnunen y Simon, 2010] Kinnunen, P. y Simon, B. (2010). Experiencing programming assignments in cs1: The emotional toll. En *Proceedings of the Sixth International Workshop on Computing Education Research, ICER 10*, p. 77–86, New York, NY, USA. Association for Computing Machinery.
- [Konrad Weihmann, 2023] Konrad Weihmann, G. F. (2023). *multimetricprog*. <https://pypi.org/project/multimetricprog/>. Accedido el 1 de mayo del 2023.
- [La Red Martínez *et al.*, 2012] La Red Martínez, D. L., Acosta, J. C., Mata, L. E., Bachmann, N. G., y Vallejos, O. (2012). Aprendizaje combinado, aprendizaje electrónico centrado en el alumno y nuevas tecnologías. En *VII Congreso de Tecnología en Educación y Educación en Tecnología*.
- [Layman *et al.*, 2007] Layman, L., Williams, L., y Slaten, K. (2007). Note to self: Make assignments meaningful. *SIGCSE Bull.*, 39(1):459–463.
- [Li y Cheung, 1987] Li, H. y Cheung, W. (1987). An empirical study of software metrics. *IEEE Transactions on Software Engineering*, SE-13(6):697–708.
- [Lionelle *et al.*, 2020] Lionelle, A., Grinslad, J., y Beveridge, J. R. (2020). Cs 0: Culture and coding. En *Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE '20*, pp. 227–233, New York, NY, USA. Association for Computing Machinery.
- [Mozilla Contributors, 2021] Mozilla Contributors (Última actualización en 2021). Javascript. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Accedido el 1 de mayo de 2023.
- [National Academy of Engineering, 2009] National Academy of Engineering (2009). *Developing Metrics for Assessing Engineering Instruction: What Gets Measured Is What Gets Improved*. The National Academies Press, Washington, DC.
- [Omer U, 2021] Omer U, Farooq MS, A. A. (2021). Introductory programming course: review and future implications. *PeerJ Computer Science* 7:e647.
- [Python Software Foundation, 2023] Python Software Foundation (2023). Python 3 documentation. <https://docs.python.org/3/>.
- [Siegel *et al.*, 2021] Siegel, A. A., Zarb, M., Alshaigy, B., Blanchard, J., Crick, T., Glassey, R., Hott, J. R., Latulipe, C., Riedesel, C., Senapathi, M., Simon, y Williams, D. (2021). Educational landscapes during and after covid-19. En *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2, ITiCSE '21*, pp. 597–598, New York, NY, USA. Association for Computing Machinery.

- [Simon, 2017] Simon (2017). Designing programming assignments to reduce the likelihood of cheating. En *Proceedings of the Nineteenth Australasian Computing Education Conference, ACE '17*, pp. 42–47, New York, NY, USA. Association for Computing Machinery.
- [Stevenson y Wagner, 2006] Stevenson, D. E. y Wagner, P. J. (2006). Developing real-world programming assignments for cs1. En *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITICSE '06*, pp. 158–162, New York, NY, USA. Association for Computing Machinery.
- [Stroud, 1967] Stroud, J. M. (1967). The fine structure of psychological time. *Annals of the New York Academy of Sciences*, 138(2):623–631.
- [Torrey, 2011] Torrey, L. (2011). Student interest and choice in programming assignments. *J. Comput. Sci. Coll.*, 26(6):110–116.
- [Tucker, 2012] Tucker, B. (2012). The flipped classroom. *Education next*, 12(1):82–83.
- [Utomo y Falahah, 2020] Utomo, P. y Falahah (2020). Building serverless website on github pages. *IOP Conference Series: Materials Science and Engineering*, 879(1):012077.
- [Vivian et al., 2013] Vivian, R., Falkner, K., y Falkner, N. (2013). Computer science students' causal attributions for successful and unsuccessful outcomes in programming assignments. En *Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli Calling '13*, pp. 125–134, New York, NY, USA. Association for Computing Machinery.
- [Vásquez et al., 2021] Vásquez, A., Meza, F., y Barrera, P. G. (2021). Emergency remote teaching model for massive programming classes. En *2021 XLVII Latin American Computing Conference (CLEI)*, pp. 1–9.
- [W3C, 2022] W3C (2022). Webassembly core specification. <https://www.w3.org/TR/wasm-core-2/>.
- [Watson y Li, 2014] Watson, C. y Li, F. W. (2014). Failure rates in introductory programming revisited. En *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, ITICSE '14*, pp. 39–44, New York, NY, USA. Association for Computing Machinery.
- [Weihmann, 2021] Weihmann, K. (2021). *multimetric*. <https://github.com/priv-kweihmann/multimetric>. Accedido el 1 de mayo del 2023.
- [Yernaux et al., 2020] Yernaux, G., Vanhoof, W., y Schumacher, L. (2020). Moulinog: A generator of random student assignments written in prolog. En *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming, PPDP '20*, New York, NY, USA. Association for Computing Machinery.