Juan Andrés Carruthers

Modelo de Proceso para la construcción y mantenimiento de muestras de proyectos software para estudios en la Ingeniería del Software Empírica

Tesis Doctoral en Ciencias Informáticas
PREMIO DR. RAÚL GALLARD | Año 2025

edulp
EDITORIAL DE LA UNLP

Modelo de proceso para la construcción y mantenimiento de muestras de proyectos software para estudios en la Ingeniería del Software Empírica

Juan Andrés Carruthers

UNIVERSIDAD NACIONAL DEL NORDESTE

TESIS DOCTORAL EN CIENCIAS INFORMÁTICAS

Modelo de proceso para la construcción y mantenimiento de muestras de proyectos software para estudios en la Ingeniería del Software Empírica

Juan Andrés Carruthers

Director: Emanuel Agustín Irrazábal
Co-director: Jorge Andrés Diaz-Pace

La Plata, Argentina
Junio de 2025

EDITORIAL DE LA UNLP

Carruthers, Juan Andrés
Modelo de proceso para la construcción y mantenimiento de muestras de proyectos software para estudios en la Ingeniería del Software Empírica / Juan Andrés Carruthers. - 1a ed. - La Plata: EDULP, 2026.
Libro digital, PDF/A

Archivo Digital: descarga y online
ISBN 978-631-6568-83-0

1. Ingeniería de Software. 2. Software. I. Título.
CDD 005

Modelo de proceso para la construcción y mantenimiento de muestras de proyectos software para estudios en la Ingeniería del Software Empírica

Juan Andrés Carruthers

EDITORIAL DE LA UNIVERSIDAD NACIONAL DE LA PLATA (EDULP)

48 Nº 551-599 4º Piso/ La Plata B1900AMX / Buenos Aires, Argentina
+54 221 44-7150
edulp.editorial@gmail.com
www.editorial.unlp.edu.ar

Edulp integra la Red de Editoriales de las Universidades Nacionales (REUN)

ISBN 978-631-6568-83-0

Queda hecho el depósito que marca la Ley 11.723

Impreso en Argentina

Agradecimientos

A mi querida familia, a mis padres Ronnie y Lupe, y mi hermano Pedri, por su apoyo incondicional en este camino.

A Emanuel, por la dirección de este trabajo, mentoría de mi carrera profesional e iniciación en la investigación. A Andrés, también por la dirección y colaboración en los estudios publicados. Al Grupo de Investigación de Calidad de Software de la Universidad Nacional del Nordeste, en especial a Andrea por el acompañamiento durante el doctorado.

Al CONICET, por el respaldo material proporcionado para conducir esta investigación. A la Universidad de Hamburgo, sobre todo a André Van Hoorn por la gentileza de recibirme en el grupo y la supervisión en la validación de la solución. Al Servicio de Intercambio Académico Alemán y la Secretaría de Educación de la República Argentina, por hacer posible la estancia de investigación en Alemania.

Juan Andrés Carruthers
La Plata, 2025

Resumen

Las muestras de proyectos de software son una de las principales fuentes de información utilizadas en la Ingeniería del Software Empírica y se espera que los resultados obtenidos de estas sean, hasta cierto punto, repetibles y generalizables. El desarrollo de software de fuente abierta y su difusión a través de plataformas para compartir código han facilitado el acceso a millones de proyectos, sin embargo, la falta de controles en la creación de repositorios implica que pueden contener información irrelevante para estos estudios (por ejemplo, proyectos no ingenieriles, archivos de texto, imágenes, diapositivas, etc.). También se utilizan las muestras o colecciones compartidas por la comunidad científica, aunque, en ocasiones, estas se conforman siguiendo procedimientos arbitrarios, con escasa documentación y cuestionables desde el punto de vista metodológico.

Para abordar estos problemas se propuso como solución la implementación del modelo de proceso SUM4SOFT para la construcción, actualización y curaduría de colecciones de proyectos software. Este modelo se desarrolló en el marco de una investigación doctoral planificada con la metodología ciencia de diseño, que define las actividades: explicar el problema, definir objetivo y requisitos, diseñar y desarrollar el artefacto, demostrar el artefacto, y evaluar el artefacto. En este contexto, se condujeron cinco estudios: dos mapeos sistemáticos de la literatura, un estudio longitudinal, un estudio de minería de repositorios de software y un estudio de caso.

Los mapeos sistemáticos indicaron la falta de reglas uniformes para la selección de proyectos, el acceso limitado a las colecciones y a los paquetes de replicación de la investigación o el uso escaso de técnicas de muestreo probabilístico, entre otros hallazgos. El estudio longitudinal demostró cómo una población de proyectos de software experimenta cambios significativos a lo largo del tiempo evidenciando la importancia de utilizar muestras vigentes. Posteriormente, se construyó un prototipo de SUM4SOFT, el cual se aplicó en un entorno real para obtener el instrumento de recolección de datos JavaQ. Esta herramienta integra estrategias de muestreo y actualización basadas en un enfoque aleatorio estratificado implementando variaciones del algoritmo para agrupamiento k-means. El estudio de minería de

repositorios validó la efectividad de las estrategias de muestreo implementadas en JavaQ para recolectar muestras representativas. Finalmente, el estudio de caso operacionalizó la incorporación de mejoras a SUM4SOFT y replicó los resultados obtenidos en el estudio de minería.

En esta investigación doctoral se diseñó SUM4SOFT con el fin de apoyar estudios empíricos a partir de los requisitos extraídos de la literatura. El modelo se probó en el contexto de un estudio de Ingeniería del Software y se perfeccionó por medio de un estudio de caso. La aplicación del modelo de proceso resultó en un instrumento de recolección de datos y colecciones de proyectos software que se validaron en un estudio de minería de repositorios. Así, se demostró que tanto SUM4SOFT como las herramientas implementadas constituyen soluciones viables para generar resultados replicables y generalizables en estudios de la Ingeniería del Software Empírica.

Índice

CAPITULO 1

Introducción

Este capítulo presenta la motivación de la tesis doctoral, el problema que busca resolver, su objetivo principal, el contexto de desarrollo de la investigación y la estructura del documento. Está organizado de la siguiente manera: la sección 1.1 describe la motivación de la tesis, una breve reseña del estado actual de las colecciones empleadas en estudios empíricos en la Ingeniería del Software y la propuesta para solucionar el problema planteado, la sección 1.2 establece la hipótesis de trabajo y los objetivos para cumplir con lo cometido, la sección 1.3 ofrece el contexto en el cual la tesis fue conducida, y la sección 1.4 especifica la estructura del documento.

1.1. Motivación

La Ingeniería del Software trabaja en la construcción y mantenimiento de aplicaciones software multiversión, por lo tanto, muchas de las actividades asociadas con una aplicación software implican revisiones para mejorar la funcionalidad o para corregir errores [1], lo que conduce a mejoras en la calidad. De acuerdo con Lehman [2], esto puede estudiarse desde dos puntos de vista: el proceso, y el código fuente resultante. En ambos casos es importante utilizar evidencias directamente relacionadas con el software a partir de métricas e indicadores para demostrar su calidad [3]. Este enfoque que recolecta y analiza datos obtenidos del proceso de desarrollo y mantenimiento del software para integrar evidencia científica se denomina Ingeniería del Software Empírica (ISE) [4].

La proliferación del software de fuente abierta a través de plataformas para compartir código como SourceForge, GitHub, Gitlab o Bitbucket; brinda a investigadores y profesionales acceso a millones de proyectos y, por lo tanto, datos para el desarrollo de estudios empíricos [5 - 7]. Dependiendo del tipo de estudio los datos recolectados en la muestra contendrán, por ejemplo, repositorios de código fuente [8], información

de rastreadores de defectos [9], métricas de calidad del producto [10, 11], o datos evolutivos [12], entre otros.

Dentro de la ISE, existen ciertos métodos de investigación donde se espera que los estudios conducidos produzcan resultados generalizables y replicables (por ejemplo, experimentos, encuestas, estudios de minería de repositorios de software) [13]. La generalización permite que los hallazgos obtenidos de una investigación puedan aplicarse a todos los sujetos de la población. Para ello se requiere que las muestras utilizadas sean representativas, es decir, que las propiedades de la población estudiada se vean reflejadas en la muestra [14]. Para el caso de las muestras o colecciones de proyectos, se espera que las conclusiones obtenidas se puedan extrapolar a toda la población.

En este sentido, la ACM define como replicación a los estudios que retornan resultados consistentes si se operan bajo las mismas condiciones independientemente del equipo que los conduzcan [15]. Esto implica que los investigadores empleen los mismos instrumentos del estudio original. La replicación es importante para validar los resultados de un estudio científico, y para ello se debe proporcionar el conjunto de datos obtenido, el protocolo de muestreo completo y las instrucciones precisas de procesamiento y análisis de datos para replicar la investigación realizada [16].

Para construir una colección de proyectos software que conduzca a resultados generalizables, en principio se debe considerar el marco muestral utilizado para elaborar la muestra, es decir, la lista de elementos seleccionables de la población. Las plataformas para compartir código fuente proporcionan datos de millones de repositorios, pero muchos de ellos no son aptos para la investigación en ISE [17, 18]. Es necesario tener en cuenta que cualquier persona puede crear un repositorio sin restricciones, por ello una cantidad considerable de repositorios contienen, por ejemplo: proyectos no ingenieriles, archivos de texto, imágenes, diapositivas, etc. [17]. Esto compromete la calidad de los datos en dichas plataformas y, en consecuencia, también la validez de los hallazgos obtenidos con estos datos. Dado el volumen de repositorios, no resulta viable revisar de forma manual cada proyecto en el marco muestral, por lo tanto, es necesario elaborar un conjunto de reglas por medio de los metadatos disponibles en estas plataformas para identificar los repositorios que cumplen con ciertos criterios de calidad.

Partiendo desde un marco muestral curado, se debe definir la estrategia de muestreo para seleccionar los sujetos. Existen estrategias de muestreo probabilísticas y no probabilísticas, y su elección dependerá del acceso que se tenga a los sujetos y de la solidez estadística esperada. En el muestreo probabilístico se emplean técnicas de aleatorización, donde cada miembro de la población tiene una probabilidad distinta de cero de ser incluido en la muestra. Por el contrario, el muestreo no probabilístico no involucra aleatorización. El caso ideal sería utilizar muestreo probabilístico dado que la mayoría de las pruebas estadísticas parten del supuesto que la muestra es aleatoria [19], lo que facilita la generalización estadística.

Asimismo, se debe evaluar la vigencia de los datos utilizados, es decir, que los datos continúen siendo válidos en el momento de realizar la investigación. La vigencia se manifiesta como una extensión de la validez externa [20] y por ello influye en la generalización de los resultados de un estudio [14]. La pérdida de vigencia cobra importancia en aquellos dominios donde los datos sufren cambios frecuentes y por consecuencia su distribución no se mantiene estable en el tiempo. Es relevante considerar este aspecto debido a la naturaleza evolutiva del software y las tecnologías de desarrollo. Por ejemplo, una colección basada en código fuente construida hace diez años difícilmente pueda ser representativa de la industria del software actual. Por lo tanto, para evitar la pérdida de vigencia se debe contemplar una estrategia de actualización de la muestra.

Sin embargo, estudios secundarios [21, 22] alertan sobre algunos problemas en la conducción de los estudios empíricos que comprometen la validez de los resultados informados. Por un lado, cuestiones relativas a la difusión de los instrumentos de la investigación que perjudican la replicación de los resultados, como la falta de instrucciones claras para conducir el estudio, la dificultad de acceso a los conjuntos de datos, la ausencia de paquetes de replicación y documentos de análisis, entre otras. Por otro lado, también incluyen problemas de índole metodológica en la conformación de las muestras: pocos sujetos (menos de cincuenta), sin criterios explícitos de evaluación de calidad o la utilización de técnicas de muestreo inapropiadas.

Como ejemplo de lo antes indicado, Jureczko & Madeyski [10] han recopilado métricas de código orientado a objetos e información sobre defectos de 15 proyectos de código abierto y 6 proyectos propietarios. Shepperd et al. [11] recopilaron métricas de tamaño y complejidad de

13 proyectos software de la NASA construidos con los lenguajes C y C ++. Las dos colecciones son de acceso público y se utilizan con frecuencia en estudios empíricos [5 - 7]. Sin embargo, ninguna de ellas informa la estrategia de muestreo, ambas contienen menos de 22 sistemas y sus últimas versiones se publicaron hace más de ocho años.

Por otra parte, las colecciones especializadas para estudios de estimación de esfuerzo [23 - 25] incluyen datos sobre la duración de los proyectos, la cantidad de líneas de código, puntos de función, cantidad de horas de trabajo, entre otros. Estas colecciones citadas fueron creadas hace más de 20 años, no se conoce la estrategia de muestreo empleada y en algunos casos tampoco el lenguaje de programación de los sistemas, no obstante, son utilizadas en estudios recientes [26 - 28].

Otra colección popular llamada *Qualitas Corpus* [8], contiene el código fuente de 112 proyectos Java de código abierto y su última versión fue publicada hace más de una década. A diferencia de las anteriormente mencionadas, ésta sigue una serie de lineamientos para seleccionar los proyectos, uno de los cuales establece la conservación de todos proyectos, incluso si se codificaron originalmente con versiones de Java que ahora están descontinuadas. El uso de proyectos con tecnología obsoleta puede garantizar la replicación de los estudios, pero también plantea problemas sobre la aplicabilidad de los resultados obtenidos en una población de proyectos actual.

Por ello, al revisar estudios secundarios sobre el estado de la ISE y las colecciones de referencia se observó la necesidad de un proceso estándar para construir y actualizar colecciones de proyectos software orientadas a estudios en la ISE que garanticen un grado de representatividad de las muestras construidas y la replicación de los estudios realizados. Por lo tanto, el objeto de investigación de esta tesis son los proyectos software de fuente abierta y las colecciones de proyectos empleadas en estudios empíricos.

Para abordar la necesidad anteriormente mencionada, se propone la construcción de un modelo de proceso [29] que guíe a los investigadores en la construcción de colecciones dirigidas a estudios en la ISE. En este sentido, se tomó la definición de Marbán et al. [30] donde un modelo de proceso es un conjunto de pasos o actividades para llegar a un objetivo determinado, con actividades que producen elementos como resultado (salidas) y reciben elementos que son necesarios para realizarlas (entradas) con el objetivo que el proceso sea repetible, gestionable y medible. De esta manera, el modelo propuesto

no solo detallará los pasos y reglas necesarios para construir y actualizar colecciones representativas, sino que también permitirá generar las herramientas para facilitar la replicación de los estudios y así mejorar la solidez de los resultados obtenidos.

1.2. Hipótesis y objetivos

De acuerdo a la problemática antes mencionada, se plantea la siguiente hipótesis de trabajo:

"Contar con un modelo de proceso para la construcción, actualización y curaduría de colecciones de proyectos software es una estrategia viable para generar resultados replicables y generalizables en estudios de la Ingeniería del Software Empírica".

El objetivo principal de la investigación propuesta es desarrollar un modelo de proceso para la construcción, actualización y curaduría de colecciones de proyectos software, para el apoyo a la conducción de estudios en ISE. Para la consecución del objetivo de este plan, se plantea el desarrollo de los siguientes objetivos específicos (OE):

OE1. Identificar las necesidades de construcción, curaduría y comunicación de una colección de proyectos software entre grupos de investigación y reportados en la literatura.

OE2. Construir el modelo de proceso basado en el análisis y el diseño de las reglas para la construcción y actualización de muestras de proyectos software, a través de las características y necesidades obtenidas del OE1.

OE3. Instanciar el modelo construido en el OE2 como una herramienta para un contexto específico y así automatizar el proceso de muestreo y actualización de una colección de proyectos software.

OE4. Implementar estrategias de muestreo y actualización para la instancia del OE3, y demostrar su efectividad en la construcción de colecciones representativas.

OE5. Validar el modelo de proceso en grupos de investigación externos, y de ser necesario, adaptar el modelo incorporando sugerencias aportadas en el proceso de validación.

La contribución principal de este trabajo de tesis involucra dos niveles: de proceso y de producto. En primer lugar, se elabora un modelo de proceso llamado SUM4SOFT para construir y mantener una colección curada de proyectos software (proceso). El modelo se instrumenta como una solución de software para validar y automatizar el enfoque. En segundo lugar, se comparte una colección de proyectos software de código abierto (producto), dirigida tanto a sectores académicos como de industria, creada y mantenida con el instrumento mencionado anteriormente. Para los investigadores, disponer de un modelo de proceso, y una colección curada construida con una técnica de muestreo apropiada reducirá la amenaza de producir resultados no generalizables ni repetibles en estudios orientados a la ISE. Para los profesionales, contar con una colección de referencia simplificaría la definición de umbrales predeterminados para perfiles de calidad en las herramientas de análisis de código.

1.3. Contexto de la investigación

Esta investigación doctoral se condujo en el marco de los siguientes proyectos:

- Proyectos de investigación PI-17F018 "Metodologías y herramientas emergentes para contribuir con la calidad del software", y PI-21F001 "Desarrollo de estudios empíricos en Ingeniería del Software" para los períodos 2018-2021 y 2022-2025 respectivamente, ambos acreditados por la Secretaría de Ciencia y Técnica de la Universidad Nacional del Nordeste (UNNE).
- Beca interna doctoral del Consejo Nacional de Investigaciones Científicas y Técnicas de la Argentina (CONICET) otorgada por la RESOL-2021-154- APN-DIR#CONICET para el período 2021-2025.
- Beca ALEARG para investigaciones doctorales de corta duración en Alemania otorgada por la Secretaría de Educación de la República Argentina y el Servicio de Intercambio Alemán a través de IF-2024-31870727-APN-DNCI#ME.

1.4. Estructura de la tesis

CAPITULO 1. Introducción: Este capítulo presenta la motivación de la tesis doctoral, el problema que apunta a resolver, su objetivo principal, el contexto en el cual la investigación fue desarrollada y la estructura del documento.

CAPITULO 2. Metodología: Este capítulo analiza el proceso de investigación llevado adelante en esta tesis y fundamenta la elección de las estrategias de investigación utilizadas.

CAPITULO 3. Estado del arte: Este capítulo describe el estado del arte en la construcción de muestras de proyectos software en ISE a través de dos revisiones de la literatura y abarca un estudio longitudinal sobre la evolución de una población de proyectos software.

CAPITULO 4. Desarrollo del artefacto: Este capítulo aborda la construcción de la versión final del artefacto propuesto y contribución principal de esta tesis: un modelo de proceso para la construcción y actualización de muestras de proyectos software orientadas a estudios empíricos en Ingeniería del Software.

CAPITULO 5. Validación del artefacto: Este capítulo expone las evaluaciones practicadas sobre la solución propuesta. Las mismas están centradas en validar que el instrumento de recolección de datos construido a través del modelo de proceso es capaz de generar muestras representativas de una población de proyectos software.

CAPITULO 6. Conclusiones: Este capítulo presenta las conclusiones de la investigación desarrollada en esta tesis doctoral y trabajos futuros.

APENDICE A. Referencias de los estudios primarios en el mapeo sistemático uno: Apéndice con las referencias a los estudios primarios seleccionados en el mapeo sistemático descrito en sección 3.1.

APENDICE B. Referencias de las colecciones en el mapeo sistemático dos: Apéndice con las referencias a las colecciones recolectadas en el mapeo sistemático descrito en sección 3.2.

APENDICE C. Guías de referencia para diseñar el protocolo de muestreo: Apéndice con la comparación de las referencias a guías de muestreo reportadas por Baltes & Ralph [21].

APENDICE D. Productos de trabajo de SUM4SOFT: Apéndice con los productos de trabajo diseñados para el modelo de proceso SUM4SOFT y sus especificaciones de acuerdo con el escenario de aplicación propuesto.

Referencias: Este capítulo lista las referencias bibliográficas utilizadas en el documento.

CAPITULO 2

Metodología

Este capítulo analiza el proceso de investigación llevado adelante en esta tesis y fundamenta la elección de las estrategias de investigación utilizadas. Este se organiza de la siguiente manera: la sección 2.1 indica el paradigma en el cual está enmarcada la tesis, la sección 2.2 detalla la metodología de investigación y los estudios empíricos conducidos y la sección 2.3 presenta la organización de las tareas planificadas.

2.1. Paradigma de investigación

Un paradigma de investigación responde preguntas metodológicas sobre formas legítimas de investigar la realidad y cómo confirmar que el conocimiento generado es válido. En particular, las comunidades científicas en el área de las Tecnologías de la Información y Comunicación (que abarca disciplinas como las Ciencias de la Computación, los Sistemas de Información y la Ingeniería del Software) se caracterizan por ser multiparadigmáticas, es decir, no existe un acuerdo universal sobre los fenómenos de interés ni en los métodos de investigación a emplear, por el contrario, recurren a la superposición de conjuntos de fenómenos y métodos de investigación [31]. Los paradigmas de investigación predominantes son: el positivismo y el interpretativismo.

El positivismo acepta el conocimiento que se basa en el sentido, la experiencia y la verificación positiva. Metodológicamente, busca una investigación objetiva y libre de sesgos; los investigadores son observadores imparciales y los hechos pueden ser descubiertos independientemente de sus creencias y valores [32], entre los cuales están los experimentos o las encuestas. Por el contrario, el interpretativismo afirma que el mundo social, incluidas las acciones sociales, solo puede entenderse mediante la comprensión de los significados y propósitos subjetivos que las personas atribuyen a sus acciones. A diferencia del positivismo, los investigadores interactúan

con los sujetos y utilizan sus ideas preconcebidas para guiar el proceso de investigación [33], como puede ser la investigación-acción o los estudios de caso.

Las estrategias de investigación positivistas proporcionan un conocimiento fiable pero superficial, mientras que las interpretativas ofrecen un conocimiento profundo, pero poco fiable. Para superar las debilidades de las respectivas estrategias de investigación los expertos sugieren que se combinen: las estrategias interpretativistas se deben utilizar para generar y proponer hipótesis de investigación, mientras que las positivistas se deben utilizar para verificarlas. En este sentido, la metodología ciencia de diseño [34] brinda un marco metodológico que admite la aplicación de ambos paradigmas para la creación de artefactos tecnológicos.

2.2. Descripción de la metodología ciencia de diseño

La ciencia de diseño (CD) es el estudio y la creación de artefactos desarrollados y utilizados por personas con el objetivo de resolver problemas prácticos de interés general. El objetivo es brindar soluciones innovadoras en forma de modelos, métodos y sistemas que ayuden a las personas a desarrollar, usar y mantener soluciones tecnológicas; y en el proceso, producir y comunicar nuevos conocimientos relevantes para una práctica global [34].

Este enfoque parece similar al diseño porque ambos se centran en el desarrollo de artefactos y apuntan a la novedad, es decir, están destinados a producir o investigar artefactos originales que difieren de los existentes, no obstante, sus propósitos son diferentes con respecto a la generalidad y contribución al conocimiento. Mientras que el diseño es un proceso para desarrollar una solución funcional a un problema que solo puede ser relevante para un solo actor, CD tiene como objetivo producir y comunicar conocimientos de interés general. Es decir, no solamente permite la generación del artefacto sino también conocimientos explícitos acerca de este, como su estructura, funciones y entorno; facilitando su reflexión e innovación.

El marco de trabajo de CD define cinco actividades principales: explicar el problema, definir objetivo y requisitos, diseñar y desarrollar el artefacto, demostrar el artefacto y evaluar el artefacto. Estas actividades se conducen de manera iterativa, es decir, se desarrollan de

acuerdo a los requerimientos de la investigación y no son conducidas de manera secuencial. Como el propósito es crear conocimiento nuevo y generalizable, se requiere que estas actividades usen estrategias y métodos de investigación rigurosos. Tales métodos son esenciales para crear resultados que puedan ser discutidos, evaluados y validados críticamente. A continuación, se detallan las tareas planificadas a partir de las actividades de CD.

2.2.1. Explicar el problema

El primer paso es definir el problema inicial de forma precisa, justificar su importancia e investigar las causas subyacentes. Esto consiste en conocimientos acerca de las características, el ambiente y posiblemente, las explicaciones a las causas de este problema.

Como se describió en el CAPITULO 1, el principal problema a abordar es que la comunidad de la ISE, en general, no aplica lineamientos uniformes para la conformación de colecciones de proyectos. Desde el punto de vista metodológico, en la mayoría de los casos los procesos de muestreo no son claros ni sistemáticos, y las estrategias utilizadas en este sentido no suelen ser apropiadas. También existen deficiencias en el reporte de los estudios, dado que los informes no facilitan los recursos, herramientas e instrucciones de los estudios. Esto afecta la generalización y replicación de los resultados obtenidos.

Dicho esto, para establecer el estado de arte y definir la base de conocimientos se realizaron tres estudios: dos estudios secundarios y un estudio longitudinal. Los estudios secundarios permitieron obtener una visión general de las colecciones de proyectos en ISE. En ambos casos se empleó la técnica: mapeo sistemático de la literatura (MS), siguiendo las guías de Kitchenham et al. [35] y Petersen et al. [36]. En el primer MS se evaluó cómo se crean las muestras en estudios empíricos, y en el segundo nuevamente se estudió la forma de construcción, pero de colecciones populares o de referencia.

Por otra parte, se condujo un estudio longitudinal para reunir evidencias del impacto del tiempo en la representatividad de una muestra. Un estudio longitudinal consiste en capturar observaciones repetidas (también llamadas olas) de las mismas unidades a lo largo de cierto tiempo en búsqueda de posibles cambios [37]. En el estudio se realizó un análisis evolutivo de una población de proyectos software en un período de seis años.

2.2.2. Definir objetivo y requisitos

El objetivo de esta actividad es identificar y delinear un artefacto que pueda abordar el problema explicado y obtener requisitos sobre ese artefacto. Se continúa extendiendo los detalles del problema, pero utilizando la solución propuesta para guiar la explicación. Primero se define el tipo de artefacto (construcción, modelo, método o instanciación) y una descripción de este. De acuerdo con esta clasificación, el artefacto construido es un modelo, es decir, una representación de posibles soluciones para problemas prácticos y que permite dar soporte a la construcción de otros artefactos. Los modelos son como planos o arquitecturas que pueden ser una base para un sistema, por lo que pueden ser instanciados en un sistema funcional y ser aplicados en la práctica.

En segundo lugar, se obtienen los requisitos del artefacto descripto. Los requisitos a incluir dependen de las características del problema, la solución delineada, las oportunidades tecnológicas, la investigación previa sobre soluciones documentadas a problemas similares, y los intereses y opiniones de las partes interesadas.

Para caracterizar el artefacto se utilizaron los resultados de las revisiones de la literatura. Primero, se condujeron MS para determinar las características que debe tener una colección, y los criterios de selección ampliamente reportados. Segundo, se revisó la literatura relevante para identificar las actividades y tareas inherentes al proceso de muestreo en ISE.

2.2.3. Diseñar y desarrollar el artefacto

Esta actividad consiste en crear el artefacto con las funcionalidades y estructura cumpliendo los requisitos de la actividad anterior. Esta actividad se diferencia de las demás en que no tiene como objetivo principal responder preguntas mediante la producción de conocimiento descriptivo o explicativo, sino producir conocimiento prescriptivo mediante la creación de un artefacto. El conocimiento prescriptivo consiste en formas para resolver problemas prácticos, ya sea a través de modelos generales, guías o procedimientos sistemáticos que faciliten la resolución de problemas.

En la construcción del artefacto se realizaron actividades individuales y grupales para favorecer la generación de ideas o soluciones orientadas

a abordar el problema en cuestión. En principio, para fomentar el pensamiento divergente, los participantes del desarrollo propusieron soluciones individuales que luego se compartieron en sesiones grupales de lluvia de ideas. Estas ideas se evaluaron y algunas de ellas se seleccionaron para consolidar la base de desarrollo del artefacto. Una vez determinadas estas ideas fundacionales, se diseñó un prototipo del artefacto para iniciar su construcción, el cual se adaptó hasta obtener el modelo definitivo.

2.2.4. Demostrar el artefacto

La demostración del artefacto consiste en probar su factibilidad en un caso real o ilustrativo. Es un tipo de evaluación, donde el artefacto resuelve una instancia del problema; y puede también ayudar a comunicar la idea detrás. En este caso, el modelo de proceso consiste en una serie de pasos generales, sin embargo, no determina cómo realizarlos, esto dependerá del estudio específico en el cual será utilizado. El estudio podría, por ejemplo, involucrar una población de proyectos con características particulares, analizar datos que imposibilitan ciertas estrategias de muestreo, emplear fuentes de datos no utilizadas con frecuencia, etc.

De esta manera, se empleó un prototipo del modelo de proceso de la actividad anterior para generar un instrumento de recolección de datos, implementando estrategias de muestreo y actualización de colecciones de proyectos software, a fin de con este instrumento extraer colecciones de prueba. Esta demostración permitió detectar y corregir errores iniciales en el diseño, mostrar el artefacto en un escenario de aplicación presentando su estructura y funcionamiento.

2.2.5. Evaluar el artefacto

En la última actividad se busca determinar si el artefacto es capaz de resolver el problema planteado en la primera actividad y hasta qué punto cumple con los requisitos de la segunda. Para ello se utilizó el instrumento de recolección de datos obtenido de la actividad anterior y se evaluó el artefacto en el contexto de un estudio de minería de repositorios de software (MRS) [38]. La minería de repositorios consiste en recolectar, analizar y entrecruzar datos sobre software de fuente abierta para descubrir información de interés. El objetivo del estudio fue calcular los umbrales de riesgo de métricas de código de tres

colecciones y compararlos entre sí. Se emplearon las siguientes colecciones: la última edición del *Qualitas Corpus* [8], una colección actual seleccionada desde cero aplicando la estrategia de muestreo implementada en el instrumento de recolección, y una versión del Qualitas actualizada también con el instrumento. Los resultados obtenidos de este estudio demostraron la capacidad de la herramienta para generar colecciones representativas.

Asimismo, se condujo un estudio de caso [39] en un grupo externo con dos objetivos: replicar el MRS para confirmar los resultados obtenidos y recibir sugerencias de mejora para el artefacto modelado. Un estudio de caso es un método empírico destinado a investigar los fenómenos en contextos reales ofreciendo una comprensión profunda de cómo y por qué ocurren y su vez revelar relaciones de causa y efecto [40]. Los fines pueden ser exploratorios y observacionales, es decir, sin intervenciones en el entorno del fenómeno; o bien para evaluar teorías [41], herramientas y métodos [42]. El estudio siguió un enfoque *ex post* y naturalista [34], es decir, que se desarrolló en un contexto real (naturalista) empleando el artefacto creado (*ex post*) para obtener alta validez externa, y posibilitar la generalización a situaciones similares.

2.3. Organización de las actividades

De acuerdo con la planificación propuesta en el marco de CD, en la Fig. 1 se esquematizó la organización de las tareas de cada actividad. El bloque superior representa las actividades de CD, y los bloques inferiores con bordes redondeados las tareas. La ubicación horizontal de las tareas determina a que actividad pertenecen, mientras que la vertical el orden de ejecución. Al final de cada capítulo se muestra el mismo esquema con un mapa de colores para sintetizar las tareas abordadas en el capítulo, las completadas en otros capítulos y las pendientes.

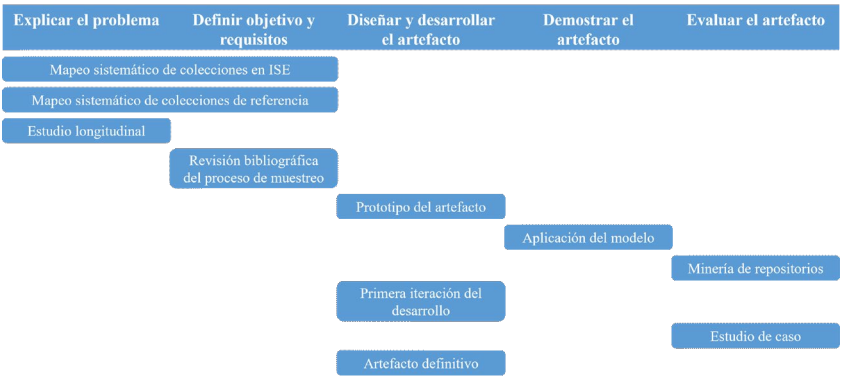

Fig. 1. Cronología de ejecución de las tareas de CD

Los estudios empíricos conducidos a lo largo de la investigación están enumerados en la Tabla 1 indicando las secciones donde cada uno de ellos se encuentran dentro del documento. Estos se pueden visualizar también en el esquema de la Fig. 1.

Tabla 1. Referencias a los estudios conducidos

Estudio conducido	**Referencia**
MS de colecciones en ISE	Sección 3.1
MS de colecciones de referencia	Sección 3.2
Estudio longitudinal	Sección 3.4
MRS	Sección 5.1
Estudio de caso	Sección 5.2

CAPITULO 3

Estado del arte

Este capítulo describe el estado del arte en la construcción de muestras de proyectos software en ISE a través de dos revisiones de la literatura, siguiendo las pautas para conducir MS en la Ingeniería del Software [35, 43]. Además, se incluye un estudio longitudinal [37] sobre la evolución de una población de proyectos software en los últimos seis años.

El capítulo se organiza de la siguiente manera: las secciones 3.1 y 3.2 presentan la planificación y resultados de los MS, la sección 3.3 es una síntesis de las conclusiones obtenidas en estos estudios secundarios, y la sección 3.4 describe el estudio longitudinal.

3.1. Mapeo sistemático de colecciones en ISE (MS1)

Para obtener una visión general del uso de colecciones de proyectos en la ISE se llevó a cabo un primer MS (MS1). El objetivo fue determinar los criterios de selección de los proyectos software que son insumos de estudios empíricos y caracterizarlos. Por otra parte, se identificaron los metadatos del código fuente de interés, las herramientas de extracción y los análisis estadísticos conducidos.

3.1.1. Planificación

La revisión sistemática contempló tres actividades: la selección de artículos, la extracción de datos y el análisis de los datos recopilados. Las preguntas de investigación (PI) que guiaron el proceso fueron las siguientes:

> **PI1. ¿Cuáles son los criterios de selección de los proyectos software objeto de estudios empíricos?** Motivación: Identificar

los criterios tenidos en cuenta por los investigadores para la selección de proyectos en la realización de estudios empíricos.

PI2. ¿Con qué tipo de proyectos trabajan los grupos de investigación para realizar estudios empíricos? Motivación: Conocer las características de los proyectos seleccionados con respecto al tipo de software y lenguaje de programación.

PI3. ¿Cuáles son los datos/metadatos que se extraen de estos proyectos? Motivación: Determinar los metadatos y métricas que son extraídos del código fuente de los proyectos para su posterior manipulación.

PI4. ¿Qué herramientas se utilizan para obtener estos datos/metadatos? Motivación: Determinar las herramientas usadas en los diferentes estudios para recolectar los metadatos.

PI5. ¿Qué análisis estadísticos se realizan con los datos/metadatos recopilados? Motivación: Definir los análisis estadísticos aplicados a los proyectos y los metadatos para interpretar los resultados obtenidos.

3.1.1.1. Selección de artículos

Para reunir los estudios primarios relevantes se llevó a cabo una búsqueda y selección iterativa en tres fases tal y como se indica en la Fig. 2.

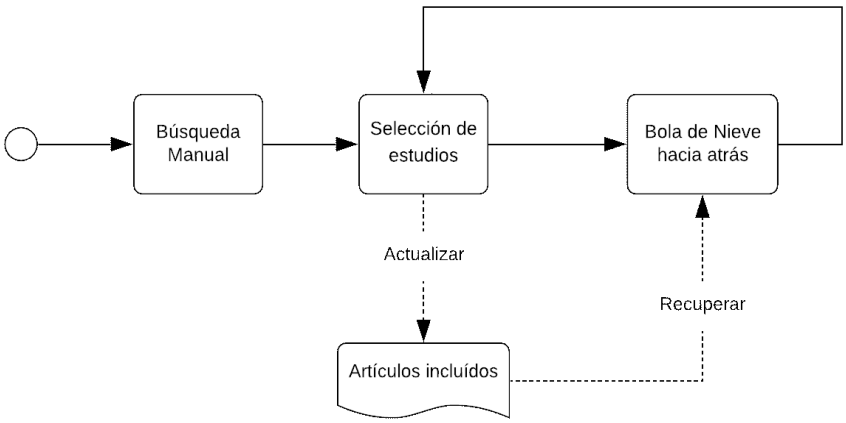

Fig. 2. Proceso de búsqueda y selección de artículos en el MS1

Búsqueda manual: se realizó una búsqueda manual en las principales conferencias y revistas enfocadas en ISE. Se seleccionó

la revista Empirical Software Engineering (EMSE) y las conferencias Empirical Software Engineering and Measurement (ESEM) e International Conference on Evaluation and Assessment in Software Engineering (EASE) por ser representativas del área de investigación y haber sido foros de consulta en trabajos relacionados [44, 45].
En consonancia con las estrategias planteadas por Zhang et al. [44] y Storey et al. [46], se comprendió el período de tiempo entre el uno de enero del 2013 hasta el 30 de noviembre del 2018 de EMSE y ESEM. Asimismo, se complementó el rango de búsqueda hasta el 31 de diciembre del 2020 conforme se publicaban nuevos números de la revista y el congreso. Asimismo, se incorporaron las ediciones del congreso EASE considerando el mismo período de tiempo. De esta manera, se consideraron las publicaciones de EMSE, ESEM y EASE de los últimos ocho años.

Selección de estudios: partiendo de los artículos obtenidos de la búsqueda manual se ejecutó una revisión dual de forma iterativa. Los artículos candidatos se incluyeron o excluyeron de acuerdo con los criterios presentes en la Tabla 2. Cada trabajo se analizó considerando resumen, introducción, metodología, resultados y conclusión. En cada iteración, se seleccionaron 15 estudios al azar y dos investigadores los revisaron. Estos anotaron sus decisiones junto con el símbolo del criterio de inclusión o exclusión en que se basó su decisión. Para medir el grado de acuerdo entre los investigadores se utilizó el estadístico *Kappa* de Cohen [47, 48] y se registró un valor de 0.77 demostrando un nivel de acuerdo alto.

Tabla 2. Criterios de inclusión y exclusión de estudios en el MS1

ID	Descripción
CI1	El artículo pertenece a ESE, EASE y ESEM.
CI2	Es un artículo completo.
CI3	En el artículo se realizan estudios empíricos.
CI4	El estudio empírico involucra la selección de un conjunto de proyectos software.
CE1	Artículos no técnicos (guías, artículos de introducción, editoriales, etc.)
CE2	Artículos duplicados.

Muestreo de bola de nieve: se complementó la búsqueda manual con el método de bola de nieve hacia atrás con los artículos incluidos. Las referencias sirvieron como artículos relacionados o similares. Aquellos artículos recopilados durante esta etapa

también se agregaron a la lista de candidatos y se seleccionaron de acuerdo con los criterios descritos en la Tabla 2.

3.1.1.2. Extracción de datos

Se utilizó un cuestionario de extracción de datos basado en las PI para recopilar la información relevante de los estudios primarios. Dos investigadores realizaron la extracción y un tercero revisó los datos para verificar la consistencia, comprobando que la información presente en el formulario corresponda a cada uno de los artículos. Como se ilustra en la Tabla 3, los datos recolectados fueron: información general (título, autores, año de publicación y fuente) e información relativa a las PI (PI1 – PI5). La información se extrajo exactamente como los autores la mencionaron en los manuscritos y los conflictos se discutieron y resolvieron internamente entre los investigadores. Se adoptó este enfoque para evitar la subjetividad y facilitar la replicación del estudio. Inicialmente, se realizó una prueba piloto con 15 artículos para calibrar el instrumento de extracción, refinar la estrategia y evitar diferencias entre los investigadores.

Tabla 3. Formulario de extracción de datos en el MS1

	ID	Item	Descripción
General	D1	Título	Título del estudio primario
	D2	Autores	Autores del estudio primario
	D3	Año de Publicación	Año de publicación del estudio primario
	D4	Fuente	Fuente donde se publicó el estudio primario
PI1	D5	Criterio	Criterio de selección de los proyectos utilizados en el estudio primario
PI2	D6	Lenguaje	Lenguaje de programación utilizado en los proyectos seleccionados
	D7	Tipo de proyecto	Tipo de proyecto de acuerdo con su funcionalidad
PI3	D8	Metadatos	Metadatos extraídos de los proyectos seleccionados en los estudios primarios
PI4	D9	Herramientas	Herramientas de recolección de metadatos
PI5	D10	Análisis	Tipo de análisis estadísticos que se realiza con los metadatos extraídos

3.1.1.3. Análisis de los datos

Las respuestas a las PI se analizaron de forma cualitativa mediante la codificación abierta de los textos encontrados en los estudios primarios [49] siguiendo la estrategia propuesta por Janssen & van der Voort [50]. El primero y segundo autor realizaron el proceso de forma separada y así se identificaron los desacuerdos, después el tercer investigador los resolvió visitando nuevamente la fuente de información considerando las justificaciones dadas por los primeros dos autores.

Como siguiente paso se utilizó la codificación cerrada [51] para identificar y resignificar las respuestas a taxonomías halladas en la literatura. Nuevamente los dos primeros autores del estudio realizaron la codificación inicial y los desacuerdos se resolvieron mediante la intermediación de un tercer investigador.

3.1.2. Ejecución

En la búsqueda manual se recolectaron 1489 artículos, de los cuales se incluyeron 1396 y excluyeron 93 según los criterios de exclusión. De los 1396 incluidos en la fase 2, 1285 se rechazaron por no cumplir alguno de los criterios de inclusión. A los 111 artículos aceptados se sumaron 11 en el proceso de bola de nieve hacia atrás, quedando finalmente la suma de 122, los cuales se listaron en el APENDICE A. Los resultados de cada fase se muestran Tabla 4.

Tabla 4. Detalle de cada fase de selección de estudios primarios en el MS1

Fase	Descripción	Incluidos	Excluidos
1	Búsqueda manual	1489	-
2	Selección por criterios de exclusión	1396	93
3	Selección por criterios de inclusión	111	1285
4	Bola de nieve hacia atrás	11	-

3.1.3. Resultados

A continuación, se responde a las PI con los datos recopilados. La planilla completa con los datos obtenidos a través del formulario de extracción está disponible en el enlace.[1]

[1] http://bit.ly/AnexoTablaCompleta

3.1.3.1. PI1: ¿Cuáles son los criterios de selección de los proyectos software objeto de estudios empíricos?

Los estudios se clasificaron de acuerdo a la estrategia de selección de proyectos teniendo en cuenta dos dimensiones de análisis: la estrategia general de muestreo y los criterios de selección específicos. La primera clasificación abarcó tres estrategias: seguir un método de selección propio, emplear una colección de referencia y muestreo aleatorio, de las cuales se registraron un 71%, 28% y 1% respectivamente (ver Fig. 3). En total, 18 colecciones de referencia se utilizaron en los 34 estudios que implementaron esta estrategia.

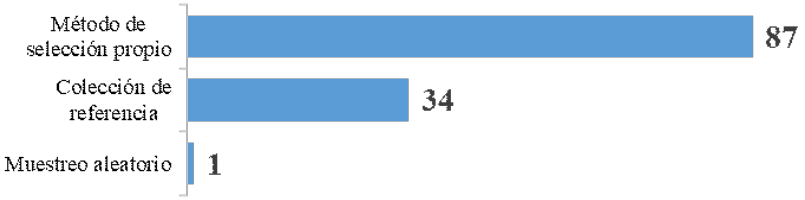

Fig. 3. Estrategias de selección de proyectos en el MS1 (n = 122)

Para la segunda clasificación (ver Tabla 5) se definieron 13 categorías no mutuamente excluyentes a partir de los datos extraídos de 94 estudios primarios. De esta manera, los criterios de selección reportados fueron: disponibilidad de los datos (37%), actividad del proyecto a lo largo del tiempo (30%), popularidad (28%), dominio de aplicación (26%), uso de herramientas para dar soporte al proceso de desarrollo (17%), detección de actividad reciente (13%), colaboración de un equipo de desarrollo (9%), software de calidad (9%) y documentación disponible (5%). La categoría con más ocurrencias correspondió a los criterios relacionados específicamente al estudio conducido (37%).

Tabla 5. Criterios de selección de proyectos en el MS1 (n = 94)

#	Criterios	Artículos
35	Específicas del caso de estudio del articulo	P2 P8 P9 P14 P18 P19 P22 P30 P34 P51 P55 P58 P61 P62 P64 P65 P66 P68 P72 P74 P79 P83 P86 P88 P90 P98 P100 P102 P105 P108 P110 P114 P116 P119 P122
35	Proyectos y metadatos disponibles públicamente	P1 P6 P7 P9 P13 P14 P16 P18 P21 P23 P25 P26 P29 P30 P34 P54 P64 P71 P72 P73 P76 P78 P83 P86 P90 P94 P95 P102 P104 P107 P113 P114 P119 P120 P122
28	Historia de desarrollo	P3 P26 P31 P44 P51 P53 P54 P55 P57 P58 P60 P64 P72 P75 P78 P79 P87 P90 P93 P94 P95 P100 P104 P106 P109 P111 P117 P119 P121

Tabla 5. Criterios de selección de proyectos en el MS1 (n = 94)

#	Criterios	Artículos
26	Popularidad	P1 P8 P10 P16 P19 P21 P26 P29 P34 P45 P48 P54 P56 P63 P64 P73 P76 P79 P86 P102 P104 P106 P109 P114 P116 P121
24	Dominio o tipo de software	P1 P4 P6 P8 P10 P12 P22 P27 P31 P32 P33 P50 P53 P55 P56 P57 P67 P73 P77 P84 P86 P99 P106 P121
24	Tamaño	P1 P2 P3 P4 P6 P16 P18 P33 P35 P55 P56 P57 P61 P68 P72 P78 P79 P85 P87 P88 P102 P106 P113 P120
16	Usan herramientas que dan soporte a procesos	P2 P3 P10 P17 P18 P44 P48 P58 P64 P69 P74 P75 P102 P107 P111 P117
12	Actividad reciente en el momento de recolección	P2 P14 P21 P25 P54 P75 P76 P78 P83 P94 P104 P117
8	Colaboración	P16 P26 P51 P72 P87 P104 P113 P114
8	Calidad	P9 P15 P16 P61 P68 P105 P108 P114
5	Documentación	P14 P16 P79 P106 P114

3.1.3.2. PI2: ¿Con qué tipo de proyectos trabajan los grupos de investigación para realizar estudios empíricos?

Los proyectos software presentes en las colecciones se clasificaron según el lenguaje de programación principal y el tipo de software, y dependiendo del estudio se podía encontrar colecciones con proyectos de diversos lenguajes y tipos. En total 110 artículos reportaron los lenguajes de programación de los proyectos, de los cuales en un 79% se construyeron en Java, seguidos por C y C++ en un 27% y 24% respectivamente. En la Fig. 4 se muestra la cantidad de estudios que reportaron cada lenguaje.

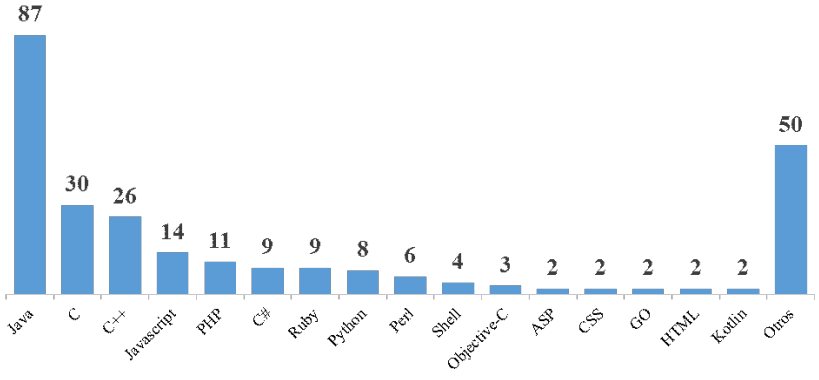

Fig. 4. Lenguajes de programación de proyectos en el MS1 (n = 110)

Para identificar los tipos de software se buscaron los nombres y descripciones de proyectos mencionadas en los estudios. En los casos donde los datos extraídos no eran suficientes para determinar el tipo de software utilizado, se realizó una búsqueda y consulta manual. De los estudios primarios seleccionados en 96 se logró identificar el tipo de software, y cada proyecto se clasificó según la taxonomía de Forward & Lethbridge [52]. La Fig. 5 refleja la cantidad de ocurrencias de cada categoría, siendo estas: aplicación de uso general (88%), diseño y construcción de software (74%), servidor (52%), redes y comunicaciones (33%), sistema operativo (14%) y sistema embebido (13%).

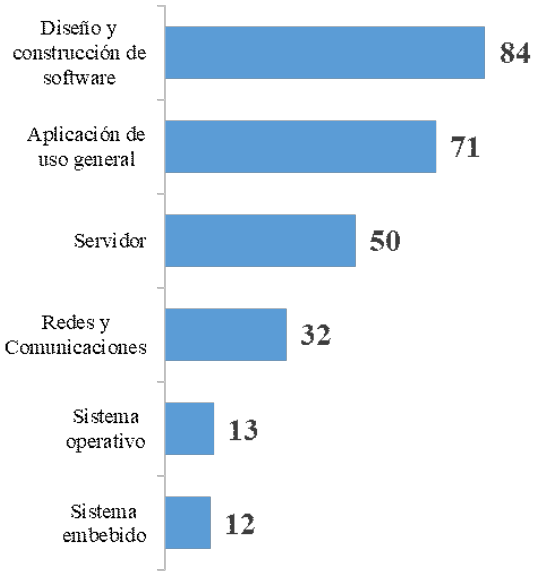

Fig. 5. Tipos de software en el MS1 (n = 96)

3.1.3.3. PI3: ¿Cuáles son los datos/metadatos que se extraen de estos proyectos?

A partir de la taxonomía de Elmidaoui et al. [53], en la Tabla 6 se agruparon los metadatos extraídos del código fuente de los proyectos. Esta taxonomía solo incluye métricas de código, por ello se incorporó la categoría *smells* en el código, dado que son estructuras extraídas también del código fuente que amenazan su calidad [54] y funcionan como método viable para el mantenimiento del software [55]. Dicho esto, de los 54 estudios que extrajeron metadatos del código para

conducir su investigación, un 67% emplearon métricas de tamaño del código fuente, un 63% métricas de diseño y un 32% *smells* en el código.

Tabla 6. Metadatos obtenidos del código fuente en el MS1 (n = 54)

#	Categorías de Metadatos	Artículos
36	Tamaño Código Fuente	P6 P9 P10 P11 P12 P18 P20 P21 P25 P26 P29 P38 P41 P46 P49 P50 P55 P56 P58 P65 P66 P69 P72 P74 P80 P81 P83 P85 P86 P89 P96 P97 P100 P105 P114 P119
34	Diseño	P6 P9 P12 P15 P16 P18 P20 P21 P25 P29 P37 P41 P46 P49 P50 P55 P59 P65 P69 P74 P78 P80 P81 P84 P85 P96 P97 P102 P105 P113 P115 P117 P118 P119
17	*Smells*	P2 P3 P7 P10 P12 P18 P19 P21 P29 P37 P44 P56 P63 P78 P79 P85 P105

Asimismo, en la Tabla 7 se clasificaron las métricas de diseño en siete subcategorías: complejidad (43%), acoplamiento (35%), diagrama de clases (33%), herencia (28%), cohesión (28%), encapsulamiento (17%) y polimorfismo (2%). Dado que muchos estudios emplearon uno o más tipos de metadatos, en ambas clasificaciones la sumatoria de ocurrencias supera el número de artículos (n).

Tabla 7. Métricas de diseño en el MS1 (n = 54)

#	Subcategorías de Diseño	Artículos
23	Complejidad	P9 P12 P18 P20 P21 P25 P29 P37 P41 P46 P49 P50 P55 P69 P74 P80 P81 P85 P97 P105 P117 P118 P119
19	Acoplamiento	P6 P9 P12 P15 P20 P25 P29 P37 P49 P59 P65 P69 P78 P80 P81 P85 P96 P118 P119
18	Diagrama de Clases	P6 P9 P12 P20 P25 P29 P49 P50 P65 P69 P78 P80 P81 P96 P102 P113 P118 P119
15	Herencia	P9 P12 P16 P20 P25 P37 P49 P69 P78 P80 P85 P105 P115 P118 P119
15	Cohesión	P6 P9 P20 P25 P37 P49 P65 P69 P78 P80 P81 P84 P85 P118 P119
9	Encapsulamiento	P12 P20 P49 P65 P69 P78 P80 P96 P118
1	Polimorfismo	P78

3.1.3.4. PI4: ¿Qué herramientas se utilizan para obtener estos datos/metadatos?

En total, 61 artículos mencionaron de manera explícita los nombres de las herramientas, de los cuales 44 corresponden a herramientas que estrictamente generen metadatos a partir del código. Debido al número de herramientas reportadas (68) en la Tabla 8 se clasificaron de acuerdo a su funcionalidad: calcular métricas de software (57%), analizar la estructura y dependencias del código fuente (37%), detectar *smells* o vulnerabilidades (37%), refactorizar el código (9%), generar pruebas automáticamente (9%) y detectar patrones de diseño (5%). Al igual que en preguntas anteriores, las categorías están solapadas por lo tanto el número de ocurrencias supera n.

Tabla 8. Herramientas para generar metadatos en el MS1 (n = 44)

#	Tipo de Herramientas	Artículos
25	Cálculo de métricas de software	P2 P3 P4 P9 P12 P17 P18 P20 P21 P25 P26 P46 P50 P66 P67 P69 P74 P78 P83 P85 P105 P107 P113 P116 P119
17	Análisis de la estructura y dependencias del código fuente	P2 P4 P8 P9 P16 P21 P25 P35 P46 P50 P55 P57 P64 P74 P90 P116 P119
17	Detección de *smells* o vulnerabilidades	P2 P3 P12 P18 P19 P20 P29 P37 P44 P46 P56 P57 P63 P67 P78 P85 P105
4	Refactorización de código	P9 P21 P54 P84
4	Generación automática de pruebas	P42 P43 P62 P99
2	Detección patrones de diseño	P57 P78

3.1.3.5. PI5: ¿Qué análisis estadísticos se realizan con los datos/metadatos recopilados?

Los investigadores necesariamente realizan un análisis previo a los datos recabados para exponer los hallazgos. Este proceso permite identificar las características que posee la muestra y a su vez seleccionar los métodos apropiados para generar los resultados. La Tabla 9 enumera los tipos de procedimientos estadísticos de los 88 estudios que informaron esto, agrupados en las dos áreas generales de la estadística: estadística inferencial (78%) y estadística descriptiva (73%).

Tabla 9. Tipos de análisis estadísticos (n = 88)

#	Estadística	Artículos
69	Inferencial	P1 P2 P3 P4 P5 P9 P13 P15 P16 P19 P20 P22 P23 P25 P26 P31 P32 P33 P34 P35 P36 P37 P39 P40 P42 P49 P50 P51 P54 P55 P56 P57 P59 P60 P61 P62 P65 P67 P68 P69 P70 P73 P74 P75 P77 P78 P79 P80 P81 P83 P85 P86 P90 P91 P94 P96 P97 P99 P101 P103 P110 P111 P113 P114 P116 P118 P119 P120 P121
64	Descriptiva	P2 P3 P4 P6 P10 P13 P14 P19 P20 P22 P23 P24 P25 P26 P29 P31 P32 P33 P35 P36 P37 P41 P42 P43 P54 P56 P57 P58 P60 P61 P64 P68 P69 P71 P72 P74 P76 P77 P79 P80 P83 P84 P86 P87 P90 P91 P92 P93 P94 P95 P96 P99 P101 P103 P105 P106 P110 P111 P113 P114 P116 P118 P119 P120

Tomando como referencia las clasificaciones de Sheskin [19], los estadísticos inferenciales se clasificaron en pruebas paramétricas y no paramétricas (ver Tabla 10).

Tabla 10. Procedimientos estadísticos inferenciales (n = 69)

#	Procedimientos Inferenciales	Artículos
61	Test No Paramétrico	P1 P2 P4 P5 P9 P13 P15 P16 P19 P20 P22 P23 P25 P26 P31 P32 P33 P34 P35 P36 P37 P42 P49 P50 P51 P54 P55 P56 P57 P59 P60 P61 P62 P67 P68 P69 P73 P74 P75 P77 P78 P79 P80 P83 P86 P90 P91 P94 P96 P99 P101 P103 P110 P111 P113 P114 P116 P118 P119 P120 P121
22	Test Paramétrico	P3 P9 P13 P23 P25 P33 P39 P40 P60 P65 P74 P75 P80 P81 P85 P90 P94 P97 P99 P103 P111 P121

De la misma manera, los descriptivos se dividieron en cinco categorías: tamaño del efecto, medida de variabilidad, medida de tendencia central, medida de asimetría y medida de curtosis (ver Tabla 11).

Tabla 11. Procedimientos estadísticos descriptivos (n = 64)

#	Procedimientos Descriptivos	Artículos
51	Tamaño del efecto	P2 P3 P4 P6 P10 P14 P19 P20 P23 P26 P29 P32 P33 P35 P37 P41 P42 P54 P56 P57 P58 P60 P64 P71 P74 P76 P77 P79 P80 P83 P84 P86 P87 P90 P92 P93 P95 P96 P99 P101 P103 P105 P106 P110 P111 P113 P114 P116 P118 P119 P120
26	Medida de Variabilidad	P2 P3 P6 P13 P22 P23 P24 P25 P31 P32 P35 P36 P37 P43 P60 P64 P68 P69 P72 P74 P77 P80 P91 P94 P103 P111
25	Medida de Tendencia Central	P2 P3 P6 P13 P22 P23 P24 P25 P31 P32 P35 P36 P37 P43 P60 P64 P68 P69 P72 P74 P77 P80 P94 P103 P111
3	Medida de Asimetría	P13 P43 P61
2	Medida de Curtosis	P13 P43

3.2. Mapeo sistemático de colecciones de referencia (MS2)

Durante la ejecución del primer MS, se detectó que el 28% de los artículos emplearon en total 18 colecciones de referencia. Entonces, para complementar los resultados reportados, se consideró pertinente extender el estudio a estas colecciones consumidas de manera frecuente por la comunidad científica en un segundo MS (MS2). En este sentido, los objetivos fueron: identificar los criterios de selección de los proyectos software empíricos, sus características, las métricas extraídas del código y las herramientas de extracción. Asimismo, se estudiaron las edades de estas colecciones y las motivaciones de creación.

3.2.1. Planificación

Nuevamente, se estipularon tres actividades para el MS: la selección de colecciones, la extracción de datos y el análisis de los datos recopilados. Las PI que guiaron el proceso fueron las siguientes:

PI1. ¿Por qué se creó la colección? Motivación: Determinar los tipos de estudios en ISE a los cuales las colecciones se dirigen.

PI2: ¿Cuándo fue la última actualización de la colección? Motivación: Buscar el año en que se actualizaron las colecciones por última vez.

PI3: ¿Cuáles son las características de los proyectos comprendidos en estas colecciones? Para responder esta pregunta, se definieron dos subpreguntas:

PI3.1. ¿Cuáles son los criterios considerados para seleccionar los proyectos? Motivación: Identificar cuáles son los criterios tenidos en cuenta por los investigadores para la selección de proyectos.

PI3.2. ¿Cuáles son los lenguajes de programación de los proyectos? Motivación: Reconocer los lenguajes de programación de los proyectos.

PI4. ¿Qué métricas del código fuente fueron extraídas de los proyectos? Motivación: Establecer las métricas extraídas del código fuente de cada proyecto.

PI5. ¿Qué herramientas se utilizaron para obtener datos relacionados con el código de los proyectos? Motivación:

Identificar las herramientas utilizadas para generar métricas y otros componentes relacionados con el código de los proyectos.

3.2.1.1. Selección de colecciones

Para recolectar las colecciones de referencia se llevó a cabo el proceso de la Fig. 6, el cual está dividido en dos partes: la selección de estudios en ISE y la selección de colecciones.

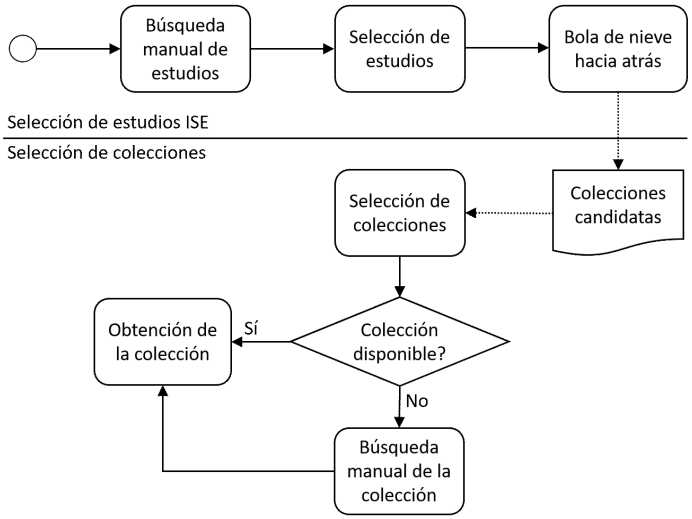

Fig. 6. Proceso de búsqueda y selección de colecciones en el MS2

Búsqueda manual: se realizó una búsqueda manual en las conferencias y revistas referentes en la Ingeniería del Software listadas en la Tabla 12 en el período comprendido entre el uno de enero del 2013 hasta el 31 de diciembre del 2021.

Tabla 12. Foros de Ingeniería del Software

Siglas	Nombre del Foro	Tipo
EASE	Evaluation and Assessment in Software Engineering	Conferencia
ESEM	Empirical Software Engineering and Measurement	Conferencia
MSR	Mining Software Repositories	Conferencia
PROMISE	Predictive Models and Data Analysis in Soft. Eng.	Conferencia
EMSE	Empirical Software Engineering	Revista
IST	Information and Software Technology	Revista
JSS	Journal of Systems and Software	Revista

Selección de estudios: partiendo de los artículos obtenidos de la búsqueda manual, se buscaron los artículos en ISE que emplearon una colección creada en otro estudio. Los trabajos se analizaron considerando resumen, introducción, metodología, resultados y conclusión. Posteriormente se extendió la lista de artículos con otros estudios relacionados que también hayan trabajado con una colección generada en otra fuente.

Muestreo de bola de nieve hacia atrás: con la lista de estudios seleccionados, se obtuvieron las colecciones candidatas citadas por los autores. Además de la referencia, se extrajo información de la misma como su nombre, autores, estudio fundacional, enlaces a documentos, entre otros.

Selección de colecciones: se incluyeron o excluyeron las candidatas de acuerdo con los criterios presentes en la Tabla 13. Se tuvieron en cuenta estudios científicos y también literatura gris (es decir, literatura no publicada formalmente como libros o artículos de revistas [56]) cuando la referencia proporcionada no era un artículo científico o no existía una descripción del conjunto de datos. Dos investigadores seleccionaron aleatoriamente las colecciones candidatas, las revisaron, y registraron sus decisiones. Para medir el grado de acuerdo entre los investigadores se utilizó el estadístico *Kappa* de Cohen contabilizando un valor de 0.7.

Tabla 13. Criterios de inclusión y exclusión de colecciones (MS2)

ID	Descripción
CI1	La colección está disponible o su contenido esta detallado.
CI2	La colección fue utilizada en un estudio en ISE.
CE1	Candidatas duplicadas.
CE2	La fuente referenciada como una colección en el estudio seleccionado es una plataforma para compartir código (por ejemplo, Github, Sourceforge, Openhub, etc.)
CE3	Los datos recopilados no dan cuenta que la colección haya sido creada con una estrategia de muestreo aparente (por ejemplo, una instantánea de todos los repositorios en Github)
CE4	La colección es un subconjunto de otra colección previamente seleccionada.

Obtención de la colección: una vez incluidas las colecciones, se pretendió descargarlas. En el caso que estuviesen en línea se procedió a obtener el recurso, en el caso contrario, se emprendió una búsqueda manual a través de motores de búsqueda y

repositorios especializados (por ejemplo, zenodo[2] y tera-PROMISE[3]).

3.2.1.2. Extracción de datos

De manera análoga al procedimiento de extracción de datos detallado en la sección 3.1.1.2, se utilizó un formulario de extracción de datos basado en las PI para recopilar información acerca de las colecciones. El formulario de la Tabla 14 incluye información general e información relativa a las PI.

Tabla 14. Formulario de extracción de datos (MS2)

	ID	Item	Descripción
General	D1	Título	Título artículo fundacional.
	D2	Autores	Autores del estudio.
	D3	Año de publicación	Año de publicación de la colección.
PI1	D4	Propósito	Motivación para construir la colección.
PI2	D5	Última actualización	Año de la última actualización de la colección.
PI3	D6	Criterio	Criterio de selección de los proyectos en la colección.
	D7	Lenguaje	Lenguaje de programación de los proyectos.
PI4	D8	Métricas	Métricas del código extraídas de los proyectos en la colección.
PI5	D9	Herramientas	Herramientas para recolectar métricas y otros componentes del código

3.2.1.3. Análisis de datos

Las estrategias para analizar los textos recopilados, al igual que la sección 3.1.1.3, fueron dos: codificación abierta [50] y codificación cerrada [51].

3.2.2. Ejecución

La búsqueda manual inicial resultó en 5357 artículos, de los cuales se seleccionaron 178 estudios en ISE que emplearon una colección de referencia. Se procedió con el muestreo de bola de nieve hacia atrás para recuperar las colecciones referenciadas, resultando en 347

[2] https://zenodo.org
[3] http://openscience.us/repo

colecciones candidatas. Al conjunto de candidatas se les aplicaron los criterios de exclusión e inclusión, quedando finalmente la suma de 74 colecciones que se enumeraron en el APENDICE B. Los resultados de cada fase se muestran Tabla 15.

Tabla 15. Detalle de cada fase de selección de colecciones de referencia (MS2)

Fase	Descripción	Incluidos	Excluidos
1	Búsqueda manual	5357	-
2	Selección de estudios	179	5178
3	Muestreo de bola de nieve	347	-
4	Selección por criterios de exclusión	81	266
5	Selección por criterios de inclusión	74	7

3.2.3. Resultados

A continuación, se responde a las PI con los datos recopilados. La planilla completa con los datos obtenidos a través del formulario de extracción está disponible en el enlace.[4]

3.2.3.1. PI1: ¿Por qué se creó la colección?

Las colecciones se clasificaron por el tipo de estudio en ISE a los cuales fueron dirigidos. Esto se determinó en base a los datos contenidos en las mismas y las descripciones presentes en los sitios web y estudios fundacionales. En la Tabla 16 se presentan se presentan las diez categorías: defectos del software (37%), estimación del software (23%), mantenimiento del software (15%), pruebas automáticas (8%), estudios del código (5%), seguridad del software (4%), revisión por pares del código (3%), integración continua (3%), diversidad de desarrolladores (1%) y latencia de *pull-requests* (1%).

[4] https://bit.ly/datasets-2022-annex

Tabla 16. Propósito de creación de las colecciones (n = 74)

#	Objetivo de la colección	Referencias
27	Defectos del software	S1 S2 S22 S24 S25 S28 S32-S37 S40 S42 S43 S45 S47 S48 S55 S56 S59 S60 S66 S69 S72-S74
17	Estimación del software	S3-S12 S15 S26 S27 S29 S30 S41 S44
11	Mantenimiento del software	S17 S31 S51 S53 S57 S63-S65 S67 S68 S71
6	Pruebas automáticas	S13 S49 S50 S54 S61 S62
4	Estudios del código	S14 S16 S21 S23
3	Seguridad del software	S38 S39 S58
2	Revisión por pares del código	S18 S52
2	Integración continua	S46 S70
1	Diversidad de desarrolladores	S19
1	Latencia de pull-requests	S20

Los estudios de defectos investigan los errores en el código que impiden el funcionamiento correcto del software [57], y abordan temáticas como la detección automática, clasificación o predicción de errores. La estimación del software involucra el estudio de modelos para estimar el costo para planificar y gestionar un proyecto [58]. Los estudios de mantenimiento buscan validar técnicas para gestionar la calidad del software y actividades relacionadas, por ejemplo, detección de *smells* [12], refactorización del código [59], evaluación de la deuda técnica [60], entre otras. Las colecciones para pruebas automáticas apuntan a automatizar la generación de: pruebas unitarias [61], de datos de entrada [62] y pruebas de mutaciones [63].

3.2.3.2. PI2: ¿Cuándo fue la última actualización de la colección?

Para determinar cuándo se actualizaron las colecciones por última vez, se realizaron búsquedas manuales en motores de búsqueda en línea. De las 74 colecciones seleccionadas, se accedió a los datos de 49. Dicho esto, en la Fig. 7 se presenta una línea de tiempo con las colecciones de público acceso, y 30 de ellas se actualizaron en el año 2016 o antes.

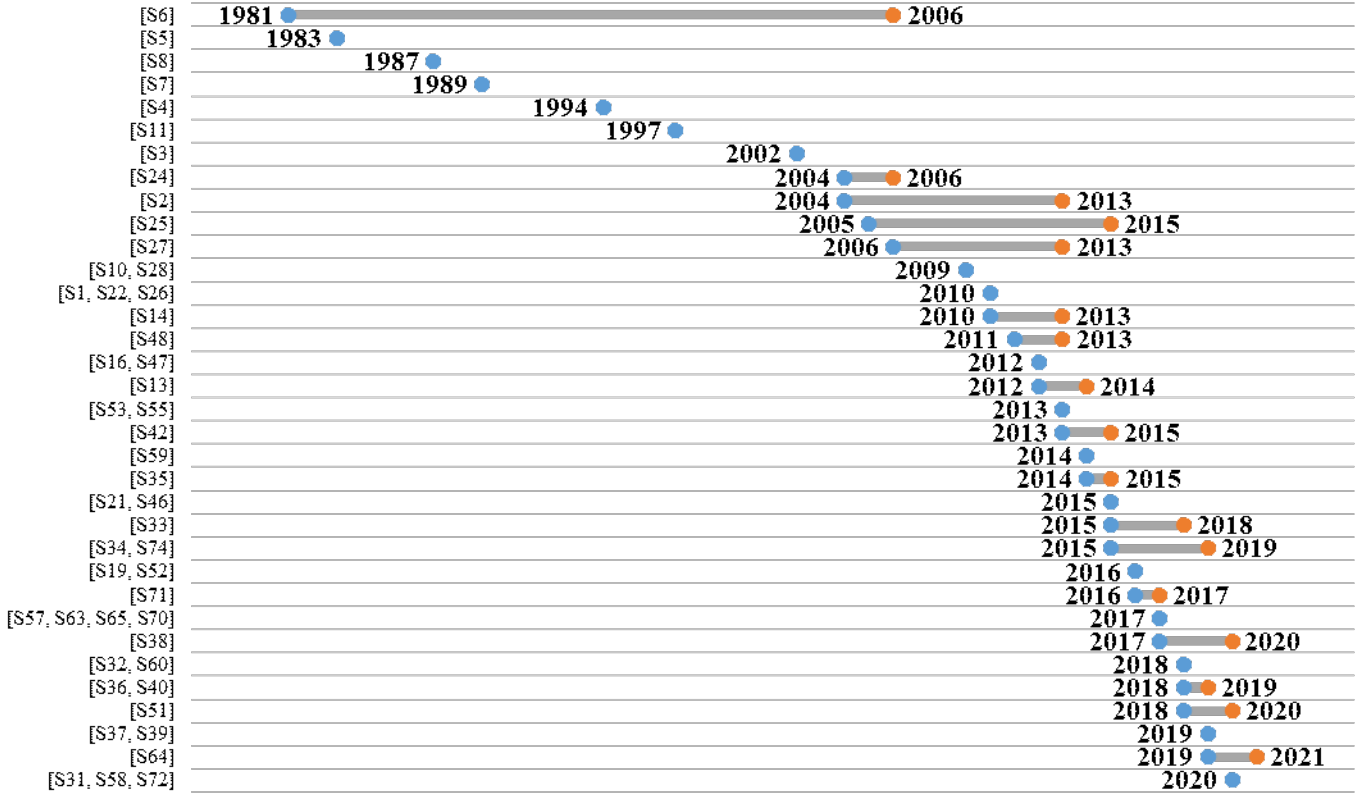

Fig. 7. Fecha de creación y última actualización de las colecciones de acceso público

3.2.3.3. PI3.1: ¿Cuáles son los criterios considerados para seleccionar los proyectos?

En la Tabla 17 se categorizaron los criterios de selección de proyectos siguiendo la clasificación de la sección 3.1.3.1. De esta manera, 47 colecciones evaluaron: en un 64% cuestiones específicas al tipo de estudio, en un 34% la disponibilidad de los datos y el uso de herramientas para dar soporte al proceso de desarrollo, en un 32% el tamaño, en un 23% la actividad del proyecto a lo largo del tiempo, en un 21% la popularidad, en un 19% el dominio de aplicación, y en un 6% la colaboración en el equipo de desarrollo.

Tabla 17. Criterios de selección de proyectos en el MS2 (n = 47)

#	Criterios	Referencias
30	Específicas del caso de estudio del articulo	S8 S13 S14 S16 S20 S21 S23 S25 S29 S31 S33 S35 S36 S39-S42 S46 S50 S53 S56 S60 S61 S63 S65 S67 S68 S70 S71 S73
16	Proyectos y metadatos disponibles públicamente	S14 S21 S22 S25 S29 S34 S36 S37 S50 S52 S57 S59 S60 S61 S65 S73
16	Usan herramientas que dan soporte a procesos	S16 S18 S20 S32 S36 S40 S46 S52 S53 S56 S59 S64 S70-S72 S74
15	Tamaño	S8 S21 S22 S25 S32 S34 S36 S38 S51 S63 S64 S65 S67 S73 S74
11	Actividad en el proyecto a lo largo del tiempo	S19 S22 S25 S32 S34 S51 S63 S64 S67 S71 S74
10	Popularidad	S20 S35 S37 S46 S50 S57 S60 S65 S70 S72
9	Dominio o tipo de software	S15 S31 S49 S51 S56 S65-S67 S73
3	Colaboración	S19 S51 S67

3.2.3.4. PI3.2: ¿Cuáles son los lenguajes de programación de los proyectos?

En total 62 autores informaron los lenguajes de programación de los proyectos, y la mayoría de ellos recolectaron sistemas construidos con Java (77%). En la Fig. 8 se pueden observar todos los lenguajes detectados, por ejemplo: C, Python, C++, Cobol, Javascript, Ruby, PHP, entre otros.

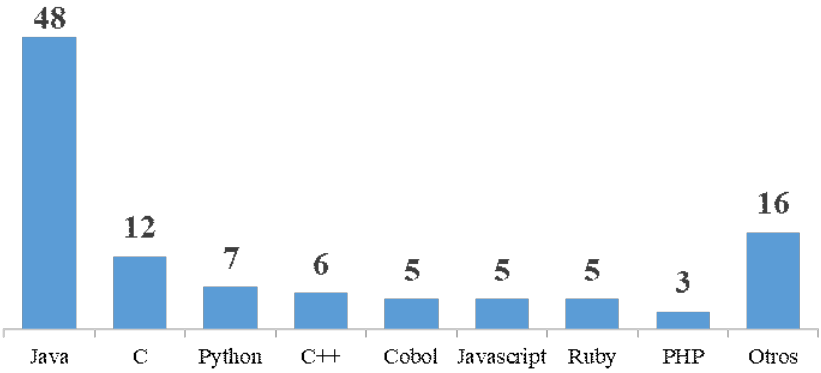

Fig. 8. Lenguajes de programación de proyectos en el MS2 (n = 62)

3.2.3.5. PI4: ¿Qué métricas del código fuente fueron extraídos de los proyectos?

Las colecciones que recolectaron métricas de código (26) se categorizaron empleando la taxonomía de Elmidaoui et al. [53], donde casi todas ellas (96%) calcularon métricas de tamaño, y la mitad (46%) métricas de diseño (ver Tabla 18).

Tabla 18. Métricas de código fuente en el MS2 (n = 26)

#	Tipo de métrica	Referencias
25	Tamaño Código Fuente	S1-S6 S8 S10 S21 S23 S24 S28 S32 S43-S46 S57 S61 S62 S64 S67 S68 S73 S74
12	Diseño	S1 S2 S21-S24 S28 S32 S57 S64 S68 S74

A su vez, en la Tabla 19 las métricas de diseño se disgregaron en cinco grupos: complejidad, diagrama de clases, acoplamiento, herencia y cohesión.

Tabla 19. Métricas de diseño (MS2)

#	Métricas de diseño	Referencias
11	Complejidad	S1 S2 S21 S22 S24 S28 S32 S57 S64 S68 S74
8	Diagrama de Clases	S1 S21-S24 S32 S57 S74
7	Acoplamiento	S1 S22-S24 S57 S68 S74
5	Herencia	S1 S22 S24 S57 S74
4	Cohesión	S1 S22 S57 S74

3.2.3.6. PI5: ¿Qué herramientas se utilizaron para obtener datos relacionados con el código de los proyectos?

Análogamente a lo descripto en la sección 3.1.3.4, las 18 colecciones que reportaron herramientas para analizar el código, se clasificaron en la Tabla 20 según el tipo de herramienta en: cálculo métricas de software (78%), detección de *smells* o vulnerabilidades (50%), análisis de la estructura y dependencias del código fuente (17%), cobertura de código (17%), refactorización del código (11%), generación automática de pruebas (6%).

Tabla 20. Herramientas para generar metadatos en el MS2 (n = 18)

#	Tipo de Herramienta	Referencias
14	Cálculo métricas de software	S1 S21 S22 S28 S32 S46 S49 S57 S62 S63 S64 S67 S68 S74
9	Detección de *smells* o vulnerabilidades	S1 S22 S32 S39 S63 S64 S68 S73 S74
3	Análisis de la estructura y dependencias del código fuente	S21 S39 S57
3	Cobertura de código	S50 S60 S62
2	Refactorización de código	S57 S64
1	Generación automática de pruebas	S49

3.3. Conclusiones de los estudios secundarios

Para conocer el estado de arte de la construcción de colecciones orientadas a estudios en ISE se condujeron dos mapeos sistemáticos. Primero, se analizaron los estudios que emplearon colecciones de proyectos software y posteriormente se apuntó específicamente a las colecciones de referencia. En ambos casos se llegaron a resultados consistentes en las preguntas de investigación relacionadas con los criterios de selección, la caracterización de los proyectos, los metadatos y el uso de herramientas.

Empezando por los criterios de selección, la categoría con mayor cantidad de ocurrencias fue la de criterios específicos con el caso de estudio, evidenciando la falta de estrategias de selección estandarizadas en la comunidad de la ISE. Otra cuestión a remarcar fue la ausencia del muestreo probabilístico, en total solo dos estudios [64, 65] reportaron técnicas de aleatorización. No obstante, se recopilaron algunos criterios compartidos por las colecciones que podrían servir en la identificación

de proyectos viables para la investigación: 1) disponibilidad del código fuente, 2) utilización de herramientas para dar soporte a procesos, 3) tamaño, 4) historia de desarrollo, 5) popularidad, 6) colaboración del equipo de desarrollo y 7) actividad reciente. Además, siendo que la mayoría de estudios (75%) recopilaron colecciones con proyectos construidos en Java, sería conveniente facilitar una colección que contenga software de este lenguaje de programación.

Los investigadores casi en su totalidad no consideraron utilizar la información más reciente para conducir sus estudios (solo sucedió en el 11% de los estudios del MS1). En este sentido, la Fig. 7 tampoco muestra que exista un interés prevalente por mantener las colecciones con datos actualizados, la mayoría de ellas nunca se actualizaron desde su concepción. Existen excepciones en donde los autores procuraron preservarlas difundiendo nuevas versiones, como las colecciones *SF100* [64], *Eclipse Bug Data* [66] o *Software-artifact Infrastructure Repository* [67]; pero eventualmente dejaron de hacerlo. Tal y como se ha indicado en el CAPITULO 1, poblaciones que sufren cambios de manera frecuente requieren la implementación de estrategias que actualicen las muestras para que estas permanezcan representativas. Por otra parte, la difusión de los datos no está garantizada en todos los casos, el 32% de las colecciones del MS2 no son de público acceso.

Con respecto a los metadatos, se extrajeron métricas de código, reportes de defectos, pruebas unitarias, datos de repositorio, *smells* de código, métricas del proceso de desarrollo, entre otras. Al revisar los datos en cada tipo de colección, se observó una relación entre la disponibilidad de la información y la edad; por ejemplo, las colecciones para estimación de software contienen principalmente métricas de proceso, siendo el 53% de estas de público acceso, y la más reciente de ellas se creó en el 2010. En contraste, las colecciones para defectos o mantenimiento del software emplean información del código fuente y del repositorio, donde el 76% son de público acceso y sus últimas versiones oscilan en el intervalo 2006-2021.

Dicho esto, se decidió trabajar con el código fuente y metadatos del repositorio por la flexibilidad de acceso que brindan las plataformas para compartir código. Por otra parte, las métricas de código más estudiadas fueron principalmente de tamaño (por ejemplo, cantidad de líneas de código) y complejidad del código fuente (por ejemplo, complejidad ciclomática [68] o peso de métodos por clase [69]). El gráfico de burbujas en la Fig. 9 muestra la cantidad de colecciones del MS2 que emplearon métricas de tamaño y complejidad.

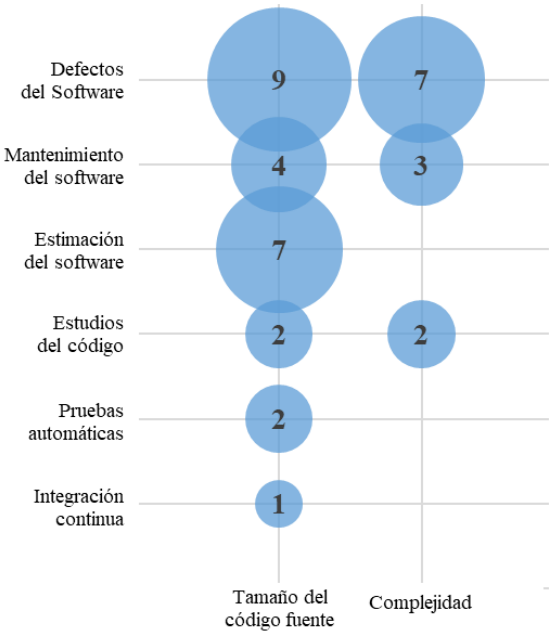

Fig. 9. Métricas de código por tipo de colección en el MS2

Finalmente, el nombre explícito o enlace a las herramientas empleadas para extraer los metadatos se reportaron en un 50% en el MS1 y en un 24% en el MS2. En muchas ocasiones se informa el modelo, técnica, procedimiento o algoritmo utilizado, más no la herramienta utilizada. En línea con lo informado por Cosentino et al. [22], si los instrumentos de extracción de datos no se difunden de manera adecuada se dificulta la replicación de los estudios y la validación de los hallazgos de la investigación. La Tabla 21 lista todas las herramientas reportadas que se encuentran disponibles y permiten calcular las métricas de código de interés en proyectos Java.

Tabla 21. Herramientas para calcular métricas de interés en proyectos Java

Herramienta	Enlace
Designite	https://www.designite-tools.com
iPlasma	http://loose.utt.ro/iplasma/iplasma.zip
Metrics	http://metrics2.sourceforge.net
SourceMeter	https://www.sourcemeter.com
Understand	https://www.scitools.com

En síntesis, los hallazgos confirman las inquietudes presentadas en el CAPITULO 1: ausencia de técnicas de muestreo probabilísticas, dificultad de acceso a los datos, reporte de herramientas y paquetes de

replicación limitado, desestimación de la vigencia de las muestras y falta de estrategias para actualizar los datos. De manera complementaria, se han determinado los lineamientos para evaluar proyectos Java de código abierto: 1) utilización de herramientas para dar soporte a procesos, 2) tamaño, 3) historia de desarrollo, 4) popularidad, 5) colaboración, 6) actividad reciente; las métricas de código a recopilar son las que cuantifican el tamaño y la complejidad del software; y las herramientas de la Tabla 21 proporcionan los medios para generarlas.

3.3.1. Selección de proyectos para las muestras

A continuación, se fundamenta la inclusión de los criterios de selección obtenidos en los estudios secundarios:

> **Colaboración.** Casi todos los proyectos software no triviales requieren esfuerzo y talento de múltiples personas trabajando juntas [70], pero la mayoría de los usuarios de GitHub utilizan la plataforma principalmente para sus propios proyectos y no con la intención de colaborar con otros [18]. El desarrollo de código abierto implica la interacción de desarrolladores dispersos globalmente contribuyendo código, conocimiento e ideas a lo largo del tiempo para lograr un conjunto de objetivos comunes [71]. Métricas como el número de colaboradores indican el éxito de un proyecto e influencian su ciclo de vida y crecimiento en el tiempo [72].
> Otras métricas relacionadas están asociadas con el modelo de desarrollo *pull-request* (PR), en el cual los colaboradores externos pueden proponer cambios a un proyecto sin necesidad de pertenecer al equipo principal [73]. En este esquema los colaboradores generan una copia independiente del repositorio (bifurcación o *fork* en inglés) y aplican cambios al código; cuando un conjunto de cambios está listo crean una PR; luego, un miembro del equipo principal del proyecto inspecciona los cambios y solicita más modificaciones, los rechaza (cierra o *close* en inglés) o los acepta (fusiona o *merge* en inglés) [74].
>
> **Historia.** Para mantenerse relevante, el software experimenta cambios continuos, como corrección de errores, incorporación de funcionalidades, mantenimiento preventivo o resolución de vulnerabilidades, entre otros [75]. Por lo tanto, tener una rica historia de desarrollo puede considerarse un indicador de calidad,

ya que sugiere que los miembros del proyecto han establecido procesos efectivos (como seguimiento de versiones y defectos, revisión colaborativa de código o gestión de equipos) para gestionar cambios a lo largo de un período prolongado. Un enfoque común para medir la historia de un proyecto es a través del número de confirmaciones (*commits* en inglés) [76 - 78], años de desarrollo activo [79, 80], frecuencia de confirmaciones [81, 82], entre otros.

Incidentes. Una práctica común en la Ingeniería del Software es que desarrolladores y usuarios finales informen los defectos que afectan el funcionamiento pretendido del software o las funcionalidades que se pueden mejorar. Estudios demostraron que esta práctica puede ser beneficiosa tanto para proyectos de código abierto como para proyectos cerrados [83], y la retroalimentación de los usuarios es ampliamente aceptada como información relevante en el desarrollo del software [84]. Los sistemas de seguimiento de incidentes buscan facilitar procesos como la gestión de requerimientos, cronogramas, tareas, defectos y versiones a través de informes de incidentes, sirviendo como un medio para la solicitud de mejoras, corrección de errores, incorporación de nuevas funcionalidades o mejora de la documentación del software.

Popularidad. Jarczyk et al. [85] definen que la reacción de la comunidad al proyecto es una medida adecuada de su calidad, y el repositorio que recibe una estrella en GitHub muestra la admiración y actitud positiva de la comunidad hacia este. Otra característica vinculada con la popularidad destacada por los autores es el número de veces que se realiza una bifurcación del proyecto. Borges & Valente [86] realizaron una encuesta y un MRS para validar las afirmaciones de Jarczyk, en el primer estudio, el 71% de los encuestados se mostraron de acuerdo con estas declaraciones, y en el segundo, descubrieron una relación positiva entre el número de estrellas y bifurcaciones de un repositorio. No obstante, se requiere precaución en la definición de los umbrales de estas métricas, Munaiah et al. [17] demostraron que implementar valores de referencia muy altos (500 a 1123 estrellas) podría excluir un gran conjunto de repositorios que contienen proyectos de calidad pero no son tan populares.

Tamaño. Aunque el tamaño por sí mismo no es un atributo de calidad, muchos estudios empíricos incluyen medidas relacionadas con el tamaño como criterios de selección de proyectos [76, 77, 87

- 90]. Algunas razones informadas son: evitar proyectos pequeños y de juguete (proyectos del tipo Hola Mundo), diversidad en el tamaño del código (elegir proyectos pequeños, medianos y grandes para construir una muestra más diversa y representativa), estudiar bases de código grandes, entre otros.

Actividad reciente. Para representar con precisión el estado actual de la industria del software, resulta intuitivo recopilar los datos de desarrollo más recientes en lugar de proyectos sin actividad. La presencia de cambios indica que el software se está modificando para garantizar su viabilidad [91]. Coelho & Valente [92] enumeran las razones comunes por las que un proyecto de código abierto podría dejar de ser mantenido: falta de tiempo o pérdida de interés del principal desarrollador, se basa en tecnologías desactualizadas, la aparición de un competidor más poderoso en el mercado o se convirtió en software obsoleto funcionalmente. Por lo tanto, muchos estudios empíricos consideran la actividad reciente como un criterio de selección [81, 93, 94].

3.3.2. Vigencia de una muestra

La Fig. 7 refleja como la mayoría de las colecciones de referencia nunca han sido actualizadas y sus últimas versiones datan de 1983 a 2016. Esto podría presentar inconvenientes en disciplinas como la ISE donde la población estudiada manifiesta cambios considerables en un periodo de tiempo relativamente corto. El momento cuando se adquieren los datos no es trivial porque como se mencionó en el CAPITULO 1, para poder generalizar los resultados de un estudio la muestra debe ser representativa de la población deseada. La vigencia es el grado en que los hallazgos o conclusiones basados en datos recopilados en un momento dado pueden generalizarse a la actualidad. En otras palabras, refiere a la medida en que una muestra representativa recopilada en el pasado aún representa a la misma población en el presente; por lo tanto, una muestra deja de ser vigente cuando la distribución de datos de las variables estudiadas ya no es representativa de la población objetivo.

Los proyectos software deben ser entendidos como el resultado de un proceso dinámico de evolución continua [91], e involucran tareas como la implementación de nuevas funcionalidades, corrección de errores, comprobación de funcionalidades, documentación, entre otras. De la misma manera, las tecnologías, entornos y bibliotecas utilizados para construir, probar y desplegar el software también evolucionan con el

tiempo, lo que provoca la actualización del proyecto para mantener su compatibilidad. En esta línea, Ait et al. [95] y Coelho et al. [96] reportaron que la probabilidad de un proyecto sobreviviendo más de cinco años es inferior al 50%. Esto sumado a la naturaleza evolutiva del software, destaca la importancia de monitorear los datos recopilados sobre pruebas, evaluación o análisis, para conservarlos vigentes y asegurar que sigan representando con precisión la población objetivo.

3.4. Estudio longitudinal

Para investigar el fenómeno de la vigencia en ISE, se llevó a cabo un estudio empírico y se evaluó si una población de proyectos de código abierto experimenta cambios significativos en el transcurso tiempo. En particular, se condujo un estudio longitudinal con 72 instantáneas de proyectos de Github y se analizó la evolución de siete métricas de repositorio: número de confirmaciones, colaboradores, PR cerradas, PR combinadas, incidentes resueltos, estrellas y bifurcaciones.

3.4.1. Planificación

La planificación del estudio comprende los procesos de recolección y análisis de datos. A continuación, se describen las PI que orientaron la investigación:

PI1. ¿El tiempo produce cambios en la población? Motivación: Se evaluaron siete métricas de repositorio en una muestra de 1991 proyectos Java tomados de Github durante un período de 6 años (desde julio del 2017 hasta junio del 2023). Se buscó identificar si el paso del tiempo conduce a diferencias significativas en las distribuciones de las métricas. En caso afirmativo, se detectó cuándo se manifestaron las diferencias, se cuantificó su magnitud y se determinaron las métricas más afectadas por el tiempo.

PI2. ¿Cuánto tiempo lleva que la población cambie? Motivación: De acuerdo con los resultados de la PI1 (y asumiendo que se detectaron cambios), para la PI2 se determinó la probabilidad de que la muestra sufra un cambio en la distribución de datos para las métricas analizadas. El objetivo es estimar la probabilidad de que una muestra continúe siendo representativa a medida que pasa el tiempo.

PI3. ¿Cuánto tiempo permanece activo un proyecto? Motivación: La duración de la actividad de los proyectos se determinó siguiendo la evolución del historial de actividad de 1991 repositorios. Además, se caracterizaron y compararon los datos de los repositorios de la última actividad registrada (último año).

3.4.1.1. Recolección de datos

Los datos utilizados para el estudio se presentan en la Tabla 22 junto con sus descripciones y motivaciones. Para responder a las PI, se extrajeron los metadatos D4 a D13. En particular, D1 y D2 se recopilaron para la PI3 con el fin de identificar los repositorios a lo largo del tiempo.

Tabla 22. Datos extraídos para el estudio longitudinal

Id.	Metadatos	Descripción	Motivación
D1	Id	Identificador global del repositorio	Identificar el repositorio en las olas.
D2	url	Url del repositorio	Acceder al repositorio
D3	name	Nombre del repositorio	Filtrado por palabras clave
D4	contributors	Número de colaboradores	Cuantificar la colaboración
D5	mergedPullReqCount	Número de PR cerradas	
D6	closedPullReqCount	Número de PR combinadas	
D7	commits	Número de confirmaciones	Cuantificar la historia
D8	closedIssuesCount	Número de incidentes resueltos	Cuantificar los incidentes
D9	stargazerCount	Número de estrellas	Cuantificar la popularidad
D10	forkCount	Número de bifurcaciones	
D11	dateLastCommit	Fecha de la última confirmación	Detectar actividad reciente
D12	closedPullReqLastDate	Fecha de la última PR cerrada	
D13	mergedPullReqLastDate	Fecha de la última PR combinada	
D14	monthlyCommits	Número de confirmaciones en el mes	Cuantificar la actividad mensual
D15	monthlyclosedPullReq	Número de PR cerradas en el mes	
D16	monthlymergedPullReq	Número de PR combinadas en el mes	

Los proyectos se obtuvieron de Github, y algunos repositorios se descartaron por no ser adecuados para la investigación en Ingeniería del Software [17]. El proceso de filtrado consistió en descartar los repositorios que no cumplieran con los umbrales de la Tabla 23, los cuales se basaron en los resultados de los mapeos sistemáticos descritos previamente en este capítulo y en las recomendaciones de Munaiah et al. [17], Kalliamvakou et al. [18] y Lewowski & Madeyski [97]. Se

excluyeron los repositorios no deseados que pudieran cumplir con los umbrales de calidad (por ejemplo, demos, tutoriales o repositorios de configuración) filtrando las siguientes palabras clave: *benchmark*, *conf*, *demo*, *docs*, *exam*, *sample*, *tutorial* y *wiki*.

Tabla 23. Criterios y umbrales de calidad del estudio longitudinal

Id.	Umbral	Criterio
U1	Lenguaje de programación Java	Proyectos Java
U2	Repositorios públicos	Metadatos disponibles
U3	Repositorios GIT	Uso de herramientas para dar soporte a procesos
U4	50 incidentes resueltos o más	
U5	Repositorios que no son bifurcaciones	Proyectos originales
U6	10000 *kilobytes* (KB) o más	Tamaño
U7	3 o más colaboradores	Colaboración
U8	50 PR o más	
U9	1 año o más desde su creación	Historia
U10	1000 confirmaciones o más	
U11	10 o más bifurcaciones	Popularidad
U12	10 o más estrellas	
U13	1 o más confirmaciones o PR cerradas/combinadas en el último mes	Actividad reciente

Una vez construido el instrumento de recolección, se aplicaron los umbrales U1 a U12 para recopilar una muestra de 2077 repositorios. Posteriormente, se recuperaron los datos históricos de estos repositorios con la rutina de Xia et al. [94],[5] que retorna archivos CSV con la actividad mensual de cada repositorio en términos de confirmaciones, colaboradores, PR, incidentes, estrellas y bifurcaciones. Se incluyeron en el conjunto de datos definitivo aquellos proyectos que cumplieran con el conjunto completo de umbrales de la Tabla 23 en alguna fecha de las instantáneas. Las instantáneas se generaron el primer día de cada mes (ola) desde el primero de julio del 2017 hasta el primero de junio del 2023 (seis años), lo que resultó en 72 instantáneas recuperadas. De los 2077 repositorios, 86 fueron descartados porque no aparecieron en ninguna instantánea, dando como resultado una muestra con información de 1991 repositorios. El conjunto de datos completo, que incluye los datos recopilados, rutinas y tablas suplementarias, se pueden acceder en el enlace.[6] La fecha y el número de repositorios obtenidos de cada instantánea se muestran en la Tabla 24.

[5] https://github.com/arennax/Health_Indicator_Prediction/
[6] https://github.com/juancarruthers/longitudinal_study

Tabla 24. Cantidad de repositorios incluidos en cada instantánea

	2017	2018	2019	2020	2021	2022	2023
Enero	-	588	724	852	1033	1187	1300
Febrero	-	637	768	888	1070	1243	1331
Marzo	-	630	755	893	1077	1202	1298
Abril	-	640	775	916	1094	1226	1341
Mayo	-	644	781	959	1106	1272	1312
Junio	-	655	802	964	1098	1271	1300
Julio	521	666	810	980	1120	1255	-
Agosto	532	678	830	985	1102	1288	-
Septiembre	545	699	826	980	1134	1269	-
Octubre	550	702	844	995	1128	1326	-
Noviembre	584	725	873	1053	1147	1308	-
Diciembre	594	725	869	1018	1158	1313	-

3.4.1.2. Análisis de datos

A continuación, se presentan los estadísticos descriptivos, inferenciales, métricas y gráficos empleados para analizar los datos obtenidos de la colección y responder las PI.

PI1. ¿El tiempo produce cambios en la población?

Se evaluaron las siete métricas en la Tabla 25 para detectar diferencias significativas entre las olas. Para resumir las distribuciones de las variables analizadas se calcularon las estadísticas descriptivas: mediana, desviación estándar, asimetría y curtosis. Además, se representaron gráficamente las funciones de densidad para visualizar las distribuciones de las variables y series temporales e interpretar la similitud entre las medianas en cada ola.

Se ejecutaron pruebas de hipótesis y medidas de tamaño del efecto para asegurar que los resultados fueran válidos desde una perspectiva estadística. Se definió un nivel de significancia de $\alpha = 0.05$ aplicando la corrección de Bonferroni [98] para las pruebas de hipótesis de cada variable. Por lo tanto, el umbral para rechazar las hipótesis nulas fue de 0.00714.

Tabla 25. Variables analizadas

Id	Metadato	Sigla	Criterio
V1	commits	COMM	Historia
V2	contributors	CONTR	Colaboración
V3	closedpullReqCount	CPR	
V4	mergedpullReqCount	MPR	
V5	closedIssuesCount	CI	Incidentes
V6	stargazerCount	STARS	Popularidad
V7	forkCount	FORKS	

Para identificar el tipo de pruebas de hipótesis más adecuada para las distribuciones, se aplicó la prueba de normalidad Shapiro-Wilk [99]. En todas las variables se rechazó la hipótesis nula, por lo tanto, se optó por pruebas no paramétricas. Dicho esto, se condujo la prueba Kruskal-Wallis H [100], donde su hipótesis nula establece que todos los grupos tienen la misma mediana. La prueba se aplica a varios grupos simultáneamente, pero no puede identificar exactamente dónde y cuánto difieren estadísticamente los grupos. En los casos donde se rechazó la hipótesis nula, es decir, las medianas de los grupos presentaron diferencias significativas, se calculó la prueba *post hoc* de Dunn [101] para identificar los pares de grupos diferentes.

La diferencia entre dos grupos se calculó con la métrica de tamaño del efecto Vargha-Delaney [102]. Si dos grupos son estadísticamente indistinguibles $\hat{A}_{12} = 0.5$, si el primer grupo obtiene valores promedio más altos que el segundo $\hat{A}_{12} > 0.5$, y si el segundo grupo obtiene valores mayores $\hat{A}_{12} < 0.5$. Las medidas se resumieron en cuatro categorías nominales basadas en $\hat{A}_{12}$, definido como $\hat{A}_{12}^{(escalado)} = (\hat{A}_{12} - 0.5) * 2$ [103]: marginal ($|\hat{A}_{12}^{(escalado)}| < 0.147$), pequeño ($0.147 \leq |\hat{A}_{12}^{(escalado)}| < 0.33$), mediano ($0.33 \leq |\hat{A}_{12}^{(escalado)}| < 0.474$) y grande ($|\hat{A}_{12}^{(escalado)}| \geq 0.474$).

PI2. ¿Cuánto tiempo lleva que la población cambie?

Se tomaron los resultados de la prueba de Dunn de la PI1 y se identificaron los grupos o instantáneas en las cuales se rechazó la hipótesis nula en cualquiera de las variables, y luego se aplicó un algoritmo de análisis de supervivencia [104]. El análisis de supervivencia es una técnica utilizada en varios estudios de Ingeniería del Software [81, 105, 106] para estimar la tasa de supervivencia de una población dada, es decir, el tiempo transcurrido hasta que ocurra un evento específico.

El tiempo de partida fue la fecha de recuperación de la instantánea. El evento considerado para el análisis fue: la instantánea no es representativa, refiriendo al momento cuando se detectó una diferencia significativa entre una instantánea y otra instantánea posterior. Según Baltes & Ralph [21], la representatividad depende de la dimensión analizada, una muestra puede ser representativa en una variable, pero no en otras. Entonces, se generaron dos tipos de curvas de supervivencia: 1) una general que registró un evento cada vez que se rechazó la prueba de Dunn para cualquier variable y 2) una dimensional

que analizó cada variable individualmente. Se calculó el tiempo de duración de las instantáneas desde el tiempo de partida hasta la ocurrencia del evento. Luego se empleó el estimador no paramétrico Kaplan-Meier [107] para modelar la curva de supervivencia.

PI3. ¿Cuánto tiempo permanece activo un proyecto?

Se inspeccionó el historial de actividad de los 1991 repositorios y se aplicó nuevamente el análisis de supervivencia para analizar la probabilidad de que un proyecto se actualice. El tiempo de partida fue la fecha de creación del repositorio y el evento fue: el proyecto no muestra signos de actividad durante 30 días o más. Un signo de actividad refiere a un repositorio recibiendo una confirmación, una PR cerrada o combinada; entonces, un proyecto se consideró inactivo si no registró confirmaciones o PR durante un período de 30 días o más. De esta manera, la duración del evento se calculó como la diferencia entre la fecha de creación del proyecto y la fecha inmediatamente anterior a detectar la inactividad. Por ejemplo, si se creó un repositorio el 11-09-2018 y este registró actividad al menos cada 30 días hasta el 06-01-2020, se registró una duración de 482 días. La curva de supervivencia también se modeló con el método de Kaplan-Meier. Se incluyó un gráfico de barras con los proyectos agrupados según su duración.

Además, se investigaron las características de los proyectos con actividad mensual más reciente. Se calculó la actividad mensual sumando las confirmaciones, PR cerradas y combinadas por mes, y posteriormente el promedio desde el 1 de julio del 2022 hasta el 1 de junio del 2023. Luego, se agruparon los proyectos por su actividad mensual promedio en dos categorías: actividad baja (primer cuartil) y actividad alta (cuarto cuartil). Finalmente, se graficaron los grupos en diagramas de caja.

3.4.2. Resultados

A continuación, se responde a las PI en base a los resultados obtenidos del análisis de los datos.

3.4.2.1. PI1. ¿El tiempo produce cambios en la población?

Inicialmente, se examinaron las estadísticas descriptivas de la Tabla 26 para evaluar la normalidad de las variables. Típicamente, una

distribución normal tiene valores de asimetría y curtosis de cero y tres respectivamente, independientemente de la desviación estándar. Por lo tanto, todas las variables parecen tener una distribución no normal. Todas las distribuciones tuvieron desviaciones considerables indicado por los altos valores de curtosis y asimetría. Las funciones de densidad en la
Fig. **10** confirman que las variables no son normales.

Tabla 26. Estadísticas descriptivas de las variables

Variables	Mediana	Desviación Estándar	Asimetría	Curtosis
V1	2972	8787.77	5.65	48.37
V2	62	182.75	7.16	77.79
V3	69	531.53	9.67	126.66
V4	332	1746.08	11.61	251.28
V5	478	1734.86	6.72	68.11
V6	413	4888.3	4.78	27.14
V7	153	1912.59	8.85	107.43

La
Fig. **11** presenta una serie temporal para ilustrar la evolución de las medianas de las variables en cada semestre, donde el primer semestre abarca las instantáneas tomadas entre julio de 2017 y diciembre de 2017, el segundo entre enero del 2018 y junio del 2018, y así sucesivamente.

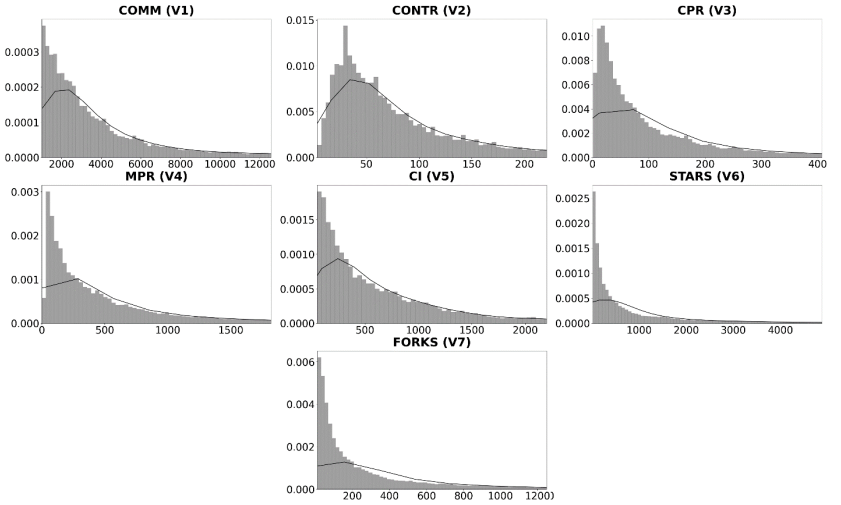

Fig. 10. Funciones de densidad de las variables de interés

Todas las variables aumentaron sus medianas con el paso del tiempo. Proporcionalmente, V7 fue la de menor incremento, con un valor

mínimo de 134 en el primer semestre y un valor máximo de 163 en el décimo segundo. En contraste, V4 tuvo el mayor aumento, comenzando con un valor mediano de 204 y terminando el décimo segundo semestre con 456.

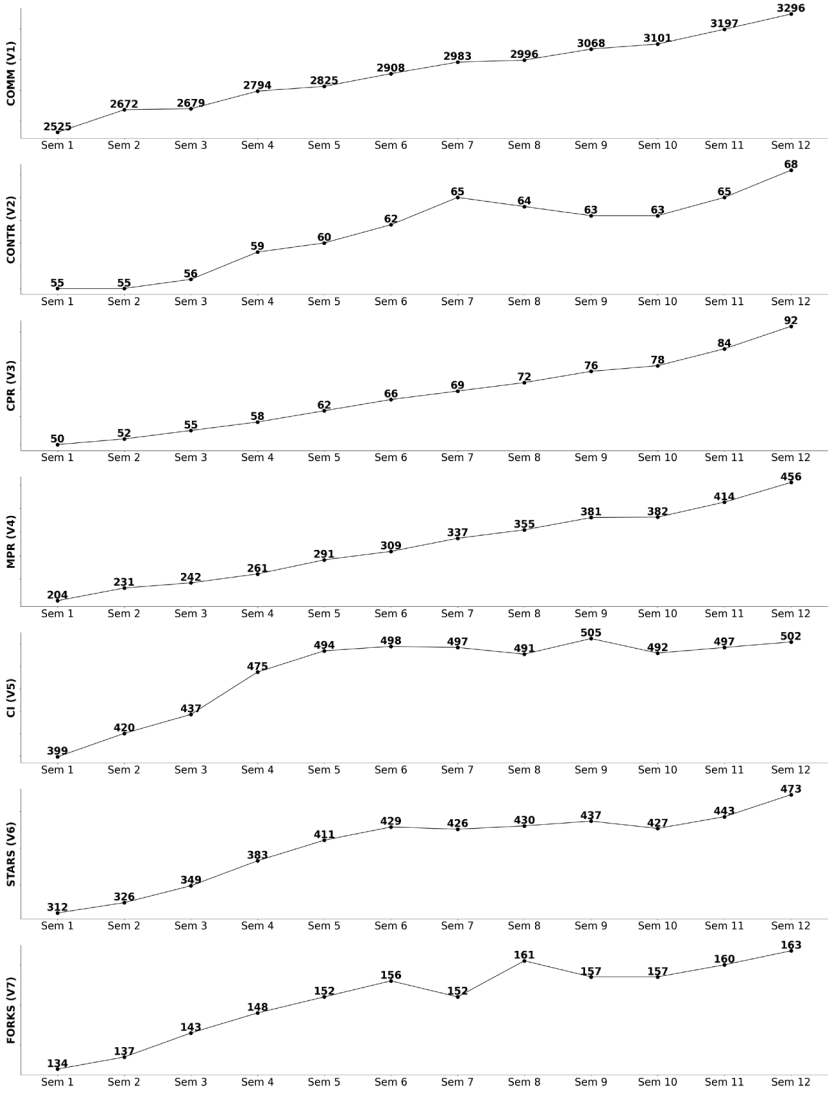

Fig. 11. Evolución de las medianas en las olas

Posteriormente, se procedió con las estadísticas inferenciales realizando las pruebas de Shapiro-Wilk y Kruskal-Wallis H, como se muestra en la Tabla 27. Los p-valores de la prueba de Shapiro-Wilk fueron de cero

en todas las variables y por ello se rechazaron las hipótesis nulas, confirmando la presunta no normalidad de las variables. Por otra parte, la prueba Kruskal-Wallis H reveló diferencias significativas en todas las variables, con p-valores por debajo del nivel de significación 0.00714.

Tabla 27. Resultados de pruebas estadísticas Shapiro-Wilk y Kruskal-Wallis

Variables	Shapiro-Wilk	Shapiro-Wilk (p-valor)	Distribución Normal	Kruskal-Wallis H	Kruskal-Wallis H (p-valor)
V1	0.48	0	No	347.74	0
V2	0.44	0	No	346.56	0
V3	0.3	0	No	1306.82	0
V4	0.38	0	No	1778.99	0
V5	0.46	0	No	215.62	0
V6	0.42	0	No	202.37	0
V7	0.3	0	No	130.29	0

Se realizó la prueba de Dunn de todas las variables para identificar las olas estadísticamente diferentes. Luego, se emparejaron las olas con un p-valor por debajo de 0.00714 y se calculó el tamaño del efecto Vargha-Delaney para medir la magnitud de la diferencia. Las tablas completas con los resultados de las pruebas también están en el conjunto de datos compartido.[6]

Comenzando con los resultados de la prueba de Dunn; las variables V1, V2 y V3 rechazaron la mayoría de las hipótesis nulas al comparar las primeras olas (desde julio hasta octubre del 2017) y la número 30 en adelante (desde diciembre del 2019), a partir de ese punto la quinta ola (noviembre 2017) registró diferencias significativas con la número 38 en adelante (desde agosto del 2020), la sexta con la número 39 en adelante y así sucesivamente. De manera similar, V6 registró su primera diferencia entre las olas número 5 (noviembre de 2017) y 31 (enero de 2020). V5 rechazó la hipótesis en menos tiempo, en las olas número 1 y 22 (abril de 2019). V4 fue la variable con más diferencias, registrando su primera entre las olas número 1 y 11 (mayo de 2018). En contraste, V7 mostró la menor cantidad de diferencias, comenzando en la ola número 6 (diciembre de 2017) y 64 (noviembre de 2022). Luego, con la puntuación escalada de Vargha-Delaney, todas las variables tuvieron tamaños de efecto desde marginales hasta pequeños. En particular, la V4 en las olas número 1 y 65 (diciembre de 2022) retornó un tamaño del efecto mediano.

3.4.2.2. PI2. ¿Cuánto tiempo lleva que la población cambie?

Partiendo de los resultados de la prueba de Dunn, se generó una curva de supervivencia para analizar la representatividad de las instantáneas a lo largo del tiempo. La curva de supervivencia representa la probabilidad de que una instantánea se mantenga representativa desde el momento en que se recopiló hasta que su representatividad desaparece. La Fig. 12 muestra la probabilidad de representatividad general de las instantáneas recuperadas de Github, que considera todas las variables. Durante los primeros 273 días la probabilidad de que una instantánea sea representativa fue del 100%. Después de un año permaneció alta, con una probabilidad del 82.3%. Sin embargo, al pasar 456 días, la probabilidad de representatividad bajó al 51.6%. Después de 516 días, resulta muy improbable que la instantánea sea representativa (24.2%). Se alcanzó la probabilidad de cero después de 579 días. En particular, la instantánea que permaneció representativa durante más tiempo fue la décima (mayo de 2018), con 579 días. En contraste, las instantáneas 59 y 61 tuvieron la duración más corta, cada una permaneciendo representativa por 273 días.

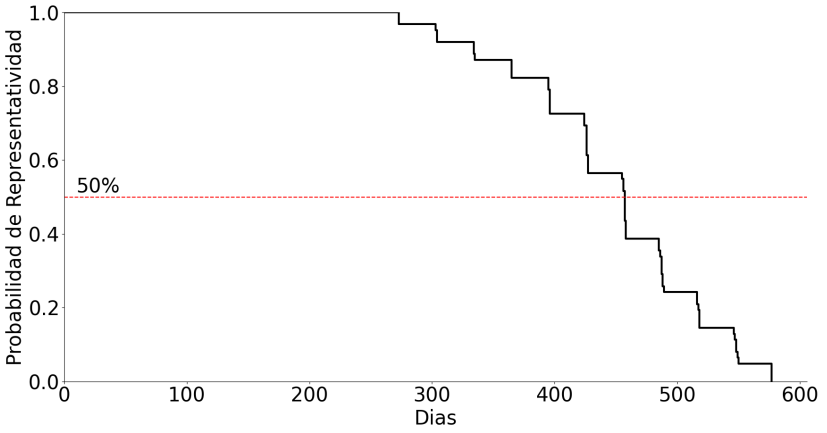

Fig. 12. Análisis de supervivencia de la representatividad general

Por otro lado, en la Fig. 13 se analizó cada variable individualmente y se detectaron diferencias en cuanto a la duración de su vigencia. Por ejemplo, V1 y V2 tuvieron una probabilidad del 100% durante más de un año (577 y 669 días), permanecieron por encima del 50% durante tres años (1096 y 1005) y llegaron a cero antes de que finalizara el cuarto año (1400 y 1280). De manera similar, V5 y V6 también tuvieron más de un año con el 100%, sin embargo, registraron un valor por encima del 50% durante más de cuatro años. En cambio, V3 y V4 permanecieron vigentes durante menos tiempo, después del segundo

año registraron un valor por debajo del 50% y llegaron a cero después de 821 y 638 días respectivamente. En particular, V5 fue la única dimensión que no registró una probabilidad por debajo del 25%.

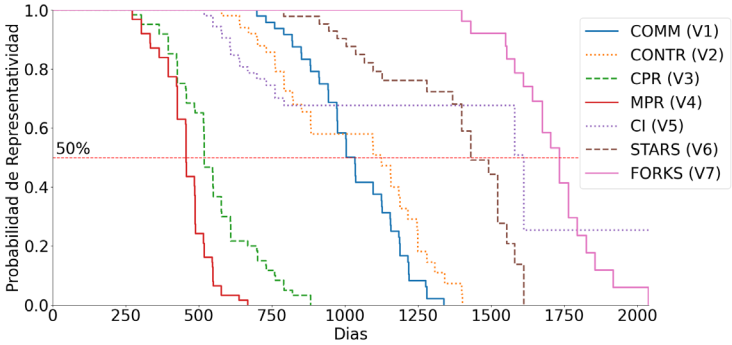

Fig. 13. Análisis de supervivencia de la representatividad dimensional

3.4.2.3. PI3. ¿Cuánto tiempo permanece activo un proyecto?

La Fig. 14 muestra la gráfica de supervivencia de los 1991 repositorios recuperados. La curva de supervivencia comenzó en 0.91 porque 177 repositorios no recibieron una confirmación, PR cerrado o fusionado durante más de 30 días desde su creación. La probabilidad de que un repositorio estuviera activo durante los dos primeros años fue del 65% y del 54% respectivamente. Después de 30 meses, la probabilidad disminuyó al 50%, y después de 161 meses disminuyó al 14%. Repositorios como apache/jmeter,[7] elastic/elasticsearch[8] y spring-projects/spring-data-commons,[9] junto con otros siete repositorios, mantuvieron un estado activo ininterrumpido durante más de 150 meses.

[7] https://github.com/apache/jmeter
[8] https://github.com/elastic/elasticsearch
[9] https://github.com/spring-projects/spring-data-commons

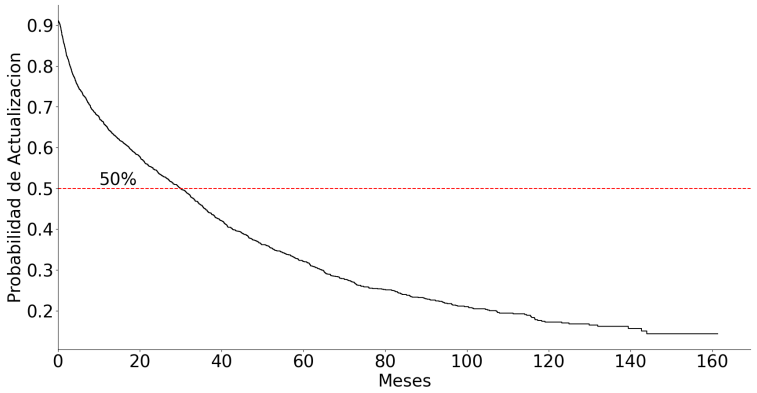

Fig. 14. Análisis de supervivencia de la actualización de un repositorio

La Fig. 15 muestra un gráfico de barras con el número de proyectos que mostraron signos de inactividad cada año desde su creación. El gráfico reveló que en el primer año de desarrollo el 35% de los proyectos tuvieron al menos un mes de inactividad, mientras que en el segundo año este número disminuyó al 11%. La cantidad de proyectos que registraron eventos de inactividad decrecieron constantemente a lo largo de los años, con un promedio de casi 118 proyectos entre el tercer y el sexto año. Sin embargo, después del séptimo año de desarrollo, el número de proyectos alcanzó el 1% promedio por año.

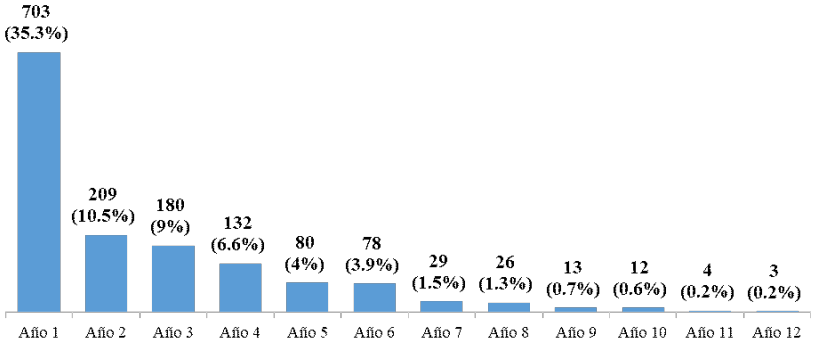

Fig. 15. Cantidad de repositorios agrupados por el primer indicio de inactividad

Posteriormente, se investigó la actividad más reciente de estos repositorios, que abarcó desde el primero de julio del 2022 hasta el primero de junio del 2023. Se clasificaron según su actividad mensual en dos grupos: actividad baja (primer cuartil) y actividad alta (cuarto cuartil), y en la Fig. 16 se graficaron diagramas de caja para cada

variable con los dos grupos. En todos los casos, los valores del grupo con actividad alta (C4) fueron mayores en comparación con los del grupo con actividad baja (C1). El último gráfico muestra las distribuciones de los mismos grupos por la edad del proyecto (es decir, el período de tiempo desde que se creó el repositorio hasta que se registró la última actividad) en meses. A diferencia de las variables V1 a V7, los números de C1 fueron mayores que los de C4.

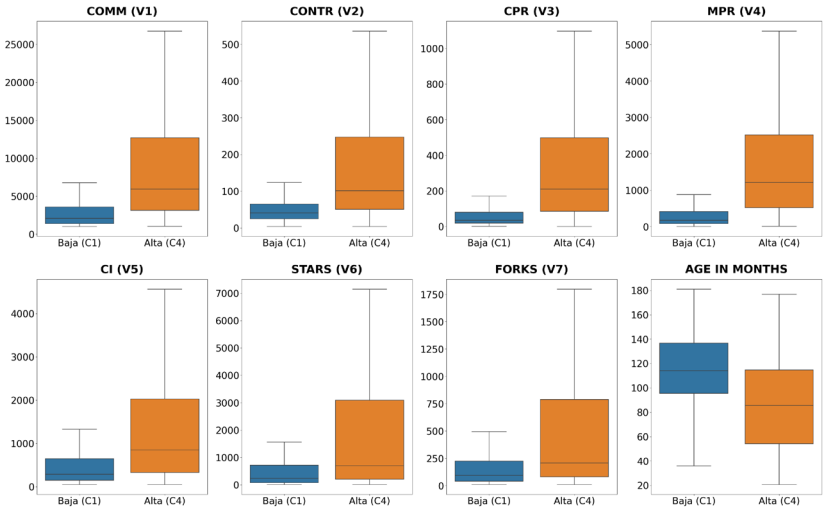

Fig. 16. Repositorios agrupados por probabilidad de actualización

3.4.3. Conclusiones

A continuación, se ofrece una interpretación de los resultados reportados discutiéndolos en el contexto de las PI y la literatura relacionada. En este sentido, se realiza una reflexión en torno a la vigencia de las muestras en la ISE y la actualización de los proyectos software.

3.4.3.1. Vigencia de las muestras

Se evaluó si una población de proyectos de código abierto experimenta cambios en las distribuciones de la cantidad de colaboradores, confirmaciones, PR, incidentes, estrellas y bifurcaciones a lo largo de seis años. A primera vista, el análisis estadístico de las instantáneas realizado con la prueba de Kruskal-Wallis reveló diferencias significativas en todas las variables, lo que significa que las

distribuciones de las variables pueden cambiar con el tiempo. Estas diferencias muestran cómo una muestra puede perder representatividad si no se actualiza a tiempo. Además, la Fig. 12 muestra cómo estos cambios son aparentes en un período de tiempo relativamente corto con una probabilidad de representatividad del 0% después de 579 días. Aunque estos cambios pueden explicarse por el crecimiento de sus valores medianos, como se muestra en la Fig. 11, estas no evolucionaron de manera uniforme.

El crecimiento detectado en las métricas de colaboración, como el número de colaboradores y PR, refleja la expansión del desarrollo de software distribuido facilitado por las plataformas colaborativas como Github. En particular, el impacto más significativo se observó en las métricas de PR, donde las PR combinadas y las cerradas tuvieron un aumento promedio anual del 20,5% y 14% respectivamente. Gousios et al. [74, 108] informaron sobre estos cambios de tendencia desde formas clásicas de contribuir, por ejemplo, en el pasado los conjuntos de cambios se enviaban a listas de correo de desarrollo y ahora se encuentran los sistemas de seguimiento de incidentes, o bien, antes se requería acceso directo al sistema de control de versiones y en la actualidad existe el modelo basado en PR. La inclusión de contribuciones externas al equipo central de desarrollo ciertamente aumentó el número de colaboradores a lo largo de los años.

En cuanto al número de confirmaciones, se puede explicar por la evolución natural de los proyectos software a lo largo del tiempo. Este fenómeno se alinea con las leyes de Lehman [109], que describen la progresión del software y su tendencia a aumentar en complejidad con el tiempo. Estudios recientes corroboraron estas leyes [110 - 112], destacando que la cantidad de confirmaciones y archivos de código fuente en el código abierto se duplica aproximadamente cada 30 y 22 meses respectivamente.

Con respecto a variables asociadas a la popularidad del proyecto, como el número de bifurcaciones y estrellas, se detectó un crecimiento constante año tras año. Borges & Valente [86] estudiaron la popularidad en términos de estrellas de proyectos y observaron que los repositorios tienden a recibir más estrellas después de su lanzamiento público. Sin embargo, después de este aumento inicial la tasa de crecimiento se estabiliza.

Por el contrario, el número de incidentes resueltos no registró un aumento constante en todo el período considerado, de hecho, ninguna

de las instantáneas recuperadas después del primero de enero del 2019 registró diferencias significativas con las posteriores. De manera similar, Cosentino et al. [113] revisaron el uso del sistema de seguimiento de incidentes de Github e informaron que los incidentes son generalmente estables o muestran una ligera tendencia negativa con el tiempo.

Aunque la evolución no se manifiesta de manera uniforme en todas las variables y la vigencia de una muestra depende de los datos específicos estudiados, emplear conjuntos de datos creados hace varios años no parece ser una estrategia viable si el objetivo es producir resultados generalizables. Después de cinco años, todas las variables mostraron probabilidades de representatividad por debajo del 25%.

3.4.3.2. Actividad de los proyectos

En la PI3 se analizó la probabilidad de que un proyecto se actualice con el tiempo utilizando una muestra de 1991 sistemas Java. Las curvas de supervivencia representadas en la Fig. 14 revelaron una probabilidad de actualización del 50% después de 30 meses, con progresivas disminuciones a 45%, 37% y 32% en el tercer, cuarto y quinto año respectivamente. En comparación con Ait et al. [95], ellos muestran una probabilidad del 50% a partir del tercer año, basando sus resultados en 1127 proyectos PHP y R. La disparidad se puede atribuir en mayor medida a la condición de supervivencia utilizada para identificar sistemas activos: "está vivo si ha mostrado actividad de cualquier tipo en los últimos seis meses", y en menor medida a la elección de los lenguajes de programación.

Asimismo, se clasificaron los proyectos según su actividad más reciente (desde el primero de julio del 2022 hasta el primero de junio del 2023), y los diagramas de cajas en la Fig. 15 indicaron que los proyectos con actividad alta (C4) tuvieron más colaboradores, confirmaciones, PR, incidentes, estrellas y bifurcaciones que los de C1. Estos resultados están en línea con estudios longitudinales anteriores. Por ejemplo, Coelho et al. [96] informaron sobre la disparidad entre proyectos con mayor y menor supervivencia en GitHub, examinando específicamente el número total de colaboradores, incidentes y PR. Los autores destacaron cómo estas variables podrían tener una correlación con la supervivencia de proyectos de código abierto. No obstante, tener un gran número absoluto de colaboradores, confirmaciones, incidentes o PR no garantiza que un repositorio mantenga su actividad en el futuro.

Por ejemplo, ControlSystemStudio/cs-studio[10] es una colección de herramientas para monitorear y operar sistemas de control a gran escala creada en 2012 con más de 100 colaboradores, 27 mil confirmaciones y mil incidentes resueltos, y el año pasado registró solo cuatro confirmaciones. Cuba-platform/cuba[11] es un marco de desarrollo para aplicaciones empresariales que data del 2016 y tiene más de 100 colaboradores, 14 mil confirmaciones y 2 mil incidentes resueltos; sin embargo, el año pasado recibió solo 10 confirmaciones. Facebook/buck[12] es un sistema de construcción creado en 2013 con más de 400 colaboradores, 22 mil confirmaciones y mil incidentes resueltos, y su última confirmación fue en abril del 2023.

En contraste, al considerar la edad, los proyectos con actividad alta son más jóvenes que los de menor actividad, con edades medianas de 80 y 108 meses, respectivamente. Esto se relaciona con las conclusiones de Xia et al. [94] y Coelho et al. [96] donde declararon que los proyectos más antiguos tienden a experimentar una reducción de actividad, y los más jóvenes al estar en una etapa de desarrollo activo atraen más atención de los programadores.

En resumen, los hallazgos sugieren que: 1) es poco probable que un proyecto tenga actividad mensual ininterrumpida durante un período prolongado (más de cuatro años), y 2) los proyectos activos tienden a ser más jóvenes y atraen una mayor participación.

Antes de finalizar el capítulo, en la Fig. 17 se muestran las actividades de CD siendo las tareas de color verde las abordadas hasta ahora y las grises aquellas a desarrollar en los siguientes capítulos.

[10] https://github.com/ControlSystemStudio/cs-studio
[11] https://github.com/Cuba-platform/cuba
[12] https://github.com/Facebook/buck

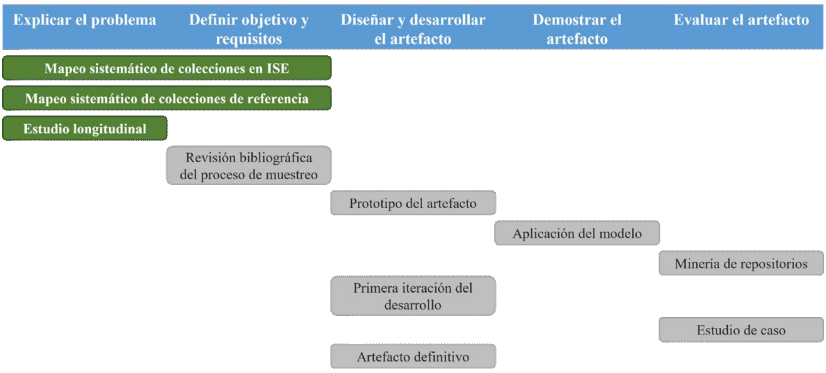

Fig. 17. Actividades abarcadas en el CAPITULO 3

CAPITULO 4

Desarrollo del artefacto

Este capítulo aborda la construcción de la versión final del artefacto propuesto y la contribución principal de esta tesis: un modelo de proceso para la construcción y actualización de muestras de proyectos software orientadas a estudios empíricos en la Ingeniería del Software. Esto incluye su especificación y utilización en un escenario de aplicación para exponer su funcionamiento.

El capítulo se organiza de la siguiente manera: La sección 4.1 detalla las actividades consideradas para la selección y actualización de muestras, la sección 4.2 presenta el modelo de proceso propuesto describiendo sus actividades, artefactos y roles, la sección 4.3 describe de manera detallada la aplicación del modelo en un contexto específico.

4.1. Identificación de tareas del modelo de proceso

Tal como fue indicado al final de la sección 1.1, y teniendo en cuenta la definición de Marbán et al. [30] un modelo de proceso es un conjunto de pasos de proceso o actividades para llegar a un objetivo determinado. A su vez estas actividades reciben las entradas necesarias para su ejecución y producen salidas. Por lo tanto, es necesario detallar cuáles son las actividades involucradas en los procesos modelados, lo que implica la especificación de sus entradas, salidas, orden de ejecución, y actores que las desempeñan, entre otras cuestiones. A continuación, se describen las actividades involucradas en los dos procesos abordados por el modelo: el muestreo y la actualización de muestras de proyectos software.

4.1.1. Muestreo de proyectos software

Para la definición de las actividades orientadas al muestreo de proyectos, en primera instancia, se estudiaron las guías de muestreo en Ingeniería del Software relevadas por Baltes & Ralph [21] (ver la Tabla 52 del APENDICE C con los enfoques de referencia). Dentro de las mismas se encuentran los libros de Cochran [114] y Henry [115], y trabajos orientados al muestreo en la Ingeniería del Software [38, 116, 117]. En su mayoría, las guías citadas están dirigidas a la construcción de muestras para encuestas con la excepción de Vidoni [38], que emplea su protocolo en la recolección de proyectos para la MRS. A continuación, se presenta una breve discusión del listado de las tareas propuestas por los autores:

Definir los objetivos de la investigación. Primero se deben establecer los objetivos de la investigación para planificar la selección de la muestra. Todas las guías incluyen esta actividad para orientar el estudio. En el caso de Vidoni [38], se enumeran más ítems para caracterizar el tipo estudio, como la definición de los focos de interés, las preguntas de investigación, y qué se observa, entre otros.

Definir la población. La población es el grupo al que se aplica el estudio realizado, es decir, es el grupo observado por los investigadores de donde se quiere obtener información. Todos los autores consideran esta actividad, y en su mayoría refieren a un grupo de participantes humanos del cual se obtendrán los resultados del estudio (encuesta, experimento, estudio de caso, entre otros). En particular, Vidoni [38] describe la población como un grupo de proyectos software que serán seleccionados y empleados en el MRS.

Determinar los datos a obtener. Consiste en determinar los datos relevantes para los propósitos de la investigación. Cochran [114] considera esta actividad para no omitir datos esenciales y evitar que se hagan muchas preguntas a los encuestados que posteriormente no se analizan. Vidoni [38] también analiza los tipos de datos y su relevancia en el estudio para definir los sitios online o fuentes que pueden proporcionarlos.

Establecer la precisión deseada. Para emplear los instrumentos de medición adecuados y reducir los errores de medición, se establece con anterioridad la precisión requerida. Solo Cochran [114] contempla esta actividad, dado que en algunos casos utilizar

instrumentos de alta precisión para recolectar datos en encuestas resulta costoso.

Definir los instrumentos de recolección. Se describen los métodos e instrumentos mediante los cuales se recolectarán los datos. Cochran [114] lo define como la forma para obtener los datos de las encuestas (cuestionarios o entrevistas) y el medio (por correo, teléfono, visita personal o una combinación de estos), teniendo en cuenta el diseño de los formularios de registro. Kitchenham & Pfleeger [117] mencionan la construcción, evaluación y documentación de los cuestionarios, abordando el diseño de las preguntas, las respuestas esperadas, el formato, entre otras cuestiones. Vidoni [38] alude a la creación de algoritmos y herramientas informáticas para la recolección de los datos. Por su parte, Henry [115] sugiere que la estrategia de muestreo y la estrategia de recolección de datos deben ser pensadas en conjunto porque pueden afectar la confiabilidad del instrumento.

Seleccionar la muestra. La muestra es un subconjunto de la población seleccionada conscientemente considerando las restricciones de recursos. Para operacionalizar este proceso se recurre al marco muestral de la población objetivo, es decir, la lista de sujetos de la población que el investigador tiene un alcance real. Esta actividad es considerada por todos los autores. En particular Kitchenham & Pfleeger [117], mencionan la importancia que esta sea representativa de la población objetivo para que los resultados puedan generalizarse. Para ello, se deben definir la técnica de selección (preferentemente, probabilística) y el tamaño de la muestra a través de un método estadístico apropiado.

Organizar el trabajo de campo. En encuestas extensivas, el personal debe recibir entrenamiento en el propósito del estudio y en los métodos de medición a emplear, además de ser supervisado en su trabajo. Es necesario un mecanismo de verificación temprana de la calidad de las respuestas recolectadas.

Realizar prueba piloto. En las encuestas es útil probar el cuestionario y los métodos de campo en una escala pequeña, esto puede revelar errores en el cuestionario y otros problemas que podrían ser graves a gran escala; resultando en mejoras en el instrumento final.

Registrar el proceso de selección y evaluación. Se registran los pasos realizados para la selección de la muestra, detallando la cantidad de elementos eliminados e incluidos en cada etapa del muestreo. Vidoni [38] busca informar los resultados de la

ejecución de cada etapa y los criterios para evaluar la calidad de la muestra, y además revisar los proyectos software recuperados.

Extraer y transformar los datos. Se reporta la ejecución de los instrumentos de recolección y transformación de datos. Dado que se busca un formato de salida uniforme, en algunos casos resulta necesario realizar una transformación de los datos. La salida obtenida debe ser revisada para alertar si existen errores en los datos capturados o en el formato de salida.

Resumir y analizar los datos obtenidos. Una vez extraídos y organizados los datos, los investigadores deben iniciar la generación de información relevante. En el caso de Cochran [114], se utiliza para enmendar los errores detectados en la captura de los datos y posteriormente estimar las respuestas a las preguntas planteadas en la encuesta. Se deben elegir y documentar los métodos empleados para analizar datos y comprender las complejas relaciones entre los resultados obtenidos.

Presentar los resultados. Los resultados obtenidos deben visualizarse de forma que facilite el análisis. El formato está limitado por el tipo de información extraída, ya que dependiendo del tipo de gráfico (por ejemplo, gráficos, tablas, cuadros específicos) se puede adaptar mejor a diferentes tipos de información.

Utilizar la información obtenida para futuros estudios. Una muestra completa es potencialmente una guía para mejorar un muestreo futuro. En una encuesta compleja lo planificado no siempre sucede exactamente como se espera. El encargado de realizar la muestra aprende a reconocer errores en la ejecución y prevenir que no ocurran en encuestas futuras.

En la Tabla 28 se listan las actividades incluidas de los modelos de referencia con su respectiva motivación.

Tabla 28. Tareas del muestreo

Tareas	Incluida	Motivación
Definir los objetivos de la investigación.	No	Esta tarea está vinculada a la planificación del estudio, e influencia el proceso de muestreo, pero en este caso, el objetivo es definir el protocolo de muestreo no una caracterización descriptiva del estudio a conducir.

Tabla 28. Tareas del muestreo

Tareas	Incluida	Motivación
Definir la población.	Si	Implica la caracterización de la población de proyectos software, es decir, el grupo de sujetos que serán seleccionados y de los cuales se extraerán los datos específicos.
Determinar los datos a obtener.	Si	Para orientar el proceso y determinar los sitios online o fuentes de donde se consumirán los datos, primero se requiere conocer los datos en cuestión.
Establecer la precisión deseada.	No	Esta tarea refiere a la precisión de los instrumentos específicos para las encuestas, por lo que no tiene relevancia de aplicación en el muestreo de proyectos software.
Definir los instrumentos de recolección.	Si	Describe la creación de algoritmos y herramientas informáticas para la recolección de los datos.
Seleccionar la muestra.	Si	Especifica la estrategia de muestreo para seleccionar los proyectos software para construir una colección de la población que se busca estudiar.
Organizar el trabajo de campo.	No	La organización del trabajo de campo, en el contexto de encuestas, busca capacitar al personal que adquiere las respuestas de los participantes y así reunir muestras válidas.
Realizar prueba piloto.	Si	Realizar pruebas preliminares del instrumento de recolección proporciona información útil para calibrarlo o solucionar problemas no identificados durante la planificación.
Registrar el proceso de selección y evaluación.	Si	Dado que es necesario documentar todo el proceso de muestreo para su posterior auditoría, esta parte es llevada a cabo en todas las actividades del proceso.
Extraer y transformar los datos.	Si	Para adquirir los datos requeridos se debe ejecutar la extracción y normalización para simplificar el uso de la colección.
Resumir y analizar los datos obtenidos. Presentar los resultados. Utilizar la información obtenida para futuros estudios.	No	No se desalienta el análisis de los datos como el resumen, presentación gráfica o la utilización de información obtenida en experiencias futuras, pero excede al tipo de proceso que se busca desarrollar.

De las actividades anteriormente descriptas resultan de interés solamente aquellas que puedan contribuir al proceso de construcción de

muestras y tengan sentido de aplicación en la selección de proyectos software.

4.1.2. Actualización de la muestra

Una vez construida la muestra de proyectos, esta debe ser conservada con información válida para reflejar de manera fiel la población objetivo. Los cambios constantes que sufren los proyectos software requieren un monitoreo frecuente del estado de la muestra para mitigar la perdida de vigencia tal y como se describió en la sección 3.4.3. En este sentido, de la misma manera que para el proceso de muestreo, se revisó la literatura para registrar las estrategias implementadas al respecto.

A diferencia de las guías de muestreo analizadas en el APENDICE C, no se encontraron estrategias de referencia para realizar la actualización de muestras en el contexto de la Ingeniería del Software; no obstante, existen aproximaciones *ad hoc* propuestas por autores de las colecciones estudiadas en el MS2 de la sección 3.2. Por ejemplo, Tempero et al. [8] definieron una serie de lineamientos para nuevos lanzamientos de la colección: 1) proyectos en ediciones anteriores de la colección, 2) desarrollos en Java, 3) disponibilidad del código fuente y compilado, 4) binarios distribuidos como archivos Jar, 5) acceso público a los proyectos y 6) estructura del proyecto organizada. Le Goues [118] brindaron las siguientes reglas para la inclusión de nuevos proyectos a la colección ManyBugs: 1) como mínimo 50000 líneas de código escrito en lenguaje C, 2) 10 casos de prueba viables y 3) 300 o más confirmaciones en el sistema de control de versiones; instando a la comunidad a actualizar el conjunto de datos original con nuevos datos de defectos. Por su parte, Do et al. [67], como mecanismo de soporte de largo plazo, propusieron un modelo de gestión a cargo de un comité especializado con usuarios de la comunidad para generar políticas de operación, mantenimiento y expansión del Repositorio SIR.[13]

Tras revisar la literatura, se analizaron y discutieron los hallazgos en una sesión de lluvia de ideas (sección 2.2.3) con el objetivo de generar las siguientes buenas prácticas de diseño para simplificar la instrumentación y replicación de la estrategia de actualización:

[13] https://sir.csc.ncsu.edu/portal/index.php

Disparador de la actualización. Se busca responder en qué momento se debe iniciar la actualización. La misma puede ser implantada a través de un conjunto de condiciones que activará el procedimiento de actualización.

Caracterización de proyectos de calidad. Describir los atributos que un proyecto debe poseer para ser conservado dentro de la colección. Estos deben ser cuantificables en cada proyecto para identificar aquellos que cumplen con los criterios de calidad.

Actualización de proyectos. A partir de los criterios de calidad se clasifican los proyectos en la colección, considerando como vigentes a los que cumplen y desactualizados a los demás. Una vez creados los grupos, se debe explicitar si se recolectan los últimos datos disponibles de los proyectos vigentes o no.

Tratamiento de los proyectos desactualizados. Definir el procedimiento que se aplicará sobre los proyectos marcados como desactualizados. Dentro de las operaciones posibles se encuentra: eliminarlos, reemplazarlos o conservarlos.

Política de inclusión de nuevos proyectos. Determinar si los proyectos ausentes en la edición previa de la colección pueden ser incluidos en una versión actualizada, la metodología empleada para agregarlos y cuáles son las reglas que deben satisfacer para ser tenidos en cuenta. Estas reglas pueden ser las mismas utilizadas para identificar los proyectos de calidad o una variación de ellas.

4.2. Modelo propuesto: SUM4SOFT

SUM4SOFT (por sus siglas en inglés, ***S****ampling and* ***U****pdate* ***M****odel* ***for Soft****ware datasets*), es el artefacto propuesto en esta tesis doctoral para abordar las problemáticas reportadas de los estudios en la ISE. El modelo comprende las actividades de la Tabla 28 consideradas como relevantes para el muestreo de proyectos software, incluyendo también tareas dirigidas a abordar la actualización de las muestras. En la Tabla 29 se presentan las actividades obtenidas de los modelos de referencia (primera columna) y su correspondiente implementación en el modelo final (segunda columna). En algunos casos se modificaron los nombres de las actividades para adaptarlas al dominio específico de selección de proyectos software.

Tabla 29. Mapeo de los enfoques de referencia con las tareas de SUM4SOFT

Procesos	Tareas	Tareas de SUM4SOFT
Muestreo	Determinar los datos a obtener.	**A1.1.** Tarea 1: Determinar los datos a obtener.
	Definir los instrumentos de recolección.	**A1.1.** Tarea 2: Describir los métodos de búsqueda.
	Registrar el proceso de selección y evaluación.	**A1.1.** Tarea 3: Definir criterios de selección de fuentes.
		A1.1. Tarea 4: Seleccionar fuentes de información.
	Definir la población.	**A1.2.** Tarea 1: Especificar la población objetivo.
	Seleccionar la muestra.	**A1.2.** Tarea 2: Definir estrategia de muestreo.
Actualización	-	**A1.2.** Tarea 3: Definir estrategia de actualización.
Muestreo	Definir los instrumentos de recolección.	**A1.2.** Tarea 4: Construir el instrumento de recolección de datos.
	Realizar prueba piloto	**A1.2.** Tarea 5: Probar el instrumento de recolección de datos.
	Registrar el proceso de selección y evaluación.	**A2.** Tarea 1**.** Extraer la colección.
	Extraer y transformar los datos.	
Actualización	-	**A2.** Tarea 2. Actualizar la colección

El diseño del modelo abarca las cuatro perspectivas propuestas por Curtis et al. [29]:

De comportamiento. Especifica cuándo se deben ejecutar las actividades, incluyendo por tanto la identificación de secuencias, paralelismos, iteraciones, entre otras. En el caso del modelo propuesto, este diagrama contiene la secuencia ordenada de las tareas definidas para cada actividad.

Funcional. Describe qué actividades deben llevarse a cabo y qué flujo de productos de trabajo (por ejemplo, documentos) son necesarios para realizar las actividades y tareas. El diagrama representa los productos de trabajo de entrada y salida de cada tarea del modelo.

Organizacional. Pretende mostrar dónde y quiénes son los agentes (en cumplimiento de roles) involucrados en la realización de las actividades.

Informacional. Centrada en la estructura de los productos de trabajo producidos o consumidos por las actividades y en sus interrelaciones.

Cada una de las perspectivas fue diagramada con el lenguaje de modelado SPEM 2.0 [119]. Este proporciona los elementos de la Tabla 30 para modelar, documentar y gestionar los procesos.

Tabla 30. Elementos empleados de SPEM

Elemento	Definición	Símbolo
Tarea	Describe una unidad atómica de trabajo asignable a roles específicos. Una tarea está asociada a productos de trabajo de entrada y de salida.	
Rol	Representa la ocurrencia de una persona real realizando una actividad o tarea y teniendo responsabilidades relacionados con esta.	
Producto de trabajo	Es un elemento que representa tanto una entrada como una salida en el contexto de una actividad o tarea.	
Actividad	Representa una unidad general de trabajo asignable a roles específicos. Una actividad puede depender de productos de trabajo de entrada y producir productos de salida. La diferencia con una tarea es que funciona como un elemento de alto nivel, por lo que puede agrupar, por ejemplo, tareas u otras actividades. La actividad comprendida por otra se la denomina subactividad.	

En la Fig. 18 se muestra la perspectiva de comportamiento del modelo completo con dos actividades centrales: construir el instrumento de recolección de datos (A1) y ejecutar el instrumento de recolección de datos (A2). En particular, de la A1 se desprenden dos subactividades: especificar las fuentes (A1.1) y diseñar la estrategia de muestreo (A1.2).

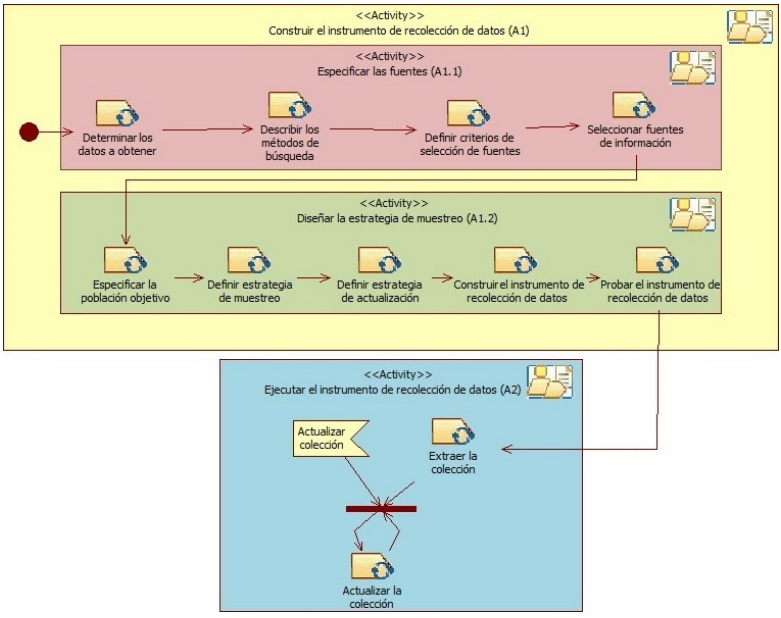

Fig. 18. Perspectiva de comportamiento de SUM4SOFT

En las siguientes secciones se detalla la estructura de SUM4SOFT. En primer lugar, se describen las actividades y subactividades (secciones 4.2.1.1, 4.2.1.2 y 4.2.2), proporcionando una visión general de ellas, junto con los diagramas de comportamiento y funcional, la lista de productos de trabajo involucrados y el desglose de las tareas con ejemplos ilustrativos. En segundo lugar, se presentan los productos de trabajo y las relaciones entre ellos a través del diagrama informacional (sección 4.2.3), destacando que las plantillas correspondientes se encuentran en las Fig. 37, Fig. 38 y Fig. 39 del APENDICE D. En tercer lugar, se representan los roles y sus responsabilidades mediante un diagrama organizacional, acompañados de una descripción de cada rol (sección 4.2.4).

4.2.1. Construir el instrumento de recolección de datos (A1)

En esta actividad se establecen las tareas necesarias para generar el instrumento con el cual construir y actualizar las muestras. La misma comprende las siguientes dos subactividades: especificar las fuentes y diseñar la estrategia de muestreo; e implica tareas como buscar los recursos que proporcionarán los datos, caracterizar la población objetivo, diseñar la estrategia de muestreo, entre otras. El resultado de la actividad es un instrumento de recolección de datos acompañado de su documentación de diseño y las instrucciones de uso.

4.2.1.1. Especificar las fuentes (A1.1)

Antes de iniciar el proceso de muestreo, es fundamental identificar claramente las fuentes de la información requerida. Es necesario especificar de dónde proviene cada unidad de información, ya que, según el proyecto y el tipo de dato, es posible que no todas las fuentes sean accesibles de la misma manera o en un mismo lugar. El objetivo de esta actividad es obtener una lista con las fuentes de los datos a utilizar y una guía simple de acceso a las mismas. En la Fig. 19 se encuentra la perspectiva funcional y de comportamiento de la actividad.

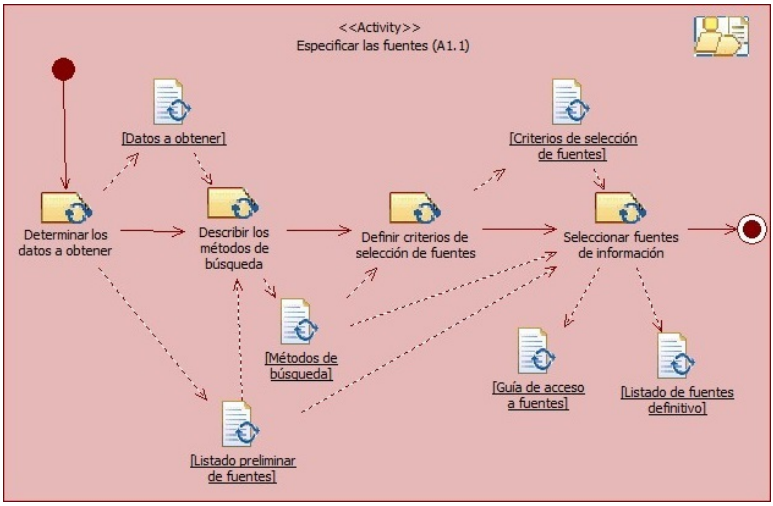

Fig. 19. Perspectiva funcional y de comportamiento de A1.1

Productos de trabajo involucrados en la actividad:

1. Datos a obtener.
2. Listado preliminar de fuentes.
3. Métodos de búsqueda
4. Criterios de selección de fuentes.
5. Guía de acceso a fuentes.
6. Listado de fuentes definitivo.

Tareas:

1. **Determinar los datos a obtener.** El responsable de la colección especifica los datos que se van a capturar y el diseñador de los instrumentos de recolección de datos enumera las fuentes de información que pueden proporcionarlos. Es probable que los datos a recopilar estén vinculados con los requerimientos de una investigación específica, por lo tanto, resulta deseable proporcionar la motivación para tenerlos en cuenta. Las fuentes pueden ser accesibles en línea o fuera de línea, y tienen su respectiva licencia de uso. Por ejemplo, la construcción de un modelo de aprendizaje predictivo basado en datos de repositorio probablemente necesite datos de múltiples proyectos, por lo que sería preferible una fuente en línea. En cambio, estudiar la forma de obtención de requerimientos dentro de una organización implicaría contactar a sus equipos de trabajo.
   Cada dato debe estar descripto con el suficiente nivel de detalle para ser identificable en las distintas fuentes. En el caso que los datos del proyecto se encuentren repartidos en más de una fuente,

se debe indicar de dónde se obtiene cada uno de ellos. El resultado es un listado preliminar de fuentes para los datos requeridos.

Productos de trabajo involucrados en la tarea:

- Datos a obtener.
- Listado preliminar de fuentes.

Ejemplos de datos a obtener:

- Número de estrellas.
- Colaboradores del proyecto.
- Total de incidentes resueltos.
- Cantidad confirmaciones en el último mes.

Ejemplos de fuentes:

- Plataforma de código fuente abierto (por ejemplo, Github, Gitlab, Bitbucket)
- Archivos de organizaciones.
- Entrevistas a integrantes del equipo de desarrollo.

2. **Describir los métodos de búsqueda.** Con los datos por obtener y una lista preliminar de fuentes, el diseñador de los instrumentos de recolección de datos define los métodos de búsqueda, proporcionando una guía detallada sobre cómo consultar y acceder a las fuentes seleccionadas. Esto implica investigar las formas o recursos de acceso a los datos de cada fuente, así como informar las restricciones aparejadas con cada método de búsqueda.

Productos de trabajo involucrados en la tarea:

- Datos a obtener.
- Listado preliminar de fuentes.
- Métodos de búsqueda.

Ejemplos de formas de acceso a fuentes:

- Servicio web a través de una interfaz REST.
- Servicio de transferencia de archivos (por ejemplo, *FTP*).
- Exportación de archivos desde una aplicación.
- *Scraping* de sitios web.

Ejemplos de herramientas específicas de descarga de repositorios:

- GHTorrent [120].
- GHS [121].
- GH Archive.[14]

Ejemplos de restricciones de acceso:

- Límites de solicitudes al servicio (por ejemplo, Github, Gitlab, Stackoverflow).
- Autenticación de usuarios (credenciales de acceso o *tokens*).
- Número máximo de elementos por consulta.

3. **Definir criterios de selección de fuentes.** A partir de los métodos de búsqueda, el diseñador de los instrumentos de recolección de datos establece los criterios de selección de fuentes, que funcionan como una lista de requisitos que una fuente debe cumplir para ser considerada viable. Cada requisito se registra como criterio de inclusión o exclusión. La redacción de los criterios debe ser lo suficientemente clara para evitar interpretaciones erróneas tanto dentro del equipo de trabajo como para investigadores externos que decidan replicar el estudio.

   Productos de trabajo involucrados en la tarea:
   - Métodos de búsqueda.
   - Criterios de selección de fuentes.

   Motivaciones que influyen en la selección de fuentes:
   - Reputación de la fuente.
   - Los términos y condiciones de acceso a la información.
   - Popularidad de su comunidad.
   - Número de repositorios disponibles.
   - Suficientes privilegios para acceder a los datos.

4. **Seleccionar fuentes de información.** Obtenidos los métodos de búsqueda, la lista preliminar de fuentes y los criterios de selección, el diseñador de los instrumentos de recolección de datos conforma el listado definitivo de fuentes junto con sus guías de acceso. El

[14] https://gharchive.org

listado incluye las fuentes que cumplieron los criterios de selección y la guía de acceso debe detallar cómo obtener las unidades de datos y las restricciones correspondientes para la elaboración del instrumento de recolección de datos.
Es conveniente realizar una evaluación con uno o más expertos de la lista y la guía obtenidas.

Productos de trabajo involucrados en la tarea:

- Métodos de búsqueda
- Criterios de selección de fuentes.
- Guía de acceso a fuentes.
- Listado de fuentes definitivo.

Ejemplos de documentación para obtener los datos de las fuentes:

- Consultas para los motores de búsqueda.
- Modelo de datos.
- Comentarios acerca de la sintaxis de las consultas.
- Enlaces a documentación complementaria.

4.2.1.2. Diseñar la estrategia de muestreo (A1.2)

El objetivo de esta actividad es diseñar las reglas y estrategias para construir y conservar una muestra curada de proyectos software. A partir de la guía de acceso a fuentes de la subactividad anterior, las salidas son el instrumento de recolección de datos, junto con su documentación de diseño e instrucciones de uso. En la Fig. 20 se encuentra la perspectiva funcional y de comportamiento de la actividad.

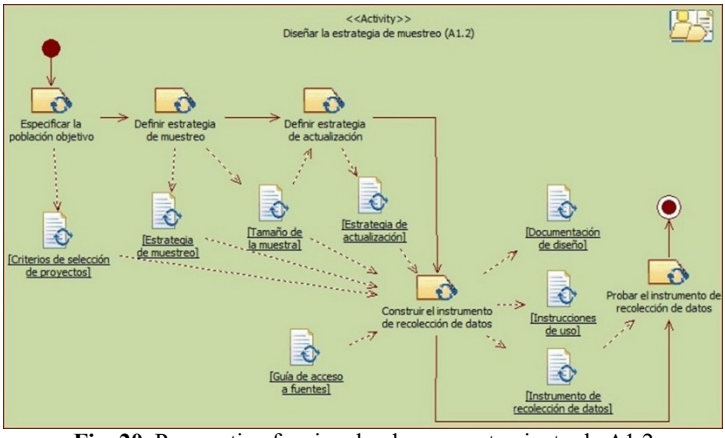

Fig. 20. Perspectiva funcional y de comportamiento de A1.2

Productos de trabajo involucrados en la actividad:

1. Criterios de selección de proyectos.
2. Estrategia de muestreo.
3. Tamaño de la muestra.
4. Estrategia de actualización.
5. Guía de acceso a fuentes.
6. Documentación de diseño.
7. Instrucciones de uso.
8. Instrumento de recolección de datos.

Tareas:

1. **Especificar la población objetivo.** El responsable de la colección especifica la población objetivo, que implica caracterizar los sujetos y en base a ello conformar el marco muestral. Esta caracterización se implementa en forma de umbrales o límites de aceptación que deben ser medibles en cada proyecto.

   Productos de trabajo involucrados en la tarea:

   - Criterios de selección de proyectos.

   Ejemplos de criterios de selección de proyectos:
   - Extensa historia de desarrollo (1000 confirmaciones o más).
   - Actividad reciente (una confirmación o más en el último mes).
   - Populares (500 estrellas o más).

2. **Definir estrategia de muestreo.** El analista de la muestra define la estrategia de muestreo y el tamaño de la muestra. La estrategia de muestreo a utilizar debe estar lo suficientemente detallada para implementar el algoritmo en el instrumento de recolección. En el caso del tamaño de la muestra, es deseable proporcionar una justificación de cómo se ha llegado al número seleccionado.

   Productos de trabajo involucrados en la tarea:

   - Estrategia de muestreo.
   - Tamaño de la muestra.

Ejemplos de estrategias de muestreo:

- Marco muestral completo: se recuperan datos de todos los proyectos en el marco muestral.
- Muestreo probabilístico: refiere a técnicas que involucran mecanismos aleatorización, es decir, todos los miembros del marco muestral tienen una probabilidad distinta de cero de ser incluidos.
- Muestreo deliberado: método no probabilístico donde se define una estrategia específica de selección, pero los sujetos no son recolectados aleatoriamente.

Herramientas para calcular el tamaño de la muestra:

- Software estadístico G*Power.[15]

3. **Definir estrategia de actualización.** El analista de la muestra proporciona la información necesaria para implementar la estrategia de actualización de la muestra. La estrategia debe incluir cuestiones como el disparador, la caracterización de los proyectos vigentes, tratamiento de los sujetos en la muestra y políticas de inclusión (revisar sección 4.1.2). De ser necesario, se podría redefinir el tamaño de la muestra actualizada.

   Productos de trabajo involucrados en la tarea:
   - Tamaño de la muestra.
   - Estrategia de actualización.

Ejemplos de disparadores de actualización:

- Nueva versión de algún proyecto en la muestra.
- Intervalo fijo de tiempo.
- Nuevo incidente abierto en algún proyecto.
- Número de confirmaciones enviadas en un proyecto el último mes.

[15] https://www.psychologie.hhu.de/arbeitsgruppen/allgemeine-psychologie-und-arbeitspsychologie/gpower

Ejemplos de estrategias de actualización de la muestra:

- Actualizar proyectos con datos recientes.
- Eliminar proyectos sin nuevos datos de desarrollo.
- Reemplazar proyectos no vigentes con nuevos proyectos fuera de la colección original.

4. **Construir el instrumento de recolección de datos.** A partir de la guía de acceso a fuentes de la A1.1, los criterios de selección de proyectos, el tamaño de la muestra y las estrategias de muestreo y actualización, el diseñador de los instrumentos de recolección de datos construye y nombra la herramienta para generar y conservar una muestra de proyectos. Esta puede ser implementada como una rutina o software entregable.
   Los lenguajes de programación y herramientas que se utilizan para implementar los algoritmos y procesamiento deben reportarse como documentación de diseño. Esto incluye las decisiones técnicas producto de la tecnología utilizada o las fuentes seleccionadas (por ejemplo, instrumentación del proceso de filtrado, como se resolvieron las restricciones de las fuentes). Como recomendación, resulta útil la búsqueda en la literatura de trabajos relacionados para identificar estudios realizados en el área de investigación y determinar cómo otros investigadores recolectaron los datos.
   Las salidas de la tarea son la rutina o software entregable para recuperar la colección de proyectos, la documentación con las especificaciones de diseño enumeradas, y las instrucciones para ejecutar la herramienta.

   Productos de trabajo involucrados en la tarea:
   - Criterios de selección de proyectos.
   - Estrategia de muestreo.
   - Tamaño de la muestra.
   - Estrategia de actualización.
   - Guía de acceso a fuentes.
   - Documentación de diseño.
   - Instrucciones de uso.
   - Instrumento de recolección de datos.

Ejemplos de decisiones en la implementación:

- Palabras claves utilizadas en los motores de búsqueda de las fuentes.
- Instrumentación del proceso de filtrado
- Tácticas de búsqueda aplicadas para mitigar las restricciones en los resultados de la búsqueda.
- El modelo de datos común en el caso que se recurra a más de una fuente de información del mismo dato. Por ejemplo, Gitlab, a diferencia de Github, emplea el término "*Merge request*" para referenciar la práctica de PR (ver sección 3.3.1).

5. **Probar el instrumento de recolección de datos.** El diseñador de los instrumentos de recolección de datos prueba la herramienta producida para revelar errores introducidos durante su construcción. De esta manera, si se hallan inconvenientes se pueden revisar las especificaciones de tareas anteriores y calibrar el instrumento.

   Productos de trabajo involucrados en la tarea:
   - Instrumento de recolección de datos.

Ejemplos de inconvenientes:

- Nuevas restricciones en las fuentes seleccionadas.
- Número de proyectos recuperados menor a lo anticipado.
- Ajustar umbrales de los criterios de selección de proyectos.

4.2.2. Ejecutar el instrumento de recolección de datos (A2)

A partir del instrumento de recolección de datos creado en la A1, esta tiene como objetivo principal crear una colección de proyectos software y mantenerla vigente pese a los cambios producidos por el paso del tiempo. La misma se divide en dos tareas: extraer la colección y actualizar la colección. En la Fig. 21 se encuentra la perspectiva funcional y de comportamiento de la actividad.

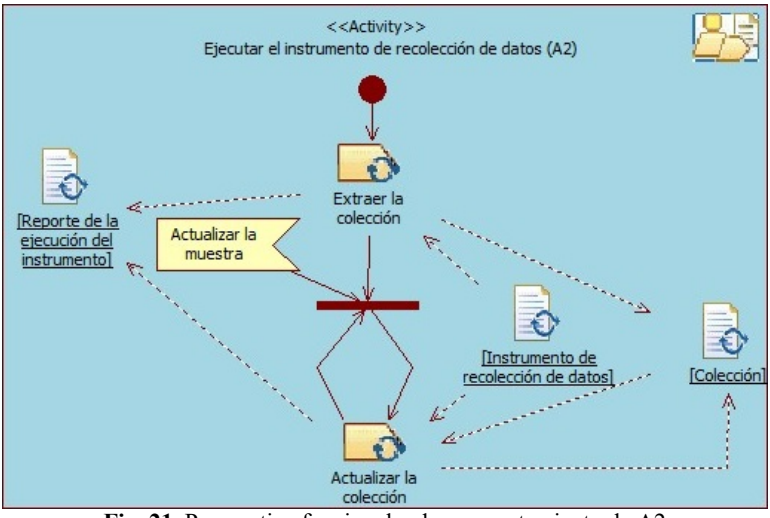

Fig. 21. Perspectiva funcional y de comportamiento de A2

Revisando la Fig. 21, se puede observar una barra de sincronización que se relaciona con la tarea Extraer la colección y la señal "Actualizar la muestra". Esto implica que para disparar la tarea 2 "Actualizar la colección" debe existir la colección en cuestión y cumplirse la condición de actualización para activar la señal.

Productos de trabajo involucrados en la actividad:

1. Instrumento de recolección de datos.
2. Reporte de la ejecución del instrumento.
3. Colección.

Tareas:

1. **Extraer la colección.** El responsable de la colección utiliza el instrumento de recolección de datos para extraer la colección con los proyectos que cumplen con los criterios de selección, aplicando la estrategia de muestreo y tamaño de muestra planificadas. Las salidas de la tarea son la colección y el reporte de ejecución del instrumento. Para garantizar que no se eliminaron repositorios relevantes, que no existan proyectos duplicados, valores faltantes o erróneos, el analista de la muestra revisa el conjunto de datos obtenido.

   Productos de trabajo involucrados en la tarea:

   - Instrumento de recolección de datos.
   - Reporte de la ejecución del instrumento.
   - Colección.

Datos acerca de la ejecución del instrumento:

- Duración de la ejecución.
- Fecha de ejecución.
- Proyectos obtenidos en cada etapa de filtrado.

2. **Actualizar la colección.** El responsable de la colección ejecuta nuevamente el instrumento de recolección de datos para renovar la colección con la estrategia de actualización una vez cumplida la condición del disparador. Se recibe como entrada la colección desactualizada y la herramienta, y se obtiene como salidas la nueva versión de la colección y el reporte de la ejecución. Para controlar que la colección no presente desviaciones notables en comparación con una muestra generada mediante la estrategia de muestreo, el analista de la colección verifica la validez de la colección actualizada.

   Productos de trabajo involucrados en la tarea:
   - Instrumento de recolección de datos.
   - Reporte de la ejecución del instrumento.
   - Colección.

4.2.3. Productos de trabajo

Además de la secuencia de tareas, los diagramas de comportamiento de las actividades reflejan la dinámica de los productos de trabajo que funcionan como entradas y salidas de las tareas. En la Fig. 22 se encuentra la perspectiva informacional de A2, donde se ven todos los productos de trabajo involucrados en el proceso y cómo se relacionan entre sí.

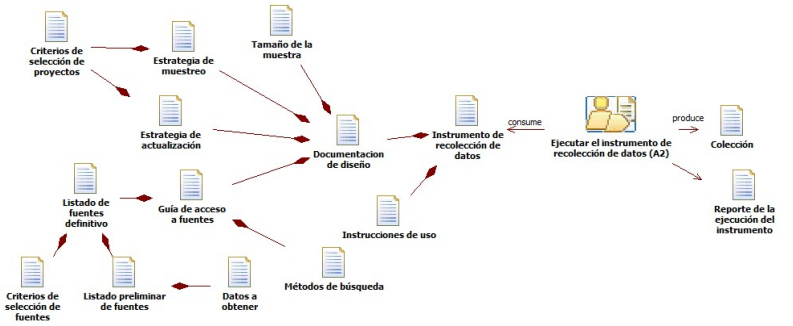

Fig. 22. Perspectiva informacional de A2

La A2 recibe como entrada el instrumento de recolección de datos. Dicho instrumento tiene una relación de composición con la documentación de diseño y las instrucciones de uso. A su vez la documentación de diseño comprende la especificación del tamaño de la muestra, la estrategia de muestreo, la estrategia de actualización y la guía de acceso a fuentes, dado que son imprescindibles para su implementación. Asimismo, las dos estrategias se diseñaron acorde a los criterios de selección de proyectos que caracterizan a la misma población. En cambio, la guía de acceso a fuentes está compuesta por los métodos de búsqueda y el listado de fuentes definitivo, siendo este último obtenido a partir de los criterios de selección de fuentes aplicado a un listado preliminar de fuentes. Por el otro lado, A2 produce dos productos de trabajo: la colección y el reporte de la ejecución del instrumento. En el APENDICE D se encuentran los modelos correspondientes a los productos de trabajo: guía de acceso a fuentes (Fig. 37), documentación de diseño (Fig. 38) y reporte de ejecución del instrumento (Fig. 39).

4.2.4. Roles

Se definieron tres roles para desempeñar las actividades y tareas del modelo propuesto: el responsable de la colección, el diseñador de los instrumentos de recolección de datos y el analista de la muestra. En la Fig. 23 se presenta la perspectiva organizacional del modelo propuesto. A continuación, se describen los roles anteriormente enumerados:

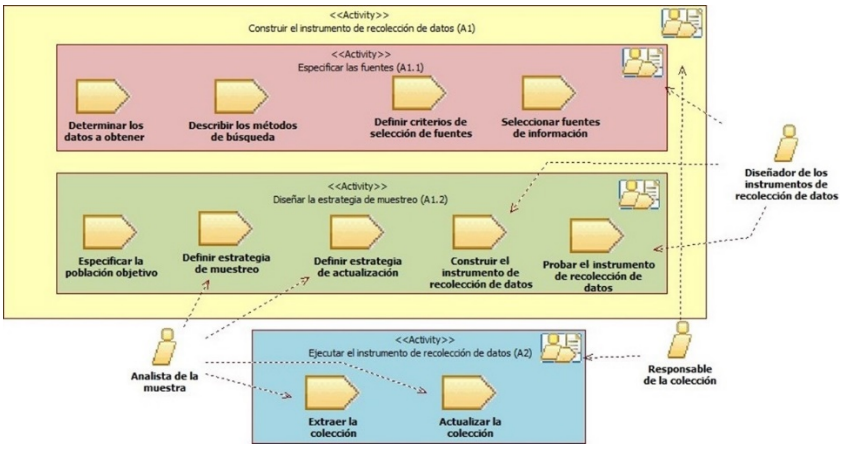

Fig. 23. Perspectiva organizacional del modelo

Responsable de la colección. Rol responsable de los resultados obtenidos de las dos actividades principales: construir el instrumento de recolección de datos y ejecutar el instrumento de recolección de datos. Es el experto del dominio, debe conocer en profundidad el tema de estudio, permitiéndole identificar los datos requeridos, dónde obtenerlos, precisar la población objetivo, antecedentes del tema, entre otras cuestiones para satisfacer los objetivos de la investigación. Dirige y supervisa todas las etapas del proceso.

Diseñador de los instrumentos de recolección de datos. Se encarga de las tareas de la subactividad A1.1, y además de construir y probar el instrumento de recolección de datos. Estas tareas implican, por un lado, la revisión y validación de los métodos de búsqueda y, por otro lado, el desarrollo y evaluación de las herramientas para recolectar los proyectos de las fuentes y filtrarlos, e implementar los algoritmos de muestreo y actualización de colecciones. Por consiguiente, el rol exige capacidad para codificar dichas rutinas.

Analista de la muestra. Se ocupa de definir las estrategias de muestreo y actualización, y verificar la composición de las colecciones. Este rol debe tener conocimientos en estadística aplicada necesarios para determinar cuestiones tales como el tamaño apropiado de la muestra, que estrategia de muestreo utilizar, si la muestra es representativa de la población. Asimismo, tiene la responsabilidad de controlar la calidad de los datos de las colecciones generadas en A2.

Dependiendo del rol el actor puede tomar funciones de ejecución, supervisión o dirección de las tareas y actividades. La definición de los roles se motivó por cuestiones de competencias, dado que los mismos requieren de los actores ciertos conocimientos técnicos. No es necesario que para cada rol exista una sola persona asignada, una persona puede realizar varios roles y un rol puede ser desempeñado por más de una persona.

4.3. Aplicación del modelo

En esta sección se presenta la utilización de SUM4SOFT en un escenario aplicación describiendo los detalles correspondientes a cada tarea y cómo operan los productos de trabajo en el marco de la solución.

Los detalles técnicos de implementación se encuentran en los modelos de productos de trabajo completados en el APENDICE D.

4.3.1. Especificar las fuentes (A1.1)

4.3.1.1. Determinar los datos a obtener

Los datos requeridos se presentan en la Tabla 31 junto con sus descripciones y motivaciones. Los metadatos de repositorio a recuperar responden a los criterios de calidad extraídos de los MS en la sección 3.3: la disponibilidad del código fuente, la utilización de herramientas para dar soporte a procesos, el tamaño, la historia de desarrollo, la popularidad, la colaboración del equipo de desarrollo y la actividad reciente. Las plataformas candidatas que brindan acceso a todos los datos de la Tabla 31 son Github[16] y Gitlab.[17]

Tabla 31. Datos a obtener

Id	Descripción	Motivación
NAME OWN	Nombre del repositorio Propietario del repositorio	Filtrado por palabras clave
URL	Url del repositorio	Acceder al repositorio
PL	Lenguajes de programación principales	Identificar lenguaje de programación principal del proyecto
SIZE	Tamaño del código	Cuantificar el tamaño
PRIV	Accesibilidad del repositorio	Determinar la disponibilidad del repositorio
MIRR	Si el repositorio es una copia	Determinar si el proyecto es original
CONTR CPR MPR	Número de colaboradores Cantidad de PR cerradas Cantidad de PR combinadas	Cuantificar la colaboración
COMM CREAT	Cantidad total de confirmaciones Fecha de creación	Cuantificar la historia
CI	Cantidad de incidentes resueltos	Cuantificar los incidentes
STARS FORKS	Número de estrellas Cantidad de bifurcaciones	Cuantificar la popularidad
COMMD CPRD MPRD ARCH	Fecha de la última confirmación Fecha de la última PR cerrada Fecha de la última PR fusionada Si el repositorio esta archivado	Detectar actividad reciente

[16] https://github.com
[17] https://gitlab.com

4.3.1.2. Describir los métodos de búsqueda

Tanto Github como Gitlab ofrecen dos interfaces de programación de aplicaciones (API) para consumir su base de datos: una a través de la arquitectura REST y un servicio de consultas GraphQL. Ambos métodos operan sobre el protocolo HTTP; sin embargo, el segundo funciona como una especificación de lenguaje de consultas, permitiendo personalizar tanto los datos como los formatos de salida.

4.3.1.3. Definir criterios de selección de fuentes

Una vez definida la lista de fuentes candidatas, se registraron los criterios de exclusión e inclusión de fuentes (ver Tabla 32). Dado el volumen de repositorios albergados en estas plataformas (en el orden de los millones), resulta necesario que las API admitan consultas con filtrado de lenguajes de programación (I1). Los demás criterios se seleccionaron para garantizar la extracción y la calidad de los datos.

Tabla 32. Criterios exclusión e inclusión de fuentes

Id	Descripción
I1	Filtrado de repositorios por lenguaje de programación desde API.
I2	Cuotas de transferencia de datos suficientes para realizar las extracciones.
E1	Retorno de datos inconsistentes.

4.3.1.4. Seleccionar fuentes de información

De la lista de fuentes candidatas (Github y Gitlab) la única que cumplió con todos los criterios considerados fue Github, dado que Gitlab no admite el filtrado de repositorios por lenguaje de programación desde sus API (I1).

4.3.2. Generar la muestra (A1.2)

4.3.2.1. Especificar la población objetivo

La población objetivo está compuesta por los proyectos Java de calidad alojados en Github. En la Tabla 33 se presentan los umbrales que operacionalizan cada criterio de selección de acuerdo con los resultados obtenidos de los MS en la sección 3.3.1 para identificar la población de proyectos Java de calidad. Además, para evitar repositorios no deseados que pudieran cumplir con los umbrales de calidad (por ejemplo, demos,

tutoriales o repositorios de configuración), se excluyeron aquellos que incluyeran alguna de las siguientes palabras clave: *benchmark*, *conf*, *demo*, *docs*, *exam*, *guide*, *sample*, *template*, *tutorial* y *wiki*.

Tabla 33. Criterios y umbrales de calidad para caracterizar la población objetivo

Id.	Umbral	Criterio
U1	Lenguaje de programación Java	Proyectos Java
U2	Repositorios públicos	Metadatos disponibles
U3	Repositorios GIT	Uso de herramientas para dar soporte a procesos
U4	50 incidentes resueltos o más	
U5	Repositorio que no es una copia	Proyectos originales
U6	10000 KB o más	Tamaño
U7	3 o más colaboradores	Colaboración
U8	50 PR o más	
U9	1 año o más desde su creación	Historia
U10	1000 confirmaciones o más	
U11	10 o más bifurcaciones	Popularidad
U12	10 o más estrellas	
U13	1 o más confirmaciones o PR cerradas/fusionadas en el último mes	Actividad reciente
U14	1 o más confirmaciones por cada mes en el último año	
U15	Repositorio no archivado	

4.3.2.2. Definir estrategia de muestreo

Se empleó muestreo aleatorio estratificado para seleccionar los sujetos en el marco muestral al igual que estudios relacionados [122]. Esta técnica divide la población en subpoblaciones no solapadas denominados estratos y obtiene los elementos de cada estrato de manera aleatoria [114]. La cantidad de elementos seleccionados de cada estrato son proporcionales a la cantidad de sujetos en el grupo con respecto al marco muestral.

La implementación del procedimiento de selección involucró la generación de los estratos o grupos a partir del tamaño del repositorio (SIZE) utilizando el algoritmo para agrupamiento k-means [123, 124]. K-means divide m sujetos de un conjunto de datos con valores en n dimensiones en k grupos buscando minimizar la suma de cuadrados dentro del mismo grupo [125]. En este caso, los valores corresponden al puntaje de SIZE de cada proyecto en el marco muestral.

Antes de aplicar el algoritmo, se separaron los proyectos con valores extremos de SIZE y se aplicó k-means con cinco grupos. El número de grupos se fijó después inspeccionar el conjunto de datos completo y

experimentar con diferentes valores. Una vez obtenidos los grupos, se procedió a seleccionar aleatoriamente los proyectos de cada grupo proporcionalmente de acuerdo con el tamaño del estrato y del marco muestral, por ejemplo: si el marco muestral tuviese 800 sujetos, y los grupos 170, 180, 220, 100 y 130 sujetos, en una muestra de 100 proyectos se elegirían 21, 22, 28, 13 y 16 proyectos respectivamente. A la muestra resultante, se le incorpora un subconjunto proporcional de los proyectos separados inicialmente por ser identificados como valores extremos. La Fig. 24 ilustra gráficamente la estrategia.

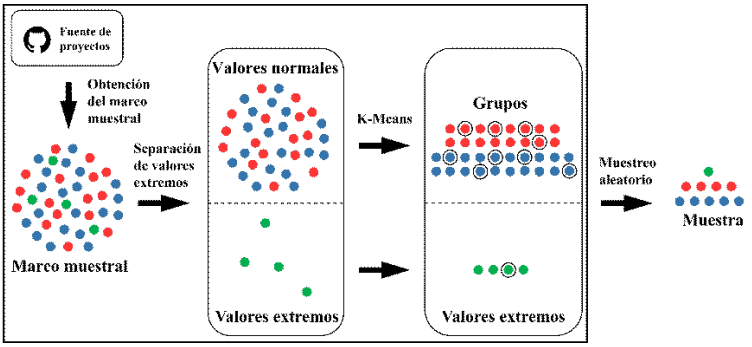

Fig. 24. Esquema de la estrategia de muestreo

4.3.2.3. Definir estrategia de actualización

La condición para desencadenar la actualización de la muestra se fijó cada 365 días, es decir, una vez por año contando desde la creación de la muestra. El número surgió al examinar el gráfico de barras de la Fig. 15, donde más de un 35% de los proyectos mostraron indicios de inactividad en el primer año de desarrollo.

El proceso inicia generando un marco muestral nuevo a partir de los umbrales definidos en la Tabla 33. Si los elementos de la muestra original tienen actividad reciente se procede a actualizar los proyectos en la muestra por las nuevas versiones disponibles. En cambio, los proyectos sin actualizaciones durante ese período de un año se descartan de la muestra actualizada.

Una vez descartados los proyectos, se inicia el reemplazo de los mismos donde los proyectos a incorporar serán obtenidos a partir de un proceso similar al muestreo estratificado aleatorio definido en la estrategia de muestreo. Primero, se generan los estratos a partir marco muestral nuevo de la misma manera que la estrategia de muestreo, es decir, se

separan lo valores extremos y se aplica k-means con cinco grupos para crear los estratos. Segundo, se registra la cantidad de proyectos de cada estrato que existen en la muestra actualizada (sin los repositorios descartados) y se obtiene la diferencia con la cantidad proporcional de elementos correspondientes al estrato equivalente en el marco muestral. Finalmente se seleccionan los proyectos en el marco muestral de acuerdo a las diferencias registradas por grupo en el paso previo. La Fig. 25 ilustra gráficamente la estrategia de actualización.

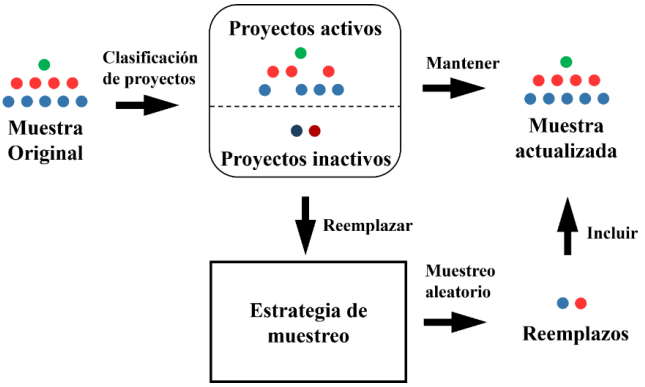

Fig. 25. Esquema de la estrategia de actualización

4.3.2.4. Construir el instrumento de recolección de datos.

Se desarrollaron rutinas en el lenguaje Python para extraer los datos de la fuente y automatizar la generación del marco muestral, el muestreo de proyectos y la actualización de la muestra. Para instrumentar la generación del marco muestral se aplicaron los umbrales de la Tabla 33 en dos etapas. En la primera etapa, se recolectaron y filtraron algunos repositorios directamente desde el motor de búsqueda de la API GraphQL de Github. En la Tabla 34 se encuentran las sentencias empleadas para cada umbral.

Tabla 34. Umbrales aplicados en el primer filtrado

Id Umbral	Sintaxis *query* en GraphQL
U1	language:java
U2	is:public
U5	mirror:false
U6	size:>=10000
U9	created:<= un año atrás
U11	forks:>=10
U12	stars:>=10
U15	archived:false

En la segunda etapa de filtrado, se aplicaron los umbrales restantes de la Tabla 35, allí se puede visualizar el listado con las condiciones para implementar cada uno de ellos incluyendo el filtrado por palabras clave, donde se evaluó que no existan ningunas de las palabras dentro del nombre del repositorio. Finalmente, se definió el nombre "JavaQ" para identificar al instrumento de recolección de datos.

Tabla 35. Umbrales aplicados en el segundo filtrado

Id Umbral	Condición
U4	CI >= 50
U7	CONTR >= 3
U8	CPR + MPR >= 50
U10	COMM >= 1000
U13	(COMMD >= un mes atrás) OR (CPRD >= un mes atrás) OR (MPRD >= un mes atrás)
U14	COMM >= 1 por mes en el último año
Palabras Clave	("simple" OR "tutorial" OR "demo" OR "conf" OR "exam" OR "docs" OR "benchmark" OR "wiki" OR "guide" OR "template") ⊄ NAME

4.3.3. Ejecutar el instrumento de recolección de datos (A2)

El tres de enero del 2024 se ejecutó el instrumento de recolección de datos JavaQ. En esta primera ejecución, la herramienta obtuvo como resultado 15868 proyectos Java, que la segunda etapa de filtrado redujo a un marco muestral de 890 sujetos. El tiempo de duración de la rutina fue de 3 horas 15 minutos y 32 segundos.

Con respecto a los productos de trabajo empleados en A2, el instrumento de recolección de datos JavaQ se encuentra disponible en un repositorio de Github[18] y la colección generada durante la ejecución puede descargarse en el siguiente enlace.[19]

4.4. Conclusión

En este capítulo, se describió SUM4SOFT como propuesta de solución de esta tesis. En la Fig. 26 se muestran las actividades de CD que se desarrollaron en este capítulo (color verde oscuro), y las realizadas previamente (color verde claro). SUM4SOFT se diseñó basándose en

[18] https://github.com/juancarruthers/thresholds_experiment
[19] https://bit.ly/sampling_frame_msr

enfoques de referencia sobre muestreo en la Ingeniería del Software identificados en la literatura. El artefacto, que incluye actividades, tareas, productos de trabajo y roles, tuvo una primera validación interna y refinamiento a partir de experiencias prácticas en un escenario de prueba, tal y como fue descrito en la sección anterior.

El próximo capítulo, aborda las actividades Minería de repositorios y Estudio de caso, que se enfocan en los estudios empíricos conducidos para validar el artefacto diseñado en escenarios externos y continuar el refinamiento de SUM4SOFT.

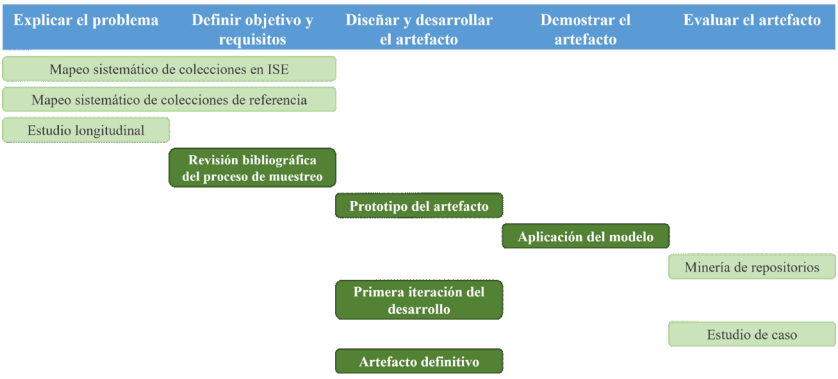

Fig. 26. Actividades abarcadas en el CAPITULO 4

CAPITULO 5

Validación del artefacto

Este capítulo presenta las evaluaciones practicadas sobre la solución propuestas, centradas en validar el modelo de procesos SUM4SOFT desde dos enfoques. Por un lado, si el instrumento de recolección de datos JavaQ, construido a través de SUM4SOFT, es capaz de generar muestras representativas de una población de proyectos software. Por el otro, recibir mejoras en el modelo de proceso SUM4SOFT. En este sentido, se condujeron dos estudios empíricos: un MRS y un estudio de caso.

El capítulo se organiza de la siguiente manera: la sección 5.1 presenta la planificación y resultados del MRS, la sección 5.2 muestra el estudio de caso conducido en la Universidad de Hamburgo que comprende la replicación del MRS y las mejoras a SUM4SOFT.

5.1. Estudio de minería de repositorios

Se realizó un estudio de minería de repositorios (MRS) siguiendo los lineamientos de Vidoni [38] con tres colecciones de proyectos, dos de las cuales fueron construidas con el instrumento de recolección de datos JavaQ, descripto en la sección 4.3. El objetivo fue generar los umbrales de calidad de tres métricas de código: complejidad ciclomática (McCC), cantidad de líneas de código (LOC) y peso de métodos por clase (WMC). Este estudio sirvió para demostrar la utilidad del modelo en un contexto real de la ISE.

5.1.1. Umbrales de métricas de código

Según Fenton & Bieman [126], la Ingeniería del Software abarca técnicas aplicadas en la construcción y el soporte de productos de software, involucrando actividades como la gestión, la planificación, el modelado, la prueba, el mantenimiento, entre otras. Estas actividades adhieren a un enfoque ingenieril, es decir, que deben ser comprendidas

y controladas para reducir la incertidumbre mientras el software es especificado, diseñado, construido y mantenido. Para lograr este nivel de control, se necesita monitorear el estado de los proyectos, productos, procesos y recursos a través de métricas [127]. Al trabajar con métricas, es esencial establecer puntos de referencia para vincular un valor métrico particular con semánticas útiles [127], ya que disponer solamente de un valor numérico carece de contexto y no es suficiente para evaluar el estado del sistema. Los umbrales delimitan qué valores son aceptables y cuáles son anormales en los componentes de software, proporcionando una orientación para la toma de decisiones a los desarrolladores, gerentes y otros miembros del equipo [128].

La literatura ofrece diferentes estrategias para establecer umbrales de calidad. Por ejemplo, McCabe [68] definió un umbral para McCC basado en la experiencia de los programadores, sugiriendo que un método con un valor superior a 10 debería ser refactorizado. Nejmeh [129] adoptó un enfoque similar para fijar el umbral de otra métrica de complejidad (NPATH), y condujo estudios empíricos en los Laboratorios AT&T Bell estableciendo un umbral de 200. Sin embargo, estos valores se basan en experiencias subjetivas, por lo que su replicación y generalización son cuestionables. De esta manera surgen los métodos dirigidos por datos, donde los umbrales dependen de los datos volcados en un modelo más que de las opiniones aportadas por usuarios experimentados. Estos modelos determinan los umbrales de acuerdo a propiedades estadísticas de las métricas basándose en colecciones de proyectos de referencia [130 - 132].

5.1.2. Planificación

Para conducir el MRS se abordaron las tres fases propuestas por Vidoni [38]: planificación, ejecución y análisis. Para guiar el estudio se formularon las siguientes preguntas de investigación (PI):

> **PI1. ¿Es necesario actualizar las colecciones de proyectos de software?** Motivación: Evaluar la vigencia de una colección referenciada con frecuencia en la literatura y determinar si resulta necesario actualizarla.
>
> **PI2. ¿JavaQ es capaz de generar muestras representativas?** Motivación: Evaluar la efectividad del instrumento de recolección de datos obtenido de SUM4SOFT para construir muestras representativas.

5.1.2.1. Colecciones generadas

Se generaron tres colecciones para abordar las PI: una muestra desactualizada (*Qualitas Corpus*), una muestra con datos recientes (Colección Vigente) y una versión nueva de la muestra desactualizada (Qualitas Actualizado). Todas las colecciones contienen 112 proyectos.

Qualitas corpus (Q). Corresponde a la última versión del Qualitas Corpus de septiembre del 2013. Se descargó el código fuente del paquete de distribución "r" del sitio web oficial de la colección[20] y se mapearon los proyectos a sus respectivos enlaces en Github.

Colección Vigente (C). Esta colección se obtuvo partiendo de un marco muestral vigente de Github aplicando la estrategia de muestreo aleatorio estratificado (ver Fig. 24). Es el resultado de aplicar la estrategia de muestreo de la sección 4.3.2.2 en el marco muestral obtenido en la ejecución del instrumento de recolección de datos descripto en la sección 4.3.

Qualitas Actualizado (Qu). Tomando la colección Q como base, se actualizaron los proyectos con la estrategia de actualización definida en la sección 4.3.2.3. Para ello, se descargaron todos los sistemas de Q y se mapearon a sus respectivos enlaces en Github. Después se ejecutó el instrumento de recolección de datos JavaQ en lo que correspondería a la tarea 2 "Actualizar la colección" de la A2 de SUM4SOFT.

5.1.2.2. Generación de umbrales

Se recolectaron los proyectos de las colecciones y se calcularon los umbrales de tres métricas de código (ver Tabla 36): dos a nivel de método (McCC y LOC); y una a nivel de clase (WMC). Las métricas fueron elegidas a partir de los resultados de los mapeos sistemáticos de la sección 3.3, dado que son empleadas en actividades tales como la detección de defectos, mantenimiento del software, estimación del esfuerzo, entre otras. Al igual que otros artículos relevantes [133 - 135], se utilizó la herramienta Sourcemeter[21] para analizar el código Java y computar las métricas.

Tabla 36. Métricas de código analizadas

[20] http://www.qualitascorpus.com/
[21] https://sourcemeter.com/

Id	Descripción	Tipo
LOC	Cantidad de líneas de código	Tamaño
McCC	Complejidad ciclomática	Complejidad
WMC	Peso de métodos por clase	

Para la generación de los umbrales de las métricas se implementó el enfoque estadístico basado en datos planteado por Alves et al. [130]. Esta técnica calcula la función de distribución acumulada de la métrica estudiada y establece tres valores de referencia: límite inferior (70%), medio (80%) y límite superior (90%). Estos umbrales delimitan cuatro niveles de riesgo que puede registrar un componente de sistema (método, clase o paquete): riesgo bajo (<= 70%), riesgo moderado (70% - 80%), riesgo alto (80% - 90%) y riesgo muy alto (> 90%). Esta categorización de riesgo refiere al grado de urgencia en el cual se debe refactorizar el código, donde el riesgo medio exige atención a largo plazo, el riesgo alto mediano plazo y el riesgo muy alto debe atenderse lo antes posible. Por ejemplo, los umbrales de McCC obtenidos con la colección C en la Fig. 27 registraron 4, 7 y 12 que corresponden a los límites inferior, medio y superior respectivamente; por lo tanto, las clases con valores menores a 4 son de bajo riesgo, entre 4 y 7 son moderadas, entre 7 y 12 son de alto riesgo, y mayores a 12 de muy alto riesgo.

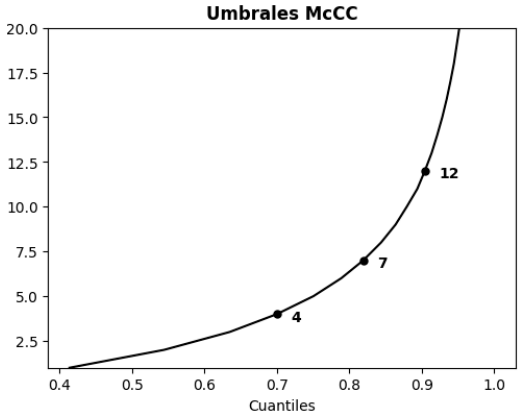

Fig. 27. Función de distribución acumulada de McCC de la colección C

Se demostró la viabilidad del enfoque en múltiples contextos de ISE, como predicción de defectos [136], *smells* en el código de pruebas [137] y líneas de producto software [138]. El procedimiento se descompone en seis pasos:

Extracción de métricas. Las métricas se extraen para cada componente de todos los sistemas de la muestra, registrando un valor de métrica para el análisis y otro valor de métrica para la ponderación o peso. Por ejemplo, tomando el valor de McCC como métrica de análisis y LOC como peso, en el repositorio liquibase/liquibase[22] existe el método "Scope.getCurrentScope()" con McCC de 6 y LOC de 36.

Cálculo de peso proporcional. Para cada componente se calcula el porcentaje de peso dentro de su sistema, es decir, se divide la cantidad de LOC del componente entre el total de LOC del sistema. Siguiendo con el ejemplo, para el método "Scope.getCurrentScope()", se divide 36 entre 82873 (total de LOC en liquibase/liquibase), lo que representa el 0.043% del sistema completo.

Agregación de componentes. Se suman los pesos proporcionales de todos los componentes de un sistema por cada valor de la métrica analizada. Esto permite, graficar un histograma de frecuencias por sistema con la distribución del peso en cada valor de la métrica analizada. Entonces según el ejemplo, todos los métodos con McCC igual a 6 equivalen al 4.98% del total de LOC de liquibase/liquibase (gráfico a) de la Fig. 28).

Agregación de sistemas. Se normalizan los pesos de acuerdo al número de sistemas, y se suman los pesos de todos los sistemas por cada valor de la métrica. Esto tiene como resultado un histograma que describe la distribución ponderada de la métrica analizada para toda la muestra. Entonces, al dividir todos los pesos por los 112 sistemas y realizar la sumatoria de pesos por cada valor de McCC, se obtuvo que, el valor 6 en McCC representa el 3.836% de todo el código en la muestra (gráfico b) de la Fig. 28).

Agregación de los pesos proporcionales. Se ordenan los valores de la métrica de manera ascendente y se acumulan los pesos de cada valor para obtener el porcentaje que representan en la muestra, lo que equivale a calcular su función de distribución acumulada. Por ejemplo, realizando la sumatoria de los pesos

[22] https://github.com/liquibase/liquibase

proporcionales de la muestra, el 70% del código registra como máximo un McCC de 4 (ver distribución acumulada del gráfico c) en la Fig. 28).

Generación de umbrales. Los umbrales se crean al seleccionar el porcentaje del código que se busca representar. Por ejemplo, para representar el 70% del código en McCC, el umbral derivado es 4.

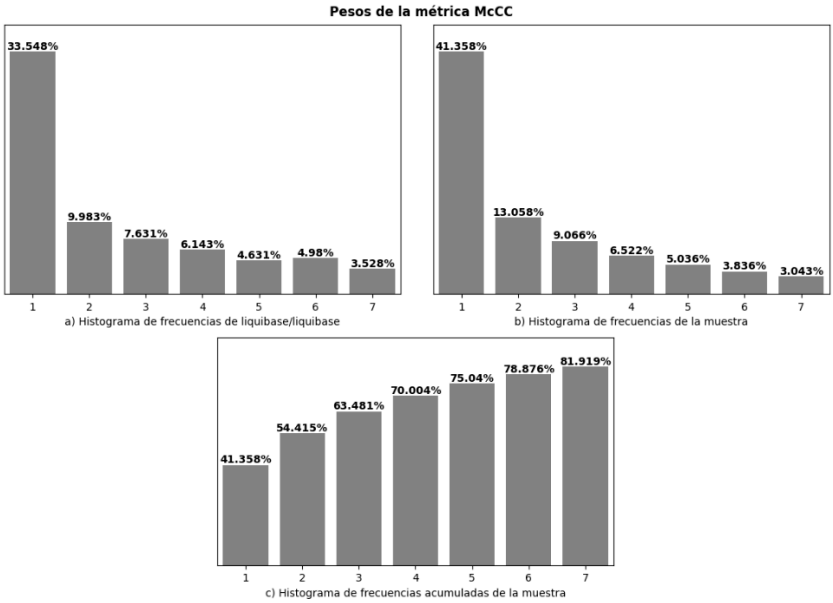

Fig. 28. Pesos de McCC calculados con el método de Alves et al

5.1.2.3. Detección de valores extremos

Para detectar aquellos sistemas que sean valores fuera de rango, se analizaron de manera gráfica las distribuciones acumuladas de estos individualmente en cada métrica. En el gráfico a) de la Fig. 29 se pueden observar curvas grises que representan las distribuciones de los proyectos, y una curva roja que corresponde a la función de distribución acumulada del conjunto de datos completo. Este gráfico permite identificar los sistemas con curvas más distantes a la de referencia, siendo un claro ejemplo de ello la línea roja marcada en el gráfico b) de la Fig. 29, la cual corresponde a la función de distribución acumulada del repositorio hapifhir/org.hl7.fhir.core.[23]

[23] https://github.com/hapifhir/org.hl7.fhir.core

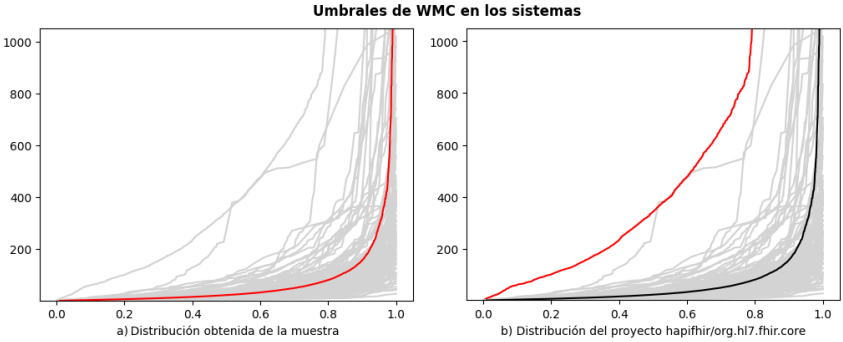

Fig. 29. Detección gráfica de valores fuera de rango

Una vez determinados los valores fuera de rango, se procedió a reemplazarlos por otro sistema similar en el marco muestral acorde a la estrategia de muestreo instrumentada. En este caso, se calculó la distancia euclidiana [139] del tamaño (SIZE) de todos los proyectos en el marco con respecto al sujeto identificado como valor fuera de rango, y aquel que registrara el valor más cercano a cero se eligió como reemplazante. Esto implicó el cómputo de métricas para el nuevo proyecto, y recalcular los umbrales.

5.1.3. Ejecución

El tres de enero del 2024 se ejecutó el instrumento de recolección de datos JavaQ reportado en la sección 4.3, obteniendo como resultado un marco muestral actual de 15868 proyectos Java que la segunda etapa de filtrado redujo a 890 sujetos. El tiempo de duración de la rutina fue en total: 3 horas, 15 minutos y 52 segundos. Posteriormente, el cálculo de métricas con la herramienta Sourcemeter de las colecciones Q, C y Qu en promedio duraron ocho horas.

Las colecciones utilizadas durante el estudio, así como las rutinas para extraer los datos y generar los gráficos se alojaron en repositorios de Zenodo[24] y Github[25] donde se pueden consultar.

[24] https://doi.org/10.5281/zenodo.11059968
[25] https://github.com/juancarruthers/thresholds_experiment

5.1.4. Análisis

En la Fig. 30, están representadas las distribuciones de probabilidad de las tres métricas estudiadas. Después de examinar los gráficos, al igual que otros autores [140 - 142], se observó que estas tienen distribuciones no normales. Por lo tanto, se decidió utilizar pruebas no paramétricas.

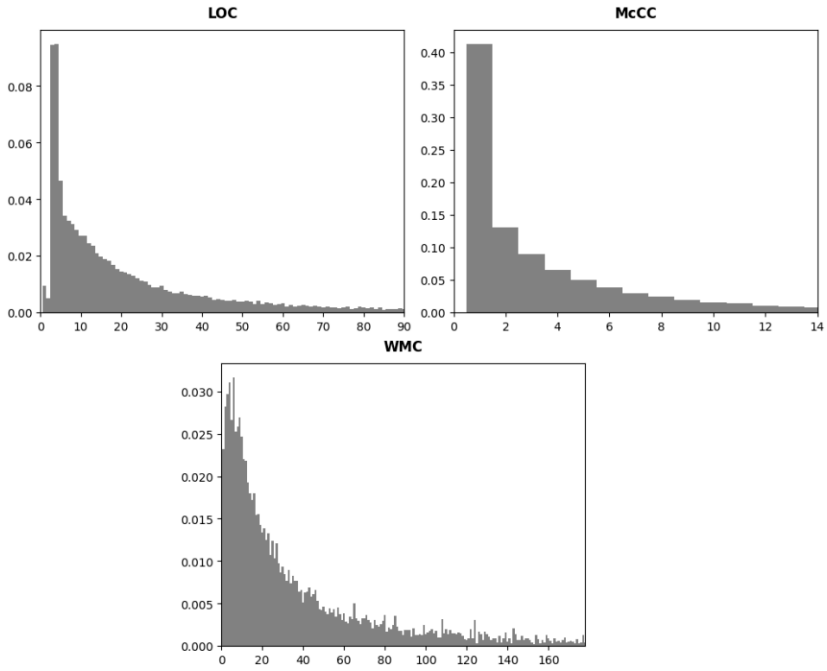

Fig. 30. Distribución de probabilidades de las métricas estudiadas

Para evaluar las PI se condujo una prueba de hipótesis para asegurar que los resultados fueran válidos desde una perspectiva estadística fijando un nivel de significación de 0.05. La prueba Kolmogorov-Smirnov para dos muestras independientes (KS) es una prueba no paramétrica diseñada para ser empleada con datos ordinales que requiere la construcción de la distribución de frecuencia acumulada de la variable estudiada [19]. Esta prueba evalúa si dos muestras provienen de la misma población, y si ese fuese el caso, se espera que sus distribuciones de frecuencias acumuladas sean idénticas o muy cercanas entre sí.

La hipótesis nula de la prueba Kolmogorov-Smirnov es: $F_1(X) = F_2(X)$ para todos los valores de X, donde $F_i(X)$ alude a la función de distribución acumulada de la i-ésima muestra. Al computar los umbrales de una muestra con el método de Alves et al. [130], se

determina la función de distribución acumulada de la métrica analizada. La prueba KS evalúa si dos distribuciones de muestras independientes son estadísticamente similares en la misma variable.

Dado que se analiza la consistencia de las dos muestras completas en todas sus dimensiones, si la prueba de hipótesis de alguna de las métricas es rechazada, la hipótesis nula será rechazada, es decir, todas las pruebas de hipótesis deben ser aceptadas para considerarse que las muestras son equivalentes o consistentes. La Fig. 31 presenta la expresión simbólica de la hipótesis nula evaluada en las dos PI, donde $F_i(X)$ refiere a la distribución acumulada de las muestras comparadas y las variables entre paréntesis corresponden a las métricas analizadas.

$$H_0: F_1(X) = F_2(X):$$
$$[F_1(LOC) = F_2(LOC)] \wedge [F_1(McCC) = F_2(McCC)] \wedge [F_1(WMC) = F_2(WMC)]$$

Fig. 31. Hipótesis nula de las PI

Para la PI1, se evaluó la vigencia de la colección considerada como desactualizada; por lo tanto, se emparejaron Q y C para identificar si, después de diez años, los umbrales producidos con el *Qualitas Corpus* pueden considerarse vigentes en las dimensiones analizadas. Este escenario llevó a las siguientes hipótesis:

> **H_0: Q(X) = C(X).** La distribución de datos del *Qualitas Corpus* y la colección actual recuperada de Github provienen de la misma población.
>
> **H_1: Q(X) ≠ C(X).** La distribución de datos del *Qualitas Corpus* y la muestra actual recuperada de Github no provienen de la misma población.

La PI2 tenía como objetivo evaluar la efectividad del instrumento de recolección de datos JavaQ obtenido de SUM4SOFT para generar muestras representativas de proyectos; por lo tanto, en este caso se emparejaron las muestras C y Qu. Este escenario llevó a las siguientes hipótesis:

> **H_0: Qu(X) = C(X).** La distribución de datos de la versión actualizada del *Qualitas Corpus* y la muestra actual recuperada de Github provienen de la misma población.
>
> **H_1: Qu(X) ≠ C(X).** La distribución de datos de la versión actualizada del *Qualitas Corpus* y la muestra actual recuperada de Github no provienen de la misma población

5.1.5. Resultados

A continuación, se responden las PI de acuerdo con los resultados obtenidos del análisis de datos.

5.1.5.1. PI1. ¿Es necesario actualizar las colecciones de proyectos de software?

En la Tabla 37 se volcaron los umbrales calculados de LOC, McCC y WMC, de las colecciones Q y C. El análisis de los umbrales reveló diferencias sustanciales entre las dos colecciones, se observaron valores más bajos para C en todos ellos. Comenzando con LOC, Q registró 46, 71 y 132 para los umbrales inferior, medio y superior respectivamente; mientras que C registró 29, 43 y 77 en los mismos puntos. Comparando el mismo umbral entre colecciones, los valores de LOC en Q fueron entre 58.6% y 71.4% mayores. Una tendencia similar se observó en las métricas de complejidad, con Q registrando números en McCC entre un 57.1% y 83.3% más altos, y entre un 62% y 79% en WMC.

Tabla 37. Umbrales de las colecciones Q y C

Colección	Métrica	Inferior	Medio	Superior
Q	LOC	46	71	132
	McCC	7	11	22
	WMC	81	136	265
C	LOC	29	43	77
	McCC	4	7	12
	WMC	50	80	148

Al contrastar los valores de diferentes umbrales (como ser, los límites inferiores de Q y los medios de C) se detectó que un método con 45 LOC sería etiquetado como alto riesgo en la muestra C, pero de bajo riesgo con los umbrales de Q. Esto se notó también en otras métricas: una clase con WMC igual a 81 se categorizaría como alto riesgo en C y bajo riesgo en Q.

Estos resultados se extendieron con un análisis inferencial realizando la prueba KS para comparar las distribuciones de umbrales entre las muestras Q y C para cada métrica. Los resultados de la Tabla 38 indican p-valores por debajo del nivel de significación de 0.05 para todas las métricas confirmando la suposición inicial de que los umbrales generados con estas dos muestras tienen diferencias significativas.

Tabla 38. Resultados de la prueba KS entre los umbrales de Q y C

Métrica	Comparación Q y C	
	KS	P-valor
LOC	0.13	0
McCC	0.13	0.01
WMC	0.13	0

De esta manera, se rechazó la hipótesis nula y se aceptó la hipótesis alternativa: La distribución de datos del *Qualitas Corpus* y la muestra actual recuperada de Github no provienen de la misma población. Por lo tanto, se considera al *Qualitas Corpus* como una colección no vigente para las métricas de código fuente LOC, McCC y WMC, motivando la necesidad de implementar mecanismos para actualizar las colecciones.

5.1.5.2. PI2. ¿JavaQ es capaz de generar muestras representativas?

Al igual que en la PI1, se calcularon y emparejaron los umbrales de dos colecciones: Qu y C, pero en este caso ambas se crearon con el instrumento de recolección de datos reportado en la sección 4.3. En la Tabla 39 se registraron los umbrales inferior, medio y superior de las muestras para cada métrica. Al comparar estos umbrales se detectaron diferencias marginales entre las dos. En particular, los umbrales de McCC fueron idénticos, con valores inferior, medio y superior de 4, 7 y 12 respectivamente.

Tabla 39. Umbrales de las colecciones Qu y C

Colección	Métrica	Inferior	Medio	Superior
Qu	LOC	29	42	74
	McCC	4	7	12
	WMC	52	80	151
C	LOC	29	43	77
	McCC	4	7	12
	WMC	50	80	148

Nuevamente se condujo la prueba KS para comparar las distribuciones acumuladas de los umbrales de las muestras Qu y C de cada métrica. Los resultados de la prueba (Tabla 40) presentaron p-valores superiores a 0.98 en todas las variables analizadas, infiriendo que no hay diferencias significativas en las distribuciones de umbrales entre las muestras Qu y C.

Tabla 40. Resultados de la prueba KS entre los umbrales de Qu y C

Métrica	Comparación Qu y C	
	KS	**P-valor**
LOC	0.02	0.99
McCC	0.01	1
WMC	0.01	1

Dado que los p-valores son mayores al nivel de significación en todas las métricas se acepta la hipótesis nula: La distribución de datos de la versión actualizada del *Qualitas Corpus* y la muestra actual recuperada de Github provienen de la misma población. Esto implica que el instrumento de recolección de datos JavaQ fue capaz de generar muestras consistentes, y por ende, representativas de la población objetivo en las métricas de código fuente LOC, McCC y WMC.

5.1.6. Conclusiones

A continuación, se interpretan los resultados informados discutiéndolos en el contexto de las PI y la literatura relacionada. Esta discusión se centra en los umbrales de métricas obtenidos y el instrumento de recolección de datos evaluado.

5.1.6.1. Umbrales de métricas

El análisis estadístico realizado mediante pruebas KS en la PI1 reveló diferencias significativas entre una colección vigente de Github y el *Qualitas Corpus* en la distribución de probabilidades de las métricas LOC, McCC y WMC. Esto plantea dudas acerca de la validez del *Qualitas Corpus* para conducir investigaciones actuales centradas en estas propiedades.

Por otra parte, al comparar los umbrales inferiores, medios y superiores de estas colecciones, se observó una notable disminución en los valores de referencia. Componentes de software categorizados como de bajo riesgo según los umbrales de Qualitas, actualmente serían de alto riesgo, es decir, métodos o clases que deberían ser refactorizados a mediano plazo. En este sentido, si los profesionales emplearan los umbrales derivados del *Qualitas Corpus* para monitorear el desarrollo de proyectos software en lugar de una colección actual, esto los podría llevar a producir código de baja calidad. A medida que el código aumenta en complejidad y tamaño, tiende a impactar negativamente en

la calidad externa de los sistemas, como la fiabilidad, la mantenibilidad o la seguridad [143].

5.1.6.2. Evaluación del instrumento de recolección de datos

La PI2 evaluó la efectividad de las estrategias de muestreo y actualización propuestas para generar colecciones representativas. Para ello, se extrajeron dos colecciones con proyectos actuales de Github empleando el instrumento de recolección de datos: una desde cero y una versión actualizada del *Qualitas Corpus*. Se compararon estadísticamente los umbrales de estas colecciones y las diferencias fueron marginales. No obstante, estos valores por sí solos no transmiten completamente las similitudes entre estas muestras. En la Fig. 32 se trazaron los umbrales de las tres muestras analizadas en el estudio, donde las curvas azul y naranja representan las muestras creadas con la herramienta y la verde los umbrales del *Qualitas Corpus*. Las curvas azul y naranja prácticamente se superponen de principio a fin, en cambio la verde tiene una distancia considerable de las demás.

En este estudio de minería se evaluó el instrumento de recolección de datos para construir y actualizar muestras representativas producido a partir de SUM4SOFT. Se analizaron gráfica y estadísticamente los umbrales de las colecciones obtenidas en tres métricas de código fuente: LOC, McCC y WMC presentando funciones de distribución acumuladas casi sin diferencias. Por lo tanto, en este contexto se demostró la efectividad de la herramienta JavaQ para la generación de muestras representativas.

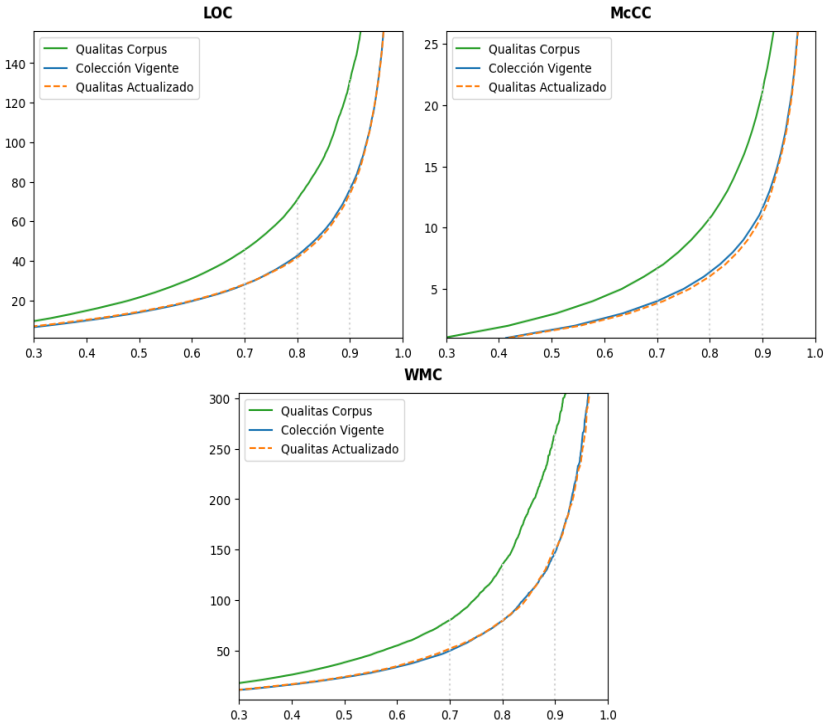

Fig. 32. Funciones de distribución acumulada de cada colección

5.2. Estudio de caso

Para validar los resultados obtenidos en el MRS reportado en la sección 5.1 y recibir mejoras en el modelo de proceso SUM4SOFT, se condujo un estudio de caso [144] con investigadores externos de la Universidad de Hamburgo.

5.2.1. Planificación

Se siguieron los lineamientos establecidos por Runeson et al. [144] para el diseño del protocolo del estudio de caso apuntando a dos objetivos. En primer lugar, replicar el MRS conducido y comparar la consistencia de los resultados obtenidos en ambas instancias. En segundo lugar, recibir sugerencias para mejorar modificar, eliminar o incorporar elementos a SUM4SOFT.

En este contexto se formularon las siguientes PI:

PI1. ¿La replicación presenta resultados consistentes con la investigación original? Motivación: Validar los resultados obtenidos en el MRS. Esto implica la efectividad del instrumento de recolección de datos para construir colecciones representativas.

PI2. ¿Qué cambios incorporar en SUM4SOFT? Motivación: Analizar las sugerencias recibidas e incluir las modificaciones que mejoren la claridad y relevancia de los elementos del modelo propuesto.

5.2.1.1. Diseño

El diseño de un estudio de caso involucra la definición del tipo de estudio de caso, las unidades de análisis y las actividades que desarrollarán los participantes y evaluadores. En este caso el estudio de caso se clasificó como evaluativo, dado que busca estudiar y perfeccionar el funcionamiento de un artefacto tecnológico en un entorno real.

El estudio se desarrolló en la Universidad de Hamburgo, en particular, en el grupo de investigación *Software Engineering and Construction Methods* (SWK), donde cinco de los investigadores miembros actuaron como participantes. La investigación se condujo en el marco de una estancia doctoral realizada en dicha institución. Se siguió una estructura de estudio de caso simple holístico [144], donde la unidad de análisis y caso corresponde al grupo de investigación SWK. A continuación, se describen las actividades propuestas para dirigir la investigación:

- **Reunión inicial**. Se compartió a los participantes del estudio los objetivos, las preguntas de investigación y el protocolo del estudio de caso.
- **Replicación del MRS**. Se proporcionaron el manuscrito del MRS incluyendo el instrumento de recolección de datos y sus instrucciones de uso. El documento se redactó cumpliendo con las recomendaciones de Gonzalez-Barahona & Robles [145] para facilitar la replicación de estudios en ISE. Dado que el estudio aún se encuentra en proceso de revisión, se brindó una versión preliminar. Asimismo, se incluyó la guía para la replicación de estudios en Ingeniería del Software de Carver et al. [146] y demás literatura relevante.

- **Recolección de datos.** Una vez recibidas las colecciones producidas por los investigadores, se programó otra reunión grupal para brindar una presentación guiada de SUM4SOFT al equipo. Terminada la reunión, se les envió un documento con el modelo completo y las especificaciones en el contexto del MRS (ver sección 4.3) junto con un cuestionario para capturar las sugerencias.
- **Análisis de resultados**. Se analizaron las colecciones obtenidas de la replicación y las respuestas al cuestionario, y se presentaron los resultados obtenidos.

5.2.1.2. Recolección de datos

Se emplearon dos métodos para la recolección de datos: análisis de documentos (independiente) y cuestionarios (directa) [144]. En cuanto al análisis de documentos, para responder la PI1 los participantes generaron dos colecciones con el paquete de replicación del MRS: una colección vigente (C) y una muestra actualizada del *Qualitas Corpus* (Qu), que posteriormente se compararon con los resultados reportados en el estudio original (ver sección 5.1).

El cuestionario de la Tabla 41 se diseñó para obtener una retroalimentación sobre las tareas y roles del modelo de proceso en el marco de las actividades A1.1 "Especificar las fuentes", A1.2 "Diseñar la estrategia de muestreo" y A2 "Ejecutar el instrumento de recolección de datos". En total, se formularon cuatro preguntas cerradas con posibilidad de dejar comentarios: dos sobre la claridad y relevancia de las tareas en el contexto de las actividades, una para la inclusión de nuevas tareas y otra para la modificación de los roles.

Tabla 41. Cuestionario para sugerencias a SUM4SOFT

Elemento evaluado	Tipo de preguntas	Preguntas
Tareas	Claridad	¿La tarea es clara, precisa? Comentarios
	Relevancia	¿La tarea es relevante para la actividad? Comentarios
	Inclusión de tareas	¿Incluiría nuevas tareas? Comentarios
Roles	Modificación de roles	¿Modificaría el rol? Comentarios

Las preguntas sobre claridad apuntaron a modificar las tareas, mientras que las de relevancia a definir si mantenerlas en SUM4SOFT. Por otra parte, las preguntas sobre la inclusión de tareas y modificación de roles buscaron recolectar sugerencias de los participantes para incorporar nuevas tareas y ajustar los roles del modelo, respectivamente.

El cuestionario se estructuró en cuatro secciones, una para cada actividad y la cuarta dedicada a los roles. Las preguntas de claridad y relevancia se organizaron de manera ascendente según el número de tarea en las secciones de la actividad a la cual pertenecen, por ejemplo, en la sección correspondiente a la A1.1 se presentaron primero las preguntas sobre la tarea 1 "Determinar los datos a obtener", segundo las de la tarea 2 "Describir lo métodos de búsqueda" y así sucesivamente. La Fig. 33 muestra como ejemplo las preguntas confeccionadas para la tarea 1 "Determinar los datos a obtener ".

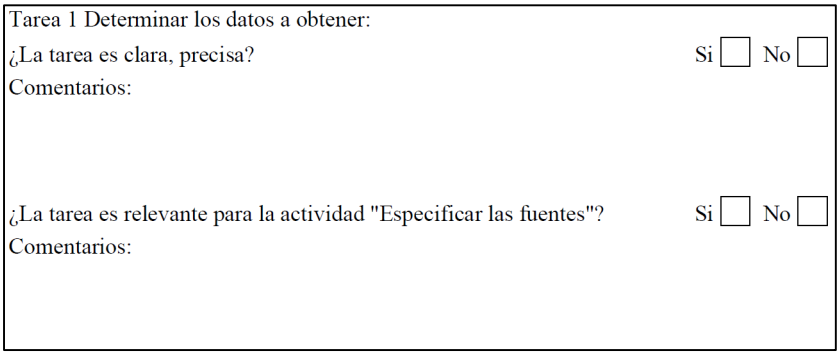

Fig. 33. Preguntas del cuestionario sobre la tarea 1 "Determinar los datos a obtener"

Al final de cada sección de actividad se incluyeron las preguntas orientadas a sumar nuevas tareas en el modelo de proceso. La Fig. 34 muestra la pregunta formulada para la actividad A1.1.

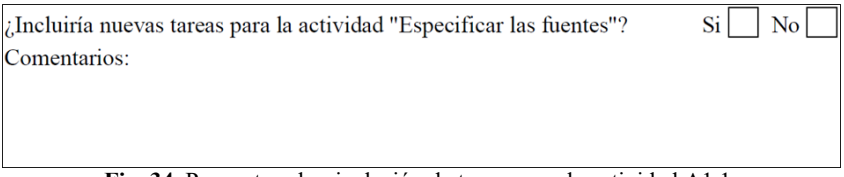

Fig. 34. Pregunta sobre inclusión de tareas para la actividad A1.1

En la Fig. 35 se muestran las preguntas para la modificación de los roles, la cual se posicionó en la cuarta y última sección del cuestionario.

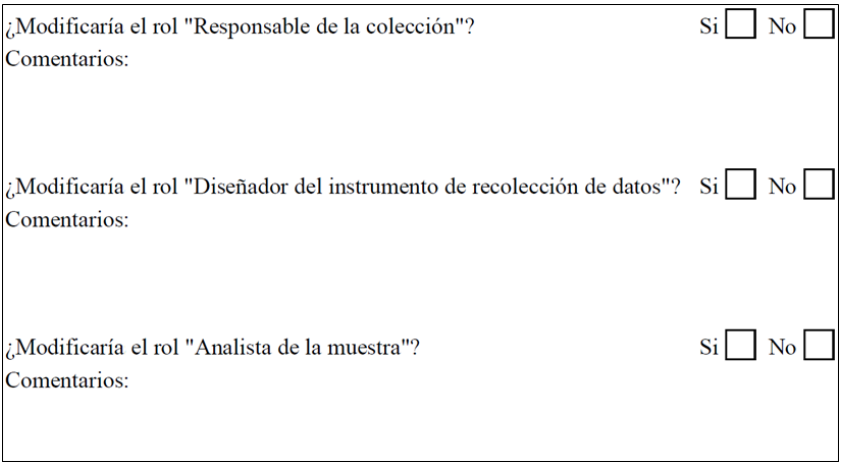

Fig. 35. Preguntas sobre modificación de los roles

5.2.1.3. Análisis de datos

A continuación, se presentan los procedimientos empleados para analizar los documentos y respuestas del cuestionario para responder las PI.

> **PI1. ¿La replicación presenta resultados consistentes con el estudio original?**

De manera similar al protocolo de análisis empleado en el MRS (ver sección 5.1.4), se compararon los umbrales de referencia de LOC, McCC y WMC obtenidos con la colección vigente del estudio original (Co) y las colecciones generadas en la replicación: una colección vigente (Cr) y una muestra actualizada del *Qualitas Corpus* (Qur). Al igual que el MRS, el nivel de significación se fijó en 0.05 y se ejecutó la prueba KS evaluando la hipótesis nula de la Fig. 31 para comparar estadísticamente la consistencia entre las muestras. Se formularon dos conjuntos de hipótesis complementarias adaptadas al estudio de caso, por un lado, la comparación entre la colección actual del MRS (Co) y la colección generada con la misma estrategia de muestreo en la replicación (Cr):

> **H_0: Co(X) = Cr(X).** La distribución de datos de la colección actual del MRS y la colección actual de la replicación provienen de la misma población.

H_1: Co(X) ≠ Cr(X). La distribución de datos de la colección actual del MRS y la colección actual de la replicación no provienen de la misma población.

Por otro lado, la comparación entre la colección actual del MRS (Co) y la colección generada con la estrategia de actualización aplicada sobre el *Qualitas Corpus* en la replicación (Qur):

H_0: Co(X) = Qur(X). La distribución de datos de la colección actual del MRS y la actualización del Qualitas de la replicación provienen de la misma población.
H_1: Co(X) ≠ Qur (X). La distribución de datos de la colección actual del MRS y la actualización del Qualitas de la replicación no provienen de la misma población.

PI2. ¿Qué cambios incorporar en SUM4SOFT?

En la PI2, se resumieron los comentarios cuantitativos y cualitativos para determinar qué tareas y roles del modelo modificar, eliminar y agregar. Dependiendo del contexto. se analizaron los comentarios cualitativos para determinar cómo mejorarlas. Las modificaciones se priorizaron según el número de respuestas negativas a las preguntas cerradas y en función de los comentarios. Por lo tanto, aunque una pregunta cerrada registrara una cantidad considerable de respuestas negativas, si los comentarios cualitativos no aportan razones satisfactorias, la tarea o el rol se mantendrá sin cambios.

5.2.2. Ejecución

El 25 de abril del 2024, se organizó la reunión inicial con los participantes involucrados en el estudio. Durante esta sesión, se compartieron los objetivos generales, las preguntas de investigación y el protocolo diseñado para la ejecución del estudio de caso. Posteriormente, se les envió un correo electrónico con el manuscrito sin publicar del MRS adjuntando la siguiente consigna:

"The main goal is to replicate the study and determine the consistency between the original results and the iteration with minimal intervention from my part. Specifically, I would like you to generate new current datasets (current sample and qualitas updated) and run the data analysis using the replication package provided in the paper. When you

have the datasets ready, please let me know and I will provide you a Google Drive's link to upload them.

For your reference, I have included snippets from Carver et al. (2014) that outline guidelines for conducting study replications…"

El 10 de mayo de 2024 se recibieron las colecciones y el reporte de ejecución del instrumento de recolección de datos, el cual detalló que este se ejecutó el 8 de mayo, obteniendo como resultado un marco muestral de 18562 proyectos Java que la segunda etapa de filtrado redujo a 944 sujetos. El tiempo de duración de la rutina fue en total: 5 horas, 18 minutos y 35 segundos. La diferencia en la duración con respecto al MRS, se atribuye al uso de un solo *token* de acceso durante el proceso de extracción.

El 23 de mayo del 2024 se realizó una breve presentación guiada de SUM4SOFT al grupo. Esta duró 30 minutos y proporcionó una visión general de la estructura, funcionamiento y aplicaciones del modelo de proceso. Cabe mencionar que uno de los participantes iniciales del estudio no asistió a la reunión ni a las actividades posteriores por cuestiones de salud. Terminada la sesión, se les envió por correo electrónico: el documento con la descripción de SUM4SOFT, junto con las especificaciones fijadas para conducir el MRS (ver sección 4.3), y el enlace al cuestionario de la Tabla 41. El cuestionario tuvo la siguiente descripción:

"The purpose of this evaluation is to analyze the tasks and roles defined for the process model SUM4SOFT to improve the clarity and relevance of the model specification and facilitate its adoption by identifying improvement opportunities. The questionnaire has four sections: one for each activity in SUM4SOFT and a fourth dedicated to the roles.

A document with the description of the model's activities, roles, tasks and work products along with its application in an empirical study is provided.

To perform the evaluation, and tackle sections one through three:
1) For each task, read its description and interpret its place within the SPEM diagrams.
2) For each closed question, answer yes or no depending on the task or activity.
3) Add comments about the modification it should be applied according to the context of the question.

To tackle section four:
1) For each role, read its description and analyze the organizational diagram.
2) For each closed question, answer yes or no depending on the role.
3) If you responded yes, in the comments provide the modifications that should be applied to the role."

Cuando se reunieron todas las respuestas, se procedió a aplicar el protocolo de análisis de datos. En particular, el procesamiento de las respuestas del cuestionario implicó: resumir los resultados en inglés, responder las PI y traducir los comentarios al español.

Tanto las colecciones obtenidas para la replicación como las respuestas del cuestionario generadas por los participantes se encuentran disponibles en el siguiente enlace.[26]

5.2.3. Resultados

5.2.3.1. PI1. ¿La replicación presenta resultados consistentes con el estudio original?

Se replicó el proceso de cálculo y se emparejaron los umbrales de las colecciones Co y Cr, y Co y Qur siguiendo la metodología empleada en el MRS. La Tabla 42 presenta los valores de los umbrales inferior, medio y superior correspondientes a las métricas LOC, McCC y WMC de las tres colecciones analizadas. Al realizar la comparación entre los umbrales de Co y Cr, se observaron diferencias mínimas entre las dos colecciones. En particular, los umbrales inferior y medio de LOC coincidieron exactamente, registrando valores de 29 y 43 respectivamente. En la misma línea, los valores de referencia de Co y Qur tampoco difirieron demasiado entre sí, siendo WMC la métrica con mayor distancia.

[26] https://bit.ly/datasets_case_study

Tabla 42. Umbrales de las colecciones actuales (C) de cada estudio

Estudio	Colección	Métrica	Inferior	Medio	Superior
MRS	Co	LOC	29	43	77
		McCC	4	7	12
		WMC	50	80	148
Replicación	Cr	LOC	29	43	78
		McCC	4	6	11
		WMC	46	74	145
	Qur	LOC	27	41	72
		McCC	4	6	11
		WMC	55	86	155

Posteriormente, se aplicó la prueba KS para comparar las distribuciones de los umbrales en las muestras Co y Cr, y Co y Qur para cada métrica. Los resultados obtenidos que se muestran en la Tabla 43, revelan entre Co y Cr p-valores superiores a 0.99 para todas las variables evaluadas, lo que indica la ausencia de diferencias significativas en las distribuciones de umbrales entre las colecciones. Lo mismo se determinó en la prueba KS entre Co y Qur, con la particularidad que el p-valor de WMC fue 0.85.

Tabla 43. Prueba KS entre los umbrales de las colecciones de la replicación y el MRS

Colección MRS	Métrica	KS	P-valor
Cr	LOC	0.01	1
	McCC	0.02	1
	WMC	0.02	0.99
Qur	LOC	0.01	1
	McCC	0.02	1
	WMC	0.03	0.85

Dado que los p-valores sobrepasaron el nivel de significación en todas las métricas consideradas, se aceptó la hipótesis nula en ambos casos. Por lo tanto, la distribución de datos de la muestra actual del MRS, la muestra actual y la actualización del Qualitas de la replicación provienen de la misma población. Por lo tanto, se demostró que el instrumento de recolección de datos permitió generar muestras consistentes y representativas de la población objetivo en las métricas LOC, McCC y WMC, confirmando los resultados originales del MRS.

5.2.3.2. PI2. ¿Qué cambios incorporar en SUM4SOFT?

A continuación se presentan las respuestas a las preguntas del cuestionario de la Tabla 41 de acuerdo con el tipo de pregunta.

¿La tarea es clara, precisa?

La primera pregunta buscó identificar qué tareas de SUM4SOFT podrían no ser lo suficientemente claras o entendibles para el usuario. En la Tabla 44 se registraron las respuestas a las preguntas cerradas del cuestionario orientadas a la claridad. Al analizar cuantitativamente las respuestas correspondientes a las tareas de A1.1, aunque en algunos casos los participantes incluyeron comentarios para mejorar la calidad de ellas, se observó que la mayoría las consideraron entendibles. Uno de ellos declaró lo siguiente acerca de la tarea 2 "Describir los métodos de búsqueda": "Ni el diagrama funcional de la actividad Especificar las fuentes ni la enumeración inicial de productos de trabajo mencionan la salida de la tarea Describir los métodos de búsqueda. De acuerdo con la tarea, se deberían obtener las descripciones de la forma de acceso y restricciones de cada fuente".

Tabla 44. Respuestas sobre claridad

Actividades	Tareas	n = 4	
		#Si	#No
Especificar las fuentes (A1.1)	Determinar los datos a obtener	4	0
	Describir los métodos de búsqueda	3	1
	Definir criterios de selección de fuentes	3	1
	Seleccionar fuentes de información	4	0
Diseñar la estrategia de muestreo (A1.2)	Especificar la población objetivo	2	2
	Definir estrategia de muestreo	2	2
	Definir estrategia de actualización	1	3
	Construir el instrumento de recolección de datos	4	0
Ejecutar el instrumento de recolección de datos (A2)	Extraer la colección	4	0
	Actualizar la colección	3	1

En cuanto a las tareas de A1.2, fueron las que contabilizaron un mayor número de respuestas negativas. En particular, la tarea 3 "Definir estrategia de actualización" acumuló tres respuestas negativas, el número más alto en las preguntas cerradas. Además, en casi todos los casos se recibieron comentarios acerca de cuestiones técnicas que deberían ser abordadas para brindar una descripción precisa. Por ejemplo, en la tarea 1 "Especificar la población objetivo" preguntaron: "¿Que sería un criterio de selección de proyecto? ¿Cómo se instrumentan?". En la tarea 2 "Definir la estrategia de muestreo"

mencionaron: "La selección del tamaño de la muestra debe ser objetiva, de ella depende la solidez de los resultados, a menos que existan restricciones inherentes a los datos", "¿Cómo determinar el tamaño de la muestra?". Por su parte, en la tarea 3 "Definir estrategia de actualización" muchos solicitaron la clarificación acerca de las pautas que deben ser tenidas en cuenta en el diseño de la estrategia de actualización.

Con respecto a los resultados de las tareas de A2, las preguntas cerradas recibieron respuestas en su mayoría positivas. No obstante, uno de los participantes comentó acerca de la tarea Actualizar la colección: "No estoy seguro a qué refiere la expresión condición del disparador, entiendo que estará relacionado con la estrategia de actualización, pero podría explicarse con más detalle para evitar confusiones".

¿La tarea es relevante?

La segunda pregunta cerrada apuntó a determinar las tareas que son relevantes dentro del proceso de muestreo y actualización de colecciones. En la Tabla 45 se detallaron las respuestas a las preguntas cerradas sobre relevancia para cada una de las tareas. Exceptuando la tarea 2 "Actualizar la colección", todas las preguntas sobre relevancia obtuvieron respuestas positivas. Dentro de los comentarios que manifestaron dudas respecto a esta tarea argumentaron: "No encuentro diferencias entre Actualizar la colección y Extraer la colección, son prácticamente iguales, eliminaría una de ellas para evitar la redundancia de tareas".

Tabla 45. Respuestas sobre relevancia

Actividades	**Tareas**	**N = 4**	
		#Si	**#No**
Especificar las fuentes (A1.1)	Determinar los datos a obtener	4	0
	Describir los métodos de búsqueda	4	0
	Definir criterios de selección de fuentes	4	0
	Seleccionar fuentes de información	4	0
Diseñar la estrategia de muestreo (A1.2)	Especificar la población objetivo	4	0
	Definir estrategia de muestreo	4	0
	Definir estrategia de actualización	4	0
	Construir el instrumento de recolección de datos	4	0
Ejecutar el instrumento de recolección de datos (A2)	Extraer la colección	4	0
	Actualizar la colección	2	2

¿Incluirías nuevas tareas?

De las sugerencias de inclusión de tareas en la Tabla 46, se destacaron dos en particular. Una de ellas describió una tarea para verificar el instrumento de recolección de datos, es decir, formalizar una tarea que consista en hacer una prueba piloto del instrumento para la detección de errores antes de enviar la herramienta a producción. La segunda también introdujo una tarea de verificación de la colección en el contexto de A2, la tarea involucraría revisar la colección actualizada (resultado de la tarea 2 "Actualizar la colección") para detectar desviaciones con respecto a una colección obtenida con la estrategia de muestreo (resultado de la tarea 1 "Extraer la colección").

Tabla 46. Respuestas sobre inclusión de nuevas tareas

Actividades	N = 4	
	#Si	#No
Especificar las fuentes (A1.1)	0	4
Diseñar la estrategia de muestreo (A1.2)	2	2
Ejecutar el instrumento de recolección de datos (A2)	1	3

¿Modificarías el rol?

Con respecto a la modificación de los roles, se recibieron dos respuestas para modificar el rol de "Analista de la muestra" (ver Tabla 47). En primer lugar, incluir la responsabilidad de garantizar la calidad de los datos recolectados, verificando su consistencia y precisión. En segundo lugar, añadir como requisito para este rol tener conocimientos metodológicos sobre los enfoques de extracción de los datos, para asegurar la solidez de las colecciones producidas por el instrumento de recolección.

Tabla 47. Respuestas sobre modificación de roles

Roles	N = 4	
	#Si	#No
Responsable de la colección	0	4
Diseñador de los instrumentos de recolección de datos	0	4
Analista de la muestra	2	2

5.2.4. Conclusiones

A continuación, se interpretan los resultados del estudio de caso, se responden a las PI y se ofrece una conclusión del capítulo.

5.2.4.1. Replicación MRS

Para validar los resultados obtenidos originalmente en el MRS a partir de las estrategias de muestreo implementadas en el instrumento de recolección de datos, se condujo una replicación del mismo trabajo enmarcado en un estudio de caso. Al igual que en el estudio original, los investigadores externos extrajeron dos colecciones con proyectos actuales de Github empleando el paquete de replicación: uno desde cero y una versión actualizada del *Qualitas Corpus*. A partir de ellos se derivaron los umbrales de las métricas LOC, McCC y WMC, y se contrastaron con los obtenidos en el estudio original.

El análisis estadístico de los umbrales de referencia obtenidos en la replicación y el estudio original mostraron resultados consistentes para las métricas LOC, McCC, y WMC. Se observaron diferencias mínimas entre las colecciones Co y Cr, junto con p-valores mayores a 0.85 en la prueba KS, sugiriendo que las distribuciones de los datos en ambos estudios provienen de la misma población. Este hallazgo confirmó la validez del instrumento de recolección de datos JavaQ, asegurando su capacidad para generar muestras representativas independientemente del equipo de trabajo. Por lo tanto, se ratificó la efectividad de las estrategias de muestreo y actualización aplicadas en el MRS, aportando robustez a los resultados originales y habilitando la generalización a contextos similares.

5.2.4.2. Mejora SUM4SOFT

La PI2 tuvo como objetivo captar sugerencias del grupo de investigación SWK para perfeccionar las tareas y roles del modelo de proceso propuesto. Para ello, se diseñó un cuestionario orientado a estudiar las tareas en términos de su claridad y relevancia en el contexto de las actividades, y recibir sugerencias de mejora. Las respuestas se analizaron cuantitativa y cualitativamente y se sintetizaron en las Tabla 48 y Tabla 49.

Tabla 48. Modificaciones de tareas incluidas en el modelo definitivo

Actividades	Tareas	Cambios	Motivación
Especificar las fuentes (A1.1)	Describir los métodos de búsqueda	Modificada	Agregar el producto de trabajo: Métodos de búsqueda
Diseñar la estrategia de muestreo (A1.2)	Especificar la población objetivo	Modificada	Aclarar cómo se instrumentan los criterios de selección
	Definir estrategia de muestreo	Modificada	Proporcionar información sobre el tamaño de la muestra
	Definir estrategia de actualización	Modificada	Enumerar los componentes de la estrategia de actualización
	Probar el instrumento de recolección de datos	Agregada	Verificar el instrumento antes de enviarlo a producción
Ejecutar el instrumento de recolección de datos (A2)	Actualizar la colección	Modificada	Comprobar que no existen desviaciones

El primer cambio se realizó sobre la tarea 2 "Describir los métodos de búsqueda" de la A1.1, donde se agregó el producto de trabajo: métodos de búsqueda. Este elemento comprende las descripciones de la forma de acceso y restricciones de cada fuente, y se omitió en la versión anterior del modelo pese a su importancia en la confección de los criterios de selección y las guías de acceso a fuentes.

Tabla 49. Modificaciones de roles incluidas en el modelo definitivo

Rol	Cambios	Motivación
Analista de la muestra	Modificada	Ampliar responsabilidades en el control de la calidad de los datos

Con respecto a la A1.2, se modificaron tres de ellas y se incorporó la tarea 5 "Probar el instrumento de recolección de datos". Para la tarea 1 "Especificar la población objetivo", además de explicar que sirve para caracterizar los sujetos de la población, se aclaró que esto operativamente se implementa a través de la definición de umbrales. En la tarea 2 "Definir la estrategia de muestreo", se proporcionó información relativa a la definición del tamaño de la muestra, brindando ejemplos de herramientas para calcularlo. En la tarea 3 "Definir estrategia de actualización", se enumeraron los componentes necesarios para diseñar una estrategia de actualización, como ser: el disparador de la actualización, las políticas de inclusión de nuevos proyectos, el tratamiento de los proyectos descartables, entre otras. La cuarta

modificación añadió la tarea 5 "Probar el instrumento de recolección de datos" para verificar el funcionamiento del instrumento y descubrir errores antes de utilizarlo en la investigación.

El último cambio de la lista de tareas se aplicó a la tarea 2 "Actualizar la colección" de A2, donde se agregó la comprobación de la validez de la colección actualizada. Es un paso adicional para controlar que la salida de la tarea no presente desviaciones al comparar con una muestra generada con la estrategia de muestreo.

En cuanto a los roles, la Tabla 49 muestra los cambios realizados al rol de "Analista de la muestra" a partir de las sugerencias de los participantes. A las responsabilidades de diseñar las estrategias de muestreo se añadió la de garantizar la calidad de los datos extraídos, ampliando las competencias del rol en ese sentido.

De acuerdo con la
Fig. **36**, a lo largo del capítulo se abordó la evaluación de SUM4SOFT. Este involucró un MRS y un estudio de caso conducido en el grupo de investigación SWK de la Universidad de Hamburgo en el marco de una estancia doctoral. El MRS evaluó la efectividad del instrumento de recolección de datos JavaQ, producido por el modelo de proceso para la generación de colecciones de proyectos representativas en el contexto de una investigación de Ingeniería del Software. Por otra parte, el estudio de caso ratificó los resultados originales del MRS por medio de una replicación, y operacionalizó el perfeccionamiento de las tareas y roles de SUM4SOFT, los cuales fueron incluidos en la versión definitiva del artefacto para facilitar su uso en grupos externos.

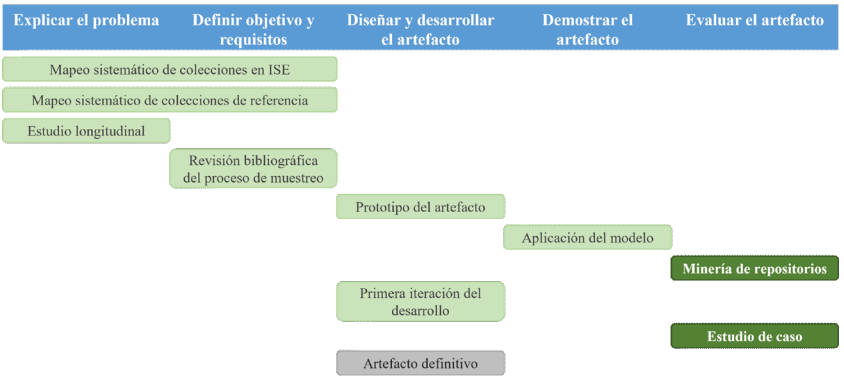

Fig. 36. Actividades abarcadas en el CAPITULO 5

CAPITULO 6

Conclusiones

El capítulo presenta las conclusiones de la investigación desarrollada en esta tesis doctoral y los trabajos futuros. Este se organiza de la siguiente manera: la sección 6.1 enumera las actividades realizadas para conseguir los objetivos y las contribuciones propuestas al inicio de la investigación, la sección 6.2 describe otras contribuciones aportadas, la sección 6.3 propone trabajos futuros que pueden surgir a través de los resultados presentados y la sección 6.4 lista los artículos publicados para diseminar los hallazgos producidos.

6.1. Objetivos de la tesis abordados

En el Capítulo 1 se analizaron las problemáticas presentes en las colecciones de proyectos software utilizadas en la Ingeniería del Software, abordando tanto aspectos relacionados con la difusión de los instrumentos de investigación como cuestiones metodológicas. Por ello, se planteó como objetivo principal de esta tesis el desarrollo de un modelo de proceso para la construcción, actualización y curaduría de colecciones de proyectos software, con el fin de apoyar la realización de estudios en la Ingeniería del Software Empírica. Para alcanzar este objetivo, se empleó el marco de trabajo de la ciencia de diseño, planificando las siguientes tareas:

> **Explicar el problema.** Para profundizar en las problemáticas planteadas inicialmente en la investigación, se condujeron tres estudios empíricos: dos mapeos sistemáticos y un estudio longitudinal. El primer mapeo (sección 3.1) estudió los criterios de selección de los proyectos software, los metadatos obtenidos, las herramientas de extracción y los análisis estadísticos aplicados en la construcción de las muestras utilizadas en la Ingeniería del Software Empírica. Este análisis se amplió en un segundo mapeo (sección 3.2) enfocado principalmente en las colecciones de referencia. Los estudios secundarios (sección 3.3) confirmaron las

problemáticas descriptas en el CAPITULO 1: la falta de estrategias de selección de proyectos estandarizadas, la omisión de la vigencia como criterio de selección, la ausencia del muestreo probabilístico y las restricciones de acceso, tanto a las colecciones como a las herramientas empleadas en la experimentación con ellas. A raíz de esta última, se llevó a cabo un estudio longitudinal (sección 3.4) para evaluar el impacto del tiempo en una población de proyectos y se demostró que en el transcurso de seis años la misma presentó cambios significativos, evidenciando la importancia de emplear colecciones vigentes para obtener resultados generalizables.

Definir objetivo y requisitos. Se analizaron guías de referencia sobre la construcción de muestras en la Ingeniería del Software con el fin de identificar los requisitos necesarios para diseñar el modelo de proceso (sección 4.1). Además, se determinaron las especificaciones necesarias para instrumentar dicho proceso con los resultados obtenidos en los mapeos sistemáticos: los criterios de selección de proyectos, los metadatos y las herramientas reportadas en la literatura. Los criterios considerados en la selección de proyectos Java de código abierto fueron: la utilización de herramientas para dar soporte a procesos, el tamaño, la historia de desarrollo, la popularidad, la colaboración y la actividad reciente. Los metadatos a recopilar del software fueron las métricas de tamaño y complejidad del código, siendo SourceMeter, Understand, iPlasma, entre otras; algunas herramientas apropiadas para obtenerlos.

Diseñar y desarrollar el artefacto. Partiendo de los resultados del relevamiento de las guías de referencia se desarrolló el modelo de proceso SUM4SOFT (sección 4.2). El artefacto abarcó la definición de las actividades y tareas consideradas como relevantes para el muestreo de proyectos software. SUM4SOFT se modeló contemplando cuatro perspectivas: de comportamiento, funcional, organizacional e informacional, y se diagramó con el lenguaje SPEM 2.0. Por lo tanto, se incluyó la especificación de los productos de trabajo y los roles asignados a cada tarea y actividad.

Demostrar el artefacto. Para probar SUM4SOFT, presentar su estructura y mostrar su funcionamiento en un escenario de aplicación se generó el instrumento de recolección de datos JavaQ a partir del modelo de proceso (sección 4.3). Los elementos del modelo se definieron de acuerdo con las especificaciones recolectadas en los estudios secundarios. Dentro de ellas se destaca

el diseño de las estrategias de muestreo y de actualización. La primera implementó un enfoque aleatorio estratificado generando los estratos con el algoritmo de agrupamiento k-means para seleccionar los proyectos. Para la segunda se planificó la siguiente estrategia: 1) disparar la actualización cada 365 días, 2) actualizar a su versión más reciente aquellos proyectos en la colección con actividad reciente, 3) separar los proyectos sin actividad, 4) reemplazar estos últimos con proyectos activos que pertenezcan al mismo estrato. Posteriormente, se obtuvo el instrumento de recolección de datos el cual se ejecutó para extraer una colección de proyectos.

Evaluar el artefacto. En la última actividad se propusieron dos estudios para evaluar la eficacia del artefacto en la resolución del problema planteado: un estudio de minería de repositorios software y un estudio de caso. En el estudio de minería (sección 5.1), se utilizó el instrumento de recolección de datos JavaQ, construido en la demostración del artefacto, para recolectar tres colecciones de proyectos y con ellas se generaron umbrales de referencia para determinar la calidad de los componentes de sistema en métricas de complejidad y tamaño del código fuente. Por otra parte, en un estudio de caso conducido en la Universidad de Hamburgo (sección 5.2) se dirigió una replicación del mismo estudio de minería para validar los resultados originales y una evaluación de las tareas y roles de SUM4SOFT. El análisis estadístico de los umbrales de calidad del estudio de minería demostró la efectividad de las estrategias de muestreo para la construcción de muestras representativas, que la replicación posteriormente confirmó aportando robustez y generalidad a los resultados originales. La segunda parte del estudio de caso operacionalizó la modificación de tareas y roles de SUM4SOFT para mejorar la claridad y relevancia de ellas, y facilitar su uso e implantación en grupos externos.

De esta manera, luego de finalizar las actividades propuestas en el marco de ciencia de diseño, la contribución principal de este trabajo de tesis es el desarrollo de herramientas para el soporte de estudios en la Ingeniería del Software Empírica. En primer lugar, el modelo de proceso para la construcción y actualización de muestras de proyectos software SUM4SOFT. En segundo lugar, los productos de trabajo de SUM4SOFT obtenidos en el contexto de un estudio de minería (Fig. 40, Fig. 41 y Fig. 42 del APENDICE D): una colección de proyectos software y el instrumento de recolección de datos JavaQ con las

herramientas para mantener la colección facilitando la replicación de los resultados. Por lo tanto, la hipótesis planteada en el CAPITULO 1: "Contar con un modelo de proceso para la construcción, actualización y curaduría de colecciones de proyectos software es una estrategia para generar resultados replicables y generalizables en estudios de la Ingeniería del Software Empírica" luego de analizar los resultados obtenidos en esta investigación doctoral se puede afirmar como correcta.

6.2. Contribuciones adicionales

En el proceso de consecución del objetivo general y comprobación de la hipótesis planteada se condujeron una serie de estudios en las actividades planificadas que aportaron las siguientes contribuciones adicionales:

Criterios para la selección de proyectos software. A partir de los mapeos sistemáticos se determinaron las características deseables en los proyectos software para construir una colección curada y representativa en la industria del software. Dentro de estas características se encuentran: uso de herramientas para dar soporte a procesos, tamaño, historia de desarrollo, popularidad, colaboración y actividad reciente. A través de métricas de repositorio; por ejemplo, número de incidentes (indicios de gestión de incidentes), cantidad de colaboradores (colaboración), tiempo de desarrollo (historia), número de estrellas (popularidad), cantidad de confirmaciones en el último mes (actividad reciente); se pueden cuantificar estas características y detectar los proyectos apropiados para conformar una muestra.

La vigencia en la Ingeniería del Software. En pocas palabras, la vigencia refiere a la medida en que una muestra recopilada en el pasado continúa siendo representativa de la misma población en el presente. Dada la evolución constante que sufre el software a lo largo del tiempo para mantenerse funcional y relevante, resulta lógico que una colección de proyectos software utilizada en estudios empíricos también establezca estrategias de actualización para prolongar su vigencia. Sin embargo, se desconocen estudios que discutan al respecto y evidencia de ello es el número de autores citados en este documento que optaron por trabajar con muestras creadas hace más de siete años. En este sentido, tanto el estudio

longitudinal como el de minería de repositorios recolectaron evidencias acerca de la importancia de la vigencia y el impacto provocado en las muestras.

Estrategias para la generación y actualización de colecciones. Para la generación de las colecciones empleadas en el estudio de minería se implementaron dos estrategias de muestreo (secciones 4.3.2.2 y 4.3.2.3): una para la extracción de una muestra y otra para la actualización. Ambas técnicas reunieron repositorios actuales de Github aplicando una forma de muestreo aleatorio estratificado a través del algoritmo de agrupamiento k-means. El estudio demostró que estas estrategias son viables para la generación de colecciones representativas.

Valores de referencia para métricas del código fuente. En el estudio de minería (sección 5.1) se calcularon tres valores de referencia de las métricas de código fuente LOC, McCC y WMC con el método basado en datos propuesto por Alves et al. [130]. Estos valores delimitan cuatro niveles de riesgo que puede registrar un componente de sistema (método o clase): riesgo bajo, riesgo moderado, riesgo alto y riesgo muy alto; indicando el nivel de atención requerido en cada caso. Dicho esto, se obtuvieron los valores de referencia (inferior, medio y superior): 29, 43 y 77 para LOC; 4, 7 y 12 para McCC; y 50, 80 y 148 en WMC.

6.3. Trabajos futuros

A partir de los hallazgos obtenidos durante esta investigación doctoral, se proponen las siguientes ideas para abordar las limitaciones observadas, continuar con trabajos que contribuyan a la línea actual, y explorar los conceptos abordados en otras temáticas:

Validación de SUM4SOFT en otros contextos. Si bien el modelo de proceso se validó inicialmente en un estudio de minería y posteriormente en un estudio de caso, utilizando el instrumento de recolección de datos JavaQ para generar las colecciones, sería interesante probar SUM4SOFT en un contexto de aplicación diferente al de métricas del código fuente. El objetivo consistirá en monitorear la utilización de SUM4SOFT en un nuevo estudio empírico diseñado por investigadores externos para evaluar los productos de trabajo generados. En particular, se considerarán

como posibles candidatos el Instituto Superior de Ingeniería de Software de Tandil (ISISTAN) para abordar los sistemas de recomendación [147] y el grupo de investigación *Applied Software Technology* (TAM) de la Universidad de Hamburgo para estudiar los sistemas de seguimiento de incidencias [148]. Esto permitirá verificar la aplicabilidad de SUM4SOFT en una variedad más amplia de escenarios proporcionándole mayor robustez al modelo diseñado.

Estrategias de actualización de colecciones. En la implementación de JavaQ se diseñó una estrategia de actualización basada en el algoritmo de agrupamiento k-means, aplicada tanto en el estudio de minería de repositorios y el estudio de caso para actualizar el *Qualitas Corpus*. Aunque este enfoque demostró capacidad generando versiones actualizadas del *Qualitas Corpus* que sean representativas de una población actual de proyectos, la misma podría no ser igualmente efectiva en otros escenarios. En este sentido, se explorarán nuevas estrategias de actualización que se ajusten mejor a los dominios propuestos en el punto anterior, como estudios sobre sistemas de recomendación y sistemas de seguimiento de incidencias.

Extensión de la vigencia. La pérdida de la vigencia de las colecciones de software es un problema que afecta no solo a las métricas de código fuente y de repositorio, sino a los proyectos en su conjunto. Por lo tanto, resulta lógico extender el estudio de la vigencia a otros tópicos de la Ingeniería del Software en los cuales podría ser aplicable, por ejemplo, las pruebas del software, la seguridad, refactorización del código, entre otras. De esta manera, se puede concientizar a los investigadores y profesionales acerca de los riesgos de trabajar con muestras no vigentes.

6.4. Difusión de resultados

Esta sección resume los artículos académicos publicados durante la investigación desarrollada para esta tesis doctoral.

6.4.1. Publicaciones

- Carruthers, J. A., Diaz-Pace, J. A., & Irrazábal, E. (2024). A longitudinal study on the temporal validity of software samples. Information and Software Technology, 168, 107404. https://doi.org/10.1016/j.infsof.2024.107404

  Estudio longitudinal. Este artículo estudió el fenómeno de la vigencia en la Ingeniería del Software Empírica y la actividad de los proyectos software. Para ello, se capturaron 72 instantáneas de una misma población de proyectos en Github y se analizó su evolución en siete métricas de repositorio.

- Carruthers, J. A., Diaz-Pace, J. A., & Irrazabal, E. A. (2022). How are software datasets constructed in Empirical Software Engineering studies? A systematic mapping study. 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 442–450. https://doi.org/10.1109/SEAA56994.2022.00075

  Segundo mapeo sistemático. Dado que un número considerable de estudios primarios seleccionados en el primer mapeo sistemático realizaron su investigación con colecciones de referencia, se consideró pertinente abordarlas y complementar los resultados reportados inicialmente. Además, se estudiaron las edades de estas colecciones y las motivaciones de creación.

- Carruthers, J. A., Diaz-Pace, J. A., & Irrazabal, E. A. (2022). A Systematic Mapping Study of Empirical Studies Performed with Collections of Software Projects. Computación y Sistemas, 26(4). https://doi.org/10.13053/cys-26-4-3981.

  Primer mapeo sistemático. Este estudio permitió obtener un panorama inicial sobre el uso de colecciones de proyectos en la Ingeniería del Software Empírica. Se identificaron los criterios de selección de los proyectos software, sus características, los metadatos del código fuente obtenidos, las herramientas de extracción y los análisis estadísticos practicados sobre los datos.

- Carruthers, J. A. (2022). Open-Source Software Projects Curating Model for Empirical Software Engineering Studies. Anais Do XXV Congresso Ibero-Americano Em Engenharia de Software (CIbSE 2022), 416–423. https://doi.org/10.5753/cibse.2022.20992 En el Simposio Doctoral celebrado en la Conferencia CibSE del 2022 se presentó un plan preliminar con las ideas centrales de la

investigación, y un boceto de la planificación y enfoque metodológico.

6.4.2. Publicaciones en proceso

- Carruthers, J. A., Lezcano Airaldi, A., Diaz-Pace, J. A., & Irrazábal, E. Building and updating software datasets: an empirical assessment. Empirical Software Engineering.

  Estudio de minería de repositorios. El estudio demostró la utilidad de un instrumento de recolección de datos obtenido del modelo SUM4SOFT para generar colecciones representativas. Las colecciones se aplicaron a la generación de umbrales de calidad de tres métricas de código. El manuscrito se envió a la revista Empirical Software Engineering el 12 de julio del 2024 y se encuentra en revisión.

- Carruthers, J. A., Diaz-Pace, J. A., & Irrazábal, E. SUM4SOFT: A process model to build software samples. Software and Systems Modeling.

- Estudio de caso. Este artículo presenta el modelo de proceso SUM4SOFT y las evaluaciones conducidas sobre el artefacto durante la estancia doctoral realizada en el grupo de investigación SWK de la Universidad de Hamburgo. El manuscrito se encuentra en redacción y será enviado a la revista Software and Systems Modeling.

APENDICE A

Referencias de los estudios primarios en el mapeo sistemático uno

En la Tabla 50 se presenta la lista de referencias a los estudios primarios seleccionados en el MS1. En cada artículo se muestra el ID asignado, título, autores, año de publicación y la revista o conferencia donde se publicó.

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P1	All complaints are not created equal: Text analysis of open source software defect reports	Raja, U.	2013	EMSE
P2	Integrating information retrieval, execution and link analysis algorithms to improve feature location in software	Dit, B., Revelle, M., & Poshyvanyk, D.	2013	EMSE
P3	Using tabu search to configure support vector regression for effort estimation	Corazza, A., Di Martino, S., Ferrucci, F., Gravino, C., Sarro, F., & Mendes, E.	2013	EMSE
P4	Sound empirical evidence in software testing	Fraser, G., & Arcuri, A.	2012	ICSE
P5	Integrating conceptual and logical couplings for change impact analysis in software	Kagdi, H., Gethers, M., & Poshyvanyk, D.	2013	EMSE
P6	On the impact of software evolution on software clustering	Beck, F., & Diehl, S.	2013	EMSE
P7	Studying re-opened bugs in open source software	Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E., & Matsumoto, K. I.	2013	EMSE
P8	Adoption and use of Java generics	Parnin, C., Bird, C., & Murphy-Hill, E.	2013	EMSE
P9	Automated topic naming to support cross-project analysis of software maintenance activities	Hindle, A., Ernst, N. A., Godfrey, M. W., & Mylopoulos, J.	2011	ICSE
P10	How (and why) developers use the dynamic features of programming languages: The case of smalltalk	Callaú, O., Robbes, R., Tanter, É., & Röthlisberger, D.	2013	EMSE
P11	Software defect prediction using Bayesian networks	Okutan, A., & Yildiz, O. T.	2014	EMSE
P12	Static test case prioritization using topic models	Thomas, S. W., Hemmati, H., Hassan, A. E., & Blostein, D.	2014	EMSE
P13	Configuring latent Dirichlet allocation based feature location	Biggers, L. R., Bocovich, C., Capshaw, R., Eddy, B. P., Etzkorn, L. H., & Kraft, N. A.	2014	EMSE

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P14	Predicting object-oriented class reuse-proneness using internal quality attributes	Al Dallal, J., & Morasca, S.	2014	EMSE
P15	A flexible method to estimate the software development effort based on the classification of projects and localization of comparisons	Khatibi Bardsiri, V., Jawawi, D. N. A., Hashim, S. Z. M., & Khatibi, E.	2014	EMSE
P16	Conducting quantitative software engineering studies with Alitheia Core	Gousios, G., & Spinellis, D.	2014	EMSE
P17	An experimental evaluation of the importance of randomness in hill climbing searches applied to software engineering problems	De O. Barros, M.	2014	EMSE
P18	Software process evaluation: a machine learning framework with application to defect management process	Chen, N., Hoi, S. C. H., & Xiao, X.	2014	EMSE
P19	Automating extract class refactoring: an improved method and its evaluation	Bavota, G., De Lucia, A., Marcus, A., & Oliveto, R.	2014	EMSE
P20	Bug characteristics in open source software	Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., & Zhai, C.	2014	EMSE
P21	Mining software repair models for reasoning on the search space of automated program fixing	Martinez, M., & Monperrus, M.	2013	EMSE
P22	On using machine learning to automatically classify software applications into domain categories	Linares-Vásquez, M., McMillan, C., Poshyvanyk, D., & Grechanik, M.	2014	EMSE
P23	MND-SCEMP: An empirical study of a software cost estimation modeling process in the defense domain	Lee, T., Gu, T., & Baik, J.	2014	EMSE
P24	Correlations between bugginess and time-based commit characteristics	Eyolfson, J., Tan, L., & Lam, P.	2014	EMSE
P25	1600 faults in 100 projects: automatically finding faults while achieving high coverage with EvoSuite	Fraser, G., & Arcuri, A.	2015	EMSE

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P26	A large study on the effect of code obfuscation on the quality of java code	Ceccato, M., Capiluppi, A., Falcarin, P., & Boldyreff, C.	2015	EMSE
P27	A Large-Scale Empirical Study of the Relationship between Build Technology and Build Maintenance	McIntosh, S., Nagappan, M., Adams, B., Mockus, A., & Hassan, A. E.	2015	EMSE
P28	Link analysis algorithms for static concept location: an empirical assessment	Scanniello, G., Marcus, A., & Pascale, D.	2015	EMSE
P29	Value-cognitive boosting with a support vector machine for cross-project defect prediction	Ryu, D., Choi, O., & Baik, J.	2016	EMSE
P30	Weighing lexical information for software clustering in the context of architecture recovery	Corazza, Anna, Di Martino, S., Maggio, V., & Scanniello, G.	2016	EMSE
P31	Empirical assessment of machine learning-based malware detectors for Android: Measuring the gap between in-the-lab and in-the-wild validation scenarios	Allix, K., Bissyandé, T. F., Jérome, Q., Klein, J., State, R., & Le Traon, Y.	2016	EMSE
P32	The impact of tangled code changes on defect prediction models	Herzig, K., Just, S., & Zeller, A.	2016	EMSE
P33	A contextual approach towards more accurate duplicate bug report detection and ranking	Hindle, A., Alipour, A., & Stroulia, E.	2016	EMSE
P34	Preprocessor-based variability in open-source and industrial software systems: An empirical study	Hunsen, C., Zhang, B., Siegmund, J., Kästner, C., Leßenich, O., Becker, M., & Apel, S.	2016	EMSE
P35	Understanding and addressing exhibitionism in Java empirical research about method accessibility	Vidal, S. A., Bergel, A., Marcos, C., & Díaz-Pace, J. A.	2016	EMSE
P36	Development nature matters: An empirical study of code clones in JavaScript applications	Cheung, W. T., Ryu, S., & Kim, S.	2016	EMSE
P37	Automatic identifier inconsistency detection using code dictionary	Kim, S., & Kim, D.	2016	EMSE

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P38	Studying high impact fix-inducing changes	Misirli, A. T., Shihab, E., & Kamei, Y.	2016	EMSE
P39	From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects	Scholtes, I., Mavrodiev, P., & Schweitzer, F.	2016	EMSE
P40	On the detection of custom memory allocators in C binaries	Chen, X., Slowinska, A., & Bos, H.	2016	EMSE
P41	Scalable data structure detection and classification for C/C++ binaries	Haller, I., Slowinska, A., & Bos, H.	2016	EMSE
P42	Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults	Jaafar, F., Guéhéneuc, Y. G., Hamel, S., Khomh, F., & Zulkernine, M.	2016	EMSE
P43	Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews	McIlroy, S., Ali, N., Khalid, H., & E. Hassan, A.	2016	EMSE
P44	Comparing and experimenting machine learning techniques for code smell detection	Arcelli Fontana, F., Mäntylä, M. V., Zanoni, M., & Marino, A.	2016	EMSE
P45	An empirical examination of the prevalence of inhibitors to the parallelizability of open source software systems	Alnaeli, S. M., Maletic, J. I., & Collard, M. L.	2016	EMSE
P46	Using text clustering to predict defect resolution time: a conceptual replication and an evaluation of prediction accuracy	Assar, S., Borg, M., & Pfahl, D.	2016	EMSE
P47	Studying the needed effort for identifying duplicate issues	Rakha, M. S., Shang, W., & Hassan, A. E.	2016	EMSE
P48	Change-based test selection: an empirical evaluation	Soetens, Q. D., Demeyer, S., Zaidman, A., & Pérez, J.	2016	EMSE
P49	Studying just-in-time defect prediction using cross-project models	Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., & Hassan, A. E.	2016	EMSE
P50	Towards building a universal defect prediction model with rank transformed predictors	Zhang, F., Mockus, A., Keivanloo, I., & Zou, Y.	2016	EMSE

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P51	An empirical study of the impact of modern code review practices on software quality	McIntosh, S., Kamei, Y., Adams, B., & Hassan, A. E.	2016	EMSE
P52	On the unreliability of bug severity data	Tian, Y., Ali, N., Lo, D., & Hassan, A. E.	2016	EMSE
P53	A comparative study of many-objective evolutionary algorithms for the discovery of software architectures	Ramírez, A., Romero, J. R., & Ventura, S.	2016	EMSE
P54	On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach	Mkaouer, M. W., Kessentini, M., Bechikh, S., Ó Cinnéide, M., & Deb, K.	2016	EMSE
P55	An experimental search-based approach to cohesion metric evaluation	Ó Cinnéide, M., Hemati Moghadam, I., Harman, M., Counsell, S., & Tratt, L.	2017	EMSE
P56	Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation	Chen, B., & (Jack) Jiang, Z. M.	2017	EMSE
P57	A stability assessment of solution adaptation techniques for analogy-based software effort estimation	Phannachitta, P., Keung, J., Monden, A., & Matsumoto, K.	2017	EMSE
P58	Investigating the use of moving windows to improve software effort prediction: a replicated study	Lokan, C., & Mendes, E.	2017	EMSE
P59	Review participation in modern code review: An empirical study of the android, Qt, and OpenStack projects	Thongtanunam, P., McIntosh, S., Hassan, A. E., & Iida, H.	2017	EMSE
P60	A detailed investigation of the effectiveness of whole test suite generation	Rojas, J. M., Vivanti, M., Arcuri, A., & Fraser, G.	2017	EMSE
P61	A robust multi-objective approach to balance severity and importance of refactoring opportunities	Mkaouer, M. W., Kessentini, M., Cinnéide, M., Hayashi, S., & Deb, K.	2017	EMSE
P62	Generating valid grammar-based test inputs by means of genetic programming and annotated grammars	Kifetew, F. M., Tiella, R., & Tonella, P.	2017	EMSE

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P63	A large-scale study of architectural evolution in open-source software systems	Behnamghader, P., Le, D. M., Garcia, J., Link, D., Shahbazian, A., & Medvidovic, N.	2017	EMSE
P64	Exception handling bug hazards in Android: Results from a mining study and an exploratory survey	Coelho, R., Almeida, L., Gousios, G., van Deursen, A., & Treude, C.	2017	EMSE
P65	fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage	Sawant, A. A., & Bacchelli, A.	2017	EMSE
P66	The last line effect explained	Beller, M., Zaidman, A., Karpov, A., & Zwaan, R. A.	2017	EMSE
P67	An empirical study for software change prediction using imbalanced data	Malhotra, R., & Khanna, M.	2017	EMSE
P68	Do developers update their library dependencies?: An empirical study on the impact of security advisories on library migration	Kula, R. G., German, D. M., Ouni, A., Ishio, T., & Inoue, K.	2018	EMSE
P69	Curating GitHub for engineered software projects	Munaiah, N., Kroh, S., Cabrey, C., & Nagappan, M.	2017	EMSE
P70	An empirical study of the integration time of fixed issues	da Costa, D. A., McIntosh, S., Kulesza, U., Hassan, A. E., & Abebe, S. L.	2018	EMSE
P71	Examining the stability of logging statements	Kabinna, S., Bezemer, C. P., Shang, W., Syer, M. D., & Hassan, A. E.	2018	EMSE
P72	Understanding semi-structured merge conflict characteristics in open-source Java projects	Accioly, P., Borba, P., & Cavalcanti, G.	2018	EMSE
P73	Are 20% of files responsible for 80% of defects?	Walkinshaw, N., & Minku, L.	2018	ESEM
P74	A Quantitative Study on Characteristics and Effect of Batch Refactoring on Code Smells	Bibiano, A. C., Fernandes, E., Oliveira, D., Garcia, A., Kalinowski, M., Fonseca, B., Oliveira, R., Oliveira, A., & Cedrim, D.	2019	ESEM

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P75	A scalable and efficient approach for compiling and analyzing commit history	Behnamghader, P., Meemeng, P., Fostiropoulos, I., Huang, D., Srisopha, K., & Boehm, B.	2018	ESEM
P76	On the Impact of Refactoring on the Relationship between Quality Attributes and Design Metrics	AlOmar, E. A., Mkaouer, M. W., Ouni, A., & Kessentini, M.	2019	ESEM
P77	Characteristics of method extractions in Java: a large scale empirical study	Hora, A., & Robbes, R.	2020	EMSE
P78	An investigation of misunderstanding code patterns in C open-source software projects	Medeiros, F., Lima, G., Amaral, G., Apel, S., Kästner, C., Ribeiro, M., & Gheyi, R.	2019	EMSE
P79	A Screening Test for Disclosed Vulnerabilities in FOSS Components	Dashevskyi, S., Brucker, A. D., & Massacci, F.	2019	TSE
P80	Learning from open-source projects: An empirical study on defect prediction	He, Z., Peters, F., Menzies, T., & Yang, Y.	2013	ESEM
P81	Incremental estimation of project failure risk with Naïve bayes classifier	Mori, T., Tamura, S., & Kakui, S.	2013	ESEM
P82	Simple empirical software effort estimation model	Rosa, W., Madachy, R., Boehm, B., & Clark, B.	2014	ESEM
P83	Patterns of Folder Use and Project Popularity: A Case Study of Github Repositories	Zhu, J., Zhou, M., & Mockus, A.	2014	ESEM
P84	The role of mentoring and project characteristics for onboarding in open source software projects	Fagerholm, F., Guinea, A. S., Münch, J., & Borenstein, J.	2014	ESEM
P85	Monitoring bottlenecks in achieving release readiness	Al Alam, S. M. D., Shahnewaz, S. M., Pfahl, D., & Ruhe, G.	2014	ESEM
P86	Function point structure and applicability validation using the ISBSG dataset: A replicated study	Quesada-López, C., & Jenkins, M.	2014	ESEM

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P87	Empirical Analysis of Comments and Fault-proneness in Methods?: Can Comments point to Faulty Methods??	Aman, H., Sasaki, T., Amasaki, S., & Kawahara, M.	2014	ESEM
P88	An Empirical Study of Design Degradation: How Software Projects Get Worse over Time	Ahmed, I., Mannan, U. A., Gopinath, R., & Jensen, C.	2015	ESEM
P89	An Empirical Study on C++ Concurrency Constructs	Wu, D., Chen, L., Zhou, Y., & Xu, B.	2015	ESEM
P90	Code Bad Smell Detection through Evolutionary Data Mining	Fu, S., & Shen, B.	2015	ESEM
P91	Don't Call Us, We'll Call You: Characterizing Callbacks in Javascript	Gallaba, K., Mesbah, A., & Beschastnikh, I.	2015	ESEM
P92	Empirical Analysis of Change-Proneness in Methods Having Local Variables with Long Names and Comments	Aman, H., Amasaki, S., Sasaki, T., & Kawahara, M.	2015	ESEM
P93	Identifying Metrics' Biases When Measuring or Approximating Size in Heterogeneous Languages	Hebig, R., Derehag, J., & Chaudron, M. R. V.	2015	ESEM
P94	So You Need More Method Level Datasets for Your Software Defect Prediction?: Voilà!	Shippey, T., Hall, T., Counsell, S., & Bowes, D.	2016	ESEM
P95	How Are Discussions Associated with Bug Reworking?: An Empirical Study on Open Source Projects	Zhao, Y., Zhang, F., Shihab, E., Zou, Y., & Hassan, A. E.	2016	ESEM
P96	Identifying Thresholds for Software Faultiness via Optimistic and Pessimistic Estimations	Lavazza, L., & Morasca, S.	2016	ESEM
P97	Predicting Crashing Releases of Mobile Applications	Xia, X., Shihab, E., Kamei, Y., Lo, D., & Wang, X.	2016	ESEM
P98	Social Diversity and Growth Levels of Open Source Software Projects on GitHub	Aué, J., Haisma, M., Tómasdóttir, K. F., & Bacchelli, A.	2016	ESEM
P99	Building an Ensemble for Software Defect Prediction Based on Diversity Selection	Petric, J., Bowes, D., Hall, T., Christianson, B., & Baddoo, N.	2016	ESEM

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P100	Security Vulnerabilities in Categories of Clones and Non-Cloned Code: An Empirical Study	Islam, M. R., Zibran, M. F., & Nagpal, A.	2017	ESEM
P101	Automatic Building of Java Projects in Software Repositories: A Study on Feasibility and Challenges	Hassan, F., Mostafa, S., Lam, E. S. L., & Wang, X.	2017	ESEM
P102	An Empirical Examination of the Relationship between Code Smells and Merge Conflicts	Ahmed, I., Brindescu, C., Mannan, U. A., Jensen, C., & Sarma, A.	2017	ESEM
P103	Where Is the Road for Issue Reports Classification Based on Text Mining?	Fan, Q., Yu, Y., Yin, G., Wang, T., & Wang, H.	2017	ESEM
P104	Managing Hidden Dependencies in OO Software: A Study Based on Open Source Projects	Ajienka, N., Capiluppi, A., & Counsell, S.	2017	ESEM
P105	Quantifying the Transition from Python 2 to 3: An Empirical Study of Python Applications	Malloy, B. A., & Power, J. F.	2017	ESEM
P106	File-Level Defect Prediction: Unsupervised vs. Supervised Models	Yan, M., Fang, Y., Lo, D., Xia, X., & Zhang, X.	2017	ESEM
P107	The Significant Effects of Data Sampling Approaches on Software Defect Prioritization and Classification	Bennin, K. E., Keung, J., Monden, A., Phannachitta, P., & Mensah, S.	2017	ESEM
P108	Common Bug-Fix Patterns: A Large-Scale Observational Study	Campos, E. C., & Maia, M. D. A.	2017	ESEM
P109	House of Cards: Code Smells in Open-Source C# Repositories	Sharma, T., Fragkoulis, M., & Spinellis, D.	2017	ESEM
P110	An Empirical Study of Open Source Virtual Reality Software Projects	Rodriguez, I., & Wang, X.	2017	ESEM
P111	Software structure evolution and relation to system defectiveness	Petric, J., & Galinac Grbac, T.	2014	EASE
P112	Code ownership in open-source software	Foucault, M., Falleri, J. R., & Blanc, X.	2014	EASE
P113	Preliminary comparison of techniques for dealing with imbalance in software defect prediction	Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., & Riquelme, J. C.	2014	EASE

Tabla 50. Referencias estudios primarios MS1

ID	Título	Autores	Año	Foro
P114	An empirical analysis of the utilization of multiple programming languages in open source projects	Mayer, P., & Bauer, A.	2015	EASE
P115	A cross-platform analysis of bugs and bug-fixing in open source projects: Desktop vs. Android vs. iOS	Zhou, B., Neamtiu, I., & Gupta, R.	2015	EASE
P116	Fault density analysis of object-oriented classes in presence of code clones	Elish, M. O., & Al-Ghamdi, Y.	2015	EASE
P117	Case consistency: A necessary data quality property for software engineering data sets	Phannachittay, P., Mondeny, A., Keungz, J., & Matsumotoy, K.	2015	EASE
P118	Evaluating random mutant selection at class-level in projects with non-adequate test suites	Parsai, A., Murgia, A., & Demeyer, S.	2016	EASE
P119	Slope-based fault-proneness thresholds for software engineering measures	Morasca, S., & Lavazza, L.	2016	EASE
P120	The Jinx on the NASA software defect data sets	Petric, J., Bowes, D., Hall, T., Christianson, B., & Baddoo, N.	2016	EASE
P121	An analysis of inheritance hierarchy evolution	Wood, M. I., Ivanov, L., & Lamprou, Z.	2019	EASE
P122	Comparing the effectiveness of using design and code measures in software faultiness estimation	Morasca, S., & Lavazza, L.	2019	EASE

APENDICE B

Referencias de las colecciones en el mapeo sistemático dos

En la Tabla 51 se presenta la lista de colecciones de proyectos de referencia recolectadas en el MS2. De cada colección se definió un ID y un nombre, y se extrajeron el título del estudio fundacional donde se creó la colección, sus autores y el enlace web al conjunto de datos. Existen casos donde los autores no proporcionan un enlace de acceso a los datos, y otros donde se desconoce la existencia de un estudio fundacional o autores vinculados a la producción de la colección.

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S1	Jureczko	Towards identifying software project clusters with regard to defect prediction	M. Jureczko and L. Madeyski	http://purl.org/MarianJureczko/MetricsRepo
S2	NASA	Data quality: Some comments on the NASA software defect datasets.	M. Shepperd, Q. Song, Z. Sun, and C. Mair	https://github.com/klainfo/NASADefectDataset
S3	Maxwell	Applied statistics for software managers	K. Maxwell	https://zenodo.org/record/268461/files/maxwell.arff
S4	Miyazaki	Robust regression for developing software estimation models	Miyazaki Y, Terakado M, Ozaki K, Nozaki H	https://zenodo.org/record/268465/files/miyazaki94.arff
S5	Albrecht	Software function, source lines of code, and development effort prediction: a software science validation	Albrecht AJ, Gaffney JE	https://zenodo.org/record/268467/files/albrecht.arff
S6	COCOMO81	Software engineering economics	Boehm BW	http://promise.site.uottawa.ca/SERepository/datasets/cocomo81.arff
S7	Desharnais	Analyse statistique de la productivitie des projets in formatique a partie de la technique des point des fonction	Desharnais JM	http://promise.site.uottawa.ca/SERepository/datasets/desharnais.arff
S8	Kemerer	An empirical validation of software cost estimation models	Kemerer CF	https://zenodo.org/record/268464/files/kemerer.arff

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S9	Finnish	Inter-item correlations among function points	Kitchenham B, Kansala K	-
S10	Cocomosdr	-	-	https://zenodo.org/record/268433
S11	Telecom 1	Estimating software project effort using analogies	Shepperd M, Schofield C	In the foundational study
S12	The International Software Benchmarking Standards Group (ISBSG)	Organizational benchmarking using the isbsg data repository	C. Lokan, T. Wright, P. Hill, and M. Stringer	-
S13	SF100	Sound empirical evidence in software testing	G. Fraser and A. Arcuri	https://www.evosuite.org/experimental-data/sf100/
S14	Qualitas Corpus	The Qualitas Corpus: A curated collection of Java code for empirical studies	E. Tempero C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble	http://qualitascorpus.com/download/
S15	Tukutuku	Cross-company vs. single-company web effort models using the Tukutuku database: An extended study	E. Mendes, S. Di Martino, F. Ferrucci, and C. Gravino	-
S16	CVS-Vintage	CVS-Vintage: A Dataset of 14 CVS Repositories of Java Software	M. Monperrus and M. Martinez	https://zenodo.org/record/16706

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S17	Mkaouer et al.	High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III	W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó. Cinnéide	-
S18	Hamasaki et al.	Who Does What during a Code Review? Datasets of OSS Peer Review Repositories	K. Hamasaki, R. Gaikovina Kula, N. Yoshida, A. E. Camargo Cruz, K. Fujiwara, and H. I. Naist	-
S19	Vasilescu et al.	A Data Set for Social Diversity Studies of GitHub Teams	B. Vasilescu, A. Serebrenik, and V. Filkov	https://github.com/bvasiles/diversity
S20	Yu et al.	Wait for It: Determinants of pull request evaluation latency on GitHub	Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu	-
S21	Qualitas Corpus of applications in Python	A curated benchmark collection of python systems for empirical studies on software engineering	M. Orrú, E. Tempero, M. Marchesi, R. Tonelli, and G. Destefanis	https://zenodo.org/record/268468
S22	Bug Prediction Dataset (BPD)	Evaluating defect prediction approaches: A benchmark and an extensive comparison	Marco D'Ambros, Michele Lanza & Romain Robbes	https://bug.inf.usi.ch/index.php

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S23	Sourcerer Project	Sourcerer: mining and searching internetscale software repositories	E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi	-
S24	Eclipse Bug Data (EBD)	Predicting defects for eclipse	T. Zimmermann, R. Premraj, and A. Zeller	https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/
S25	Software-artifact Infrastructure Repository (SIR)	Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact	H. Do, S. Elbaum, and G. Rothermel	https://sir.csc.ncsu.edu/portal/index.php
S26	China	-	-	https://zenodo.org/record/268446
S27	NASA93	-	-	https://zenodo.org/record/268419
S28	SOFTLAB (ar)	-	-	https://zenodo.org/record/322460
S29	Kitchenham	An empirical study of maintenance and development estimation accuracy	Barbara Kitchenham, Shari L. Pfleeger, Beth Mccoll, and Suzanne Eagan	-
S30	Atkinson	The Use of Function Points to Find Cost Analogies	K. Atkinson and M.J. Shepperd	-

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S31	Software Maintainability	Defining a Software Maintainability Dataset: Collecting, Aggregating and Analysing Expert Evaluations of Software Maintainability	Markus Schnappinger, Arnaud Fietzke, and Alexander Pretschner	https://figshare.com/articles/dataset/A_Software_Maintainability_Dataset/12801215
S32	Wang	Is there a golden feature set for static warning identification?: an experimental evaluation	Wang J, Wang S, Wang Q	https://github.com/wangjunjieISCAS/SAWarningIdentification
S33	IntroClass	The manybugs and introclass benchmarks for automated repair of c programs	Le Goues C, Holtschulte N, Smith EK, Brun Y, Devanbu P, Forrest S, Weimer W	https://github.com/BugZooOrg/IntroClass
S34	ManyBugs	The manybugs and introclass benchmarks for automated repair of c programs	Le Goues C, Holtschulte N, Smith EK, Brun Y, Devanbu P, Forrest S, Weimer W	https://github.com/squaresLab/ManyBugs
S35	CoREBench	Corebench: studying complexity of regression errors	Bohme M, Roychoudhury A	http://www.comp.nus.edu.sg/~release/corebench/
S36	Bench4BL	Bench4BL: Reproducibility Study of the Performance of IR-based Bug Localization	Jaekwon Lee, Dongsun Kim, Tegawendé F. Bissyandé, Woosung Jung, Yves Le Traon	https://github.com/exatoa/Bench4BL

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S37	NFBugs	A dataset of non-functional bugs	Radu A, Nadi S	https://github.com/ualberta-smr/NFBugs
S38	Secbench	Secbench: A database of real security vulnerabilities A database of existing vulnerabilities to enable controlled testing studies	Reis, S., Abreu, R.	https://tqrg.github.io/secbench/
S39	Ponta et al.	A manually-curated dataset of fixes to vulnerabilities of open-source software	Ponta, S.E., Plate, H., Sabetta, A., Bezzi, M., Dangremont, C.	https://github.com/SAP/project-kb/tree/master/MSR2019
S40	Blizzard-experimental data	Improving ir-based bug localization with context-aware query reformulation Improving bug localization with report quality dynamics and query reformulation	M. M. Rahman and C. K. Roy	https://github.com/masud-technope/BLIZZARD-Replication-Package-ESEC-FSE2018
S41	Experience database	Benchmarking software-development productivity	K.D. Maxwell, P. Forselius	-
S42	Eclipse and Mozilla Defect Tracking	The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information	A. Lamkanfi, J. Pérez, S. Demeyer	https://github.com/ansymo/msr2013-bug_dataset
S43	Fenton et al.	On the effectiveness of early life cycle defect prediction with Bayesian Nets	N.E. Fenton, M. Neil, et al.	-

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S44	China Software Benchmarking Standard Group (CSBSG)	An investigation of software development productivity in China	M. He, M. Li, Q. Wang, Y. Yang, K. Ye	-
S45	Ericsson	A second replicated quantitative analysis of fault distributions in complex software systems	T. Galinac Grbac, P. Runeson, D. Huljenić	-
S46	Pullreq_ci	Quality and productivity outcomes relating to continuous integration in github	B. Vasilescu, Yu Y., Wang H., P. Devanbu, V. Filkov	https://github.com/yuyue/pullreq_ci
S47	Bug Locator	Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports	J. Zhou, H. Zhang, D. Lo	https://code.google.com/archive/p/bugcenter/
S48	Relink	ReLink: recovering links between bugs and changes	R. Wu, H. Zhang, S. Kim, S.C. Cheung	https://code.google.com/archive/p/bugcenter/
S49	Fraser & Arcuri	Whole test suite generation	G. Fraser and A. Arcuri	-
S50	Second Round Classes	Unit Testing Tool Competitions – Lessons Learned	Sebastian Bauersfeld, Tanja E. J. Vos, Kiran Lakhotia	-

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S51	Palomba et al.	On the diffuseness and the impact on maintainability of code smells: a large scale empirical study	F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, A. De Lucia	https://figshare.com/s/5da162e21b8d54fbfce8
S52	Yang et al.	Mining the modern code review repositories: a dataset of people, process and product	X. Yang, R.G. Kula, N. Yoshida, H. Iida	http://kin-y.github.io/miningReviewRepo/
S53	SEMERU MSR 13	A dataset from change history to support evaluation of software maintenance tasks	B. Dit, A. Holtzhauer, D. Poshyvanyk, H. Kagdi	https://www.cs.wm.edu/semeru/data/msr13/
S54	Androtest	Automated test input generation for android: are we there yet?	S.R. Choudhary, A. Gorla, A. Orso	-
S55	JIT	A large-scale empirical study of just-in-time quality assurance	Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, N. Ubayashi	http://research.cs.queensu.ca/~kamei/jittse/jit.zip

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S56	Ohira et al.	A dataset of high impact bugs: Manually-classified issue reports	M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, K. Matsumoto	-
S57	Chavez et al.	How does refactoring affect internal quality attributes? A multi-project study	A. Chávez, I. Ferreira, E. Fernandes, D. Cedrim, A. Garcia	https://alexchavezlop.github.io/refactoring-and-internal-attributes/
S58	Wu et al.	Cve-assisted large-scale security bug report dataset construction method	Wu Xiaoxue, Zheng Wei, Chen Xiang, Wang Fang, Mu Dejun	https://github.com/wuxiaoxue/cve-assisted
S59	Ye et al.	Learning to rank relevant files for bug reports using domain knowledge Mapping bug reports to relevant files: a ranking model, a fine-grained benchmark, and feature evaluation	X. Ye, R. Bunescu, C. Liu	http://dx.doi.org/10.6084/m9.figshare.951967

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S60	Crater	Does the fault reside in a stack trace? Assisting crash localization by predicting crashing fault residence	Gu Y., Xuan J., Zhang H., Zhang L., Fan Q., Xie X., Qian T.	http://cstar.whu.edu.cn/p/crater/
S61	Mirzaei et al.	Reducing combinatorics in GUI testing of android applications	Mirzaei N., Garcia J., Bagheri H., Sadeghi A., Malek S.	-
S62	Deng et al.	Is mutation analysis effective at testing Android Apps?	Deng L., Offutt J., Samudio D.	-
S63	Aniche et al.	Code smells for model-view-controller architectures	Aniche M., Bavota G., Treude C., Gerosa M.A., van Deursen A.	https://zenodo.org/record/253681
S64	The technical debt dataset	The Technical Debt Dataset	Lenarduzzi, V., Saarimäki, N., Taibi, D.	https://github.com/clowee/The-Technical-Debt-Dataset/
S65	Levin & Yehudai	Boosting automatic commit classification into maintenance activities by utilizing source code changes	Levin S., Yehudai A.	https://doi.org/10.5281/zenodo.835534
S66	Misirli et al.	Studying high impact fix-inducing changes	Misirli A.T., Shihab E., Kamei Y.	-

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S67	Maldonado et al. 2	An empirical study on the removal of self-admitted technical debt	Maldonado da S.E., Abdalkareem R., Shihab E., Serebrenik A.	-
S68	SPL Repository	Defining metric thresholds for software product lines: A comparative study	Vale, G., Albuquerque, D., Figueiredo, E., Garcia, A.	-
S69	Thung et al.	Automatic defect categorization	F. Thung, D. Lo, and L. Jiang	-
S70	TravisTorrent	Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration	M. Beller, G. Gousios, and A. Zaidman	-
S71	Macho et al.	Predicting Build Co-Changes with Source Code Change and Commit Categories	C. Macho, S. McIntosh, and M. Pinzger	http://serg.aau.at/bin/view/ChristianMacho/DataAndTools
S72	SStuBs	How often do single-statement bugs occur? the manysstubs4j dataset	R.-M. Karampatsis and C. Sutton	https://zenodo.org/record/3653444
S73	Faultbench	On Establishing a Benchmark for Evaluating Static Analysis Alert Prioritization and Classification Techniques	S. Heckman and L. Williams	-

Tabla 51. Referencias colecciones MS2

ID	Colección	Estudio fundacional	Autores	Enlace colección
S74	Github Bug Dataset	Characterization of Source Code Defects by Data Mining Conducted on GitHub A public bug database of github projects and its application in bug prediction	Zoltán Tóth, Péter Gyimesi, and Rudolf Ferenc	http://www.inf.u-szeged.hu/~ferenc/papers/GitHubBugDataSet/

APENDICE C

Guías de referencia para diseñar el protocolo de muestreo

La Tabla 52 muestra la comparación entre las guías de referencia de muestreo reportadas por Baltes & Ralph [21] a partir de un conjunto de tareas comunes. Estas tareas se agruparon en tres actividades: planificación, ejecución y análisis.

Tabla 52. Tabla comparativa sobre guías de referencia para protocolo de muestreo

Actividades	**Tareas**	[114]	[116]	[115]	[149]	[117]	[38]
Planificación	Definir los objetivos de la investigación.	X	X	X	X	X	X
	Definir la población.	X	X	X	X	X	X
	Determinar los datos a obtener.	X					X
	Establecer la precisión deseada.	X					
	Definir los instrumentos de recolección.	X	X	X		X	X
	Seleccionar la muestra.	X	X	X	X	X	X
	Organizar el trabajo de campo.	X					
Ejecución	Realizar prueba piloto.	X	X			X	
	Registrar el proceso de selección y evaluación.					X	X
	Extraer y transformar los datos.						X
Análisis	Resumir y analizar los datos obtenidos.	X	X			X	X
	Presentar los resultados.		X				X
	Utilizar la información obtenida para futuros estudios.	X					

APENDICE D

Productos de trabajo de SUM4SOFT

En este apéndice se ilustran los modelos de productos de trabajo creados para SUM4SOFT presentados como formularios y además se incluye la especificación de estos formularios en el escenario de aplicación propuesto.

Formularios modelo

En la Fig. 37 se encuentra la guía de acceso a fuentes, donde se enumeran los datos a obtener para el estudio, las fuentes que pueden proporcionarlos mencionando las formas de acceso y restricciones, los criterios de selección, y las descripciones completas de las fuentes seleccionadas.

Guía de acceso a fuentes				
Datos a obtener:				
Id Dato	Dato	Descripción	Motivación	Id Fuente
Fuentes Candidatas:				
Id Fuente	Nombre	Descripción		URL
Formas de acceso a las fuentes y restricciones:				
Id Fuente 1:				
Id Fuente 2:				
Id Fuente n				
Criterios de selección de fuentes:				
Id criterio	Descripción			
CI				
CE				
Fuentes seleccionadas:				
Id Fuente	Descripción forma de acceso			

Fig. 37. Formulario para la guía de acceso a fuentes

El formulario de la Fig. 38 corresponde a la documentación de diseño del instrumento de recolección de datos, donde se detallan los criterios de selección de proyectos, la estrategia de muestreo y de actualización, tamaño estimado de la muestra, y especificaciones técnicas sobre el instrumento de recolección, como el lenguaje utilizado para programar la rutina, la descripción del proceso implementado para adquirir los datos, mitigar las restricciones, entre otras.

Documentación de diseño del instrumento de recolección de datos		
Criterios de selección de proyectos:		
Id umbral	Umbral	Criterio considerado
Estrategia de muestreo:		
[Descripción detallada del método de muestreo]		
Estrategia de actualización:		
[Descripción detallada del método de actualización]		
Tamaño de la muestra:		
Elementos en la muestra:		
Motivación:		
Elementos en la muestra actualizada:		
Motivación:		
Especificaciones del instrumento de recolección:		
Nombre del instrumento:		
Lenguaje de programación de implementación:		
Metodología utilizada para adquirir los datos:		
Cómo se afrontaron las restricciones:		
Herramientas utilizadas:		
Nombre	Descripción	URL

Fig. 38. Formulario para el diseño del instrumento de recolección de datos

La Fig. 39 muestra el modelo de reporte de ejecución del instrumento con datos sobre la fecha, cantidad de proyectos recolectados en la primera etapa, cantidad de proyectos obtenidos después del filtrado y la duración.

Reporte de ejecución del instrumento de recolección de datos	
Fecha de ejecución:	
Proyectos obtenidos inicialmente:	
Proyectos en el marco muestral:	
Duración de la ejecución:	

Fig. 39. Reporte de ejecución del instrumento de recolección de datos

Productos de trabajo de la aplicación del modelo

A partir de las plantillas presentadas en la sección previa, se generaron los productos de trabajo correspondientes al escenario de aplicación propuesto. En la Fig. 40 se encuentra el formulario de la guía de acceso a fuentes completado de acuerdo con las salidas obtenidas de la A1.1 en la aplicación del modelo.

Guía de acceso a fuentes			
Datos a obtener:			
Id Dato	Descripción	Motivación	Id Fuente
NAME	Nombre del repositorio	Filtrado por palabras clave	GHGQ, GLGQ
OWN	Propietario del repositorio		
URL	Url del repositorio	Acceder al repositorio	
PL	Lenguajes de programación principales	Identificar lenguaje de programación principal del proyecto	
SIZE	Tamaño del código	Cuantificar el tamaño	GHGQ, GLAR
PRIV	Accesibilidad del repositorio	Determinar la disponibilidad del repositorio	GHGQ, GLGQ
MIRR	Si el repositorio es una copia	Determinar si el proyecto es original	
CONTR	Número de colaboradores	Cuantificar la colaboración	GHAR, GLGQ
CPR	Cantidad de PR cerradas		GHGQ, GLGQ
MPR	Cantidad de PR combinadas		

Fig. 40. Ejemplo de guía de acceso a fuentes

COMM	Cantidad total de confirmaciones	Cuantificar la historia	GHGQ, GLAR
CREAT	Fecha de creación		GHGQ, GLGQ
CI	Cantidad de incidentes resueltos	Cuantificar los incidentes	
STARS	Número de estrellas	Cuantificar la popularidad	
FORKS	Cantidad de bifurcaciones		
COMMD	Fecha de la última confirmación	Detectar actividad reciente	GHGQ, GLAR
CPRD	Fecha de la última PR cerrada		GHGQ, GLGQ
MPRD	Fecha de la última PR fusionada		
ARCH	Si el repositorio esta archivado		

Fuentes Candidatas:

Id Fuente	Descripción	URL
GHGQ	API GraphQL de Github	https://api.github.com/graphql
GHAR	API REST de Github	https://api.github.com
GLGQ	API GraphQL de Gitlab	https://gitlab.com/api/graphql
GLAR	API REST de Gitlab	https://gitlab.com/api/v4

Formas de acceso a las fuentes y restricciones:

GHGQ	Github permite solicitudes de usuarios registrados al motor de búsqueda GraphQL hasta alcanzar 5000 puntos por hora (https://docs.github.com/en/graphql/overview/resource-limitations). Puntos por hora no es lo mismo que solicitudes por hora, la API GraphQL utiliza una fórmula dinámica de acuerdo al tráfico generado por los desarrolladores para limitar el servicio. Asimismo, Github solo permite recuperar información de hasta 100 resultados por página y como máximo 1000 elementos por consulta.
GHAR	Github en su capa gratuita admite hasta 5000 solicitudes por hora a su API REST por usuario registrado (https://docs.github.com/en/rest/using-the-rest-api/rate-limits-for-the-rest-api). Las credenciales de autenticación se gestionan a través de códigos de identificación o tokens.
GLGQ	Gitlab admite tráfico de hasta 2000 solicitudes por minuto a su API por usuario autenticado (https://docs.gitlab.com/ee/user/gitlab_com/#gitlabcom-specific-rate-limits).
GLAR	

Criterios de selección de fuentes:

Id criterio	Descripción
I1	Filtrado de repositorios por lenguaje de programación desde API.
I2	Cuotas de transferencia de datos suficientes para realizar las extracciones.
E1	Retorno de datos inconsistentes.

Fuentes seleccionadas:

Id Fuente	Descripción forma de acceso
GHGQ	La exploración de repositorios se realiza a partir del método "search" de la API GraphQL de Github, retornando como máximo 1000 resultados por consulta. Este método puede recibir hasta seis parámetros: "after", "before", "first", "last", "query" y "type", donde los dos primeros están reservados para la paginación, los dos siguientes para mostrar los n primeros o últimos elementos de la consulta, "query" define la cadena de búsqueda para el filtrado y "type" el tipo de elementos recuperados en la consulta (por ejemplo, "ISSUE", "REPOSITORY", "USER" o "DISCUSSION"). A continuación se presenta la consulta que retorna 15 de los metadatos requeridos, incluyendo los primeros diez repositorios que contengan código Java.

Fig. 40. Ejemplo de guía de acceso a fuentes

GHGQ

```
query{
  search(first: 10, query: "language:java", type: REPOSITORY) {
    edges {
      node {
        ... on Repository {
          id (ID)
          name (NAME)
          owner (OWN)
          url (URL)
          languages(orderBy: {direction: DESC, field: SIZE}, first: 1) {
            edges {
              node {
                name (PL)
              }
              size (SIZE)
            }
          }
          isPrivate (PRIV)
          isMirror (MIRR)
          defaultBranchRef {
            target {
              ... on Commit {
                history(first: 1) {
                  totalCount (COMM)
                  nodes {
                    committedDate (COMMD)
                  }
                }
              }
            }
          }
          createdAt (CREAT)
          issues(states:CLOSED){
            totalCount (CI)
          }
          stargazerCount (STARS)
          forkCount (FORKS)
          isArchived (ARCH)
        }
      }
    }
  }
}
```

Empleando los datos NAME y OWN obtenidos de la consulta "search", es posible extraer CPR, CPRD, MPR y MPRD de cada repositorio aplicando las siguientes solicitudes a la API GraphQL:

Fig. 40. Ejemplo de guía de acceso a fuentes

<table>
<tr><td>GHGQ</td><td>
<pre>query{
  repository(owner, name) {
    pullRequests(last: 1, states: CLOSED)
{
      totalCount (CPR)
      nodes {
        createdAt (CPRD)
      }
    }
  }
}</pre>
<pre>query{
  repository(owner, name){
    pullRequests(last: 1, states: MERGED)
{
      totalCount (MPR)
      nodes {
        createdAt (MPRD)
      }
    }
  }
}</pre>
</td></tr>
<tr><td>GHAR</td><td>Una vez identificado los propietarios y nombres de los repositorios se puede recuperar la cantidad de colaboradores de cada repositorio realizando la solicitud a la API REST a través del enlace: "https://api.github.com/repos/:owner/:name/contributors?per_page=n". Las expresiones en negrita son los parámetros del método, en el cual "owner" y "name" referencian a OWN y NAME respectivamente, y n a la cantidad de elementos por página en la solicitud.</td></tr>
</table>

Fig. 40. Ejemplo de guía de acceso a fuentes

La Fig. 41 muestra la documentación de diseño del instrumento de recolección de datos obtenida como resultado de la A1.2 en la especificación del modelo.

Documentación de diseño del instrumento de recolección de datos		
Criterios de selección de proyectos:		
Id umbral	Umbral	Criterio considerado
U1	Lenguaje de programación Java	Proyectos Java
U2	Repositorios públicos	Metadatos disponibles
U3	Repositorios GIT	Uso de herramientas para dar soporte a procesos
U4	50 incidentes resueltos o más	
U5	Repositorio que no es una copia	Proyectos originales
U6	10000 KB o más	Tamaño
U7	3 o más colaboradores	Colaboración
U8	50 PR o más	
U9	1 año o más desde su creación	Historia
U10	1000 confirmaciones o más	
U11	10 o más bifurcaciones	Popularidad
U12	10 o más estrellas	
U13	1 o más confirmaciones o PR cerradas/fusionadas en el último mes	Actividad reciente
U14	1 o más confirmaciones por cada mes en el último año	
U15	Repositorio no archivado	

Fig. 41. Ejemplo de documentación de diseño del instrumento de recolección de datos

Estrategia de muestreo:

Se empleó muestreo aleatorio estratificado para seleccionar los sujetos en el marco muestral. Esta técnica divide la población en subpoblaciones no solapadas denominados estratos y obtiene los elementos de cada estrato de manera aleatoria. La cantidad de elementos seleccionados de cada estrato son proporcionales a la cantidad de sujetos en el grupo con respecto al marco muestral.

La implementación del procedimiento de selección involucró la generación de los estratos o grupos a partir del tamaño del repositorio (SIZE) utilizando el algoritmo para agrupamiento k-means. Antes de aplicar el algoritmo, se separaron los proyectos con valores extremos de SIZE empleando la regla: "sistema registrara valores mayores a la suma entre el tercer cuartil y el 1.5 del rango intercuartil (tercer cuartil – primer cuartil)". Con la lista de elementos de la población sin valores extremos, se aplicó k-means con cinco grupos. El número de grupos fue definido después inspeccionar el conjunto de datos completo y experimentar con diferentes valores siendo cinco el número que generaba estratos de tamaños similares. Una vez obtenidos los grupos, se procedió a seleccionar aleatoriamente los proyectos de cada grupo proporcionalmente de acuerdo al tamaño del estrato y la población, por ejemplo: si la población tuviese 800 sujetos, y los grupos 170, 180, 220, 100 y 130 sujetos, en una muestra de 100 proyectos se elegirían 21, 22, 28, 13 y 16 proyectos respectivamente. A la muestra resultante, se le incorpora proporcionalmente un subconjunto de los proyectos separados anteriormente por ser identificados como valores extremos. El procedimiento puede explicarse a través del siguiente pseudocódigo:

```
stratified_sampling(gh_dataset, sample_size=112, group_variable={"SIZE"}, clusters=5):
  number_of_projects = count(gh_dataset)
  groups = get_groups_kmeans(gh_dataset, group_variable, clusters)
  new_sample = ∅
  for each group IN groups:
    group_quantity = round((count(group) / number_of_projects) * sample_size)
    group_sample = sample(group, group_quantity)
    new_sample.add(group_sample)
  return(new_sample)
```

Fig. 41. Ejemplo de documentación de diseño del instrumento de recolección de datos

Estrategia de actualización:

La condición para desencadenar la actualización de la muestra se fijó cada 365 días, es decir, una vez por año contando desde la creación de la muestra.

El proceso inicia generando un marco muestral nuevo a partir de los criterios de selección del muestreo. Si los elementos de la muestra original tienen actividad reciente se procede a actualizar los proyectos en la muestra por las nuevas versiones disponibles. Por otra parte, si existen proyectos que no tienen actualizaciones recientes estos son descartados de la muestra actualizada.

Una vez descartados los proyectos, se inicia el reemplazo de los mismos donde los proyectos a incorporar serán obtenidos a partir de un proceso similar al muestreo estratificado aleatorio definido en la estrategia de muestreo. Primero, se generan los estratos a partir marco muestral nuevo de la misma manera que la estrategia de muestreo, es decir, se separan lo valores extremos y se aplica k-means con cinco grupos para crear los estratos. Segundo, se registra la cantidad de proyectos de cada estrato que existen en la muestra actualizada (sin los repositorios descartados) y se obtiene la diferencia con la cantidad proporcional de elementos correspondientes al estrato equivalente en el marco muestral. Finalmente se seleccionan los proyectos en el marco muestral, de acuerdo a las diferencias registradas por grupo en el paso previo. Con respecto a la cantidad de proyectos en la muestra, se conserva el mismo número de la muestra original. El procedimiento está representado en el siguiente pseudocódigo:

```
sample_update(gh_dataset, sample_to_update, sample_size=112,
group_variable={"SIZE"}, clusters=5):
   number_of_projects = count(gh_dataset)
   groups = get_groups_kmeans(gh_dataset, group_variable, clusters)
   sample_to_update = sample_to_update IN gh_dataset
   for each group IN groups:
      group_quantity = round((count(group) / number_of_projects) * sample_size)
      sample_group_quantity = count(sample_to_update IN group)
      difference = group_quantity - sample_group_quantity
      if difference > 0:
         group = group NOT IN sample
         group_sample = sample(group, difference)
         sample_to_update.add(group_sample)
      else:
         sample_exclude = sample(sample_to_update, -difference)
         sample_to_update = sample_to_update NOT IN sample_exclude
   return(sample_to_update)
```

Tamaño de la muestra:

Elementos en la muestra:	112
Motivación:	Empleando el software estadístico G*Power se definió como tamaño de muestra 112 sujetos. A partir de este nivel es posible detectar efectos medianos entre grupos con un poder estadístico de 0.8 (1-β).
Elementos en la muestra actualizada:	112
Motivación:	-

Fig. 41. Ejemplo de documentación de diseño del instrumento de recolección de datos

Especificaciones del instrumento de recolección:	
Nombre del instrumento:	JavaQ
Lenguaje de programación de implementación:	Python

Metodología utilizada para adquirir los datos:

Para la recolección de datos se desarrollaron rutinas para automatizar la generación del marco muestral, el muestreo de proyectos y la actualización de la muestra. Para la generación del marco muestral, fue necesario instrumentar los umbrales de los criterios de selección en dos etapas. En la primera etapa, se recolectaron y filtraron algunos repositorios con el parámetro "query" de la función "search" de la API GraphQL de Github. A continuación se muestra una tabla con la sintaxis correspondiente para cada umbral implementado:

Id Umbral	Sintaxis "query" en GraphQL
U1	language:java
U2	is:public
U5	mirror:false
U6	size:>=10000
U9	created:<= un año atrás
U11	forks:>=10
U12	stars:>=10
U15	archived:false

En la segunda etapa de filtrado, se aplicaron los umbrales restantes siguiendo este listado de condiciones:

Id Umbral	Condición
U4	CI >= 50
U7	CONTR >= 3
U8	CPR + MPR >= 50
U10	COMM >= 1000
U13	(COMMD >= un mes atrás) OR (CPRD >= un mes atrás) OR (MPRD >= un mes atrás)
U14	COMM >= 1 por mes en el último año
Palabras Clave	("simple" OR "tutorial" OR "demo" OR "conf" OR "exam" OR "docs" OR "benchmark" OR "wiki" OR "guide" OR "template") ⊄ NAME

Estas condiciones implementan los umbrales incluyendo el filtrado por palabras clave, donde se evalúa que no existan ningunas de las palabras dentro del nombre del repositorio. En particular, para U14 se requirió inspeccionar la historia de cada repositorio empleando la siguiente consulta.

```
query{
  repository(owner, name){
    defaultBranchRef {
      target {
        ... on Commit {
          history(first: 1, since, until) {
            edges {
              node {
                committedDate }
            }
          }
        }
      }
    }
  }
}
```

Fig. 41. Ejemplo de documentación de diseño del instrumento de recolección de datos

A partir del método "repository" se exploró mensualmente fijando una fecha desde con el parámetro "since" y una fecha hasta con "until", la cantidad de confirmaciones registradas en el último año. Para la implementación del algoritmo de agrupamiento k-means se importó la biblioteca "KMeans" de scikit-learn con los argumentos: "n_clusters" = 5, "init" = "k-means++" y "n_init" = 10. Por cuestiones de rendimiento se paralelizaron algunas tareas con la biblioteca para hilos "ThreadPoolExecutor".
Cómo se afrontaron las restricciones:
Dado el límite de 1000 elementos retornables en una misma consulta en la API GraphQL y siendo que la cantidad de repositorios que satisfacen los umbrales superan los 10000 repositorios, la extracción no se puede realizar en una sola consulta. Por lo tanto, se utilizó el parámetro "size" para hacer n consultas independientes delimitando el número de resultados con los tamaños de los repositorios, por ejemplo: en la primera consulta se buscaron los repositorios entre 10000 y 11000 KB, en la segunda los repositorios entre 11000 y 12000 KB, y así sucesivamente hasta recuperar todos los resultados. Además, para cada consulta independiente (delimitada entre dos valores de "size") es necesario iterar los resultados empleando la paginación generada por la herramienta con el parámetro "after" porque el valor definido en el parámetro "first" no puede ser mayor a 100. Por cuestiones de confiabilidad, se recomienda configurar "first" por un valor menor a 100 porque el servicio puede volverse inestable y retornar errores.
Otras herramientas utilizadas:

Nombre	Descripción	URL
-	-	-

Fig. 41. Ejemplo de documentación de diseño del instrumento de recolección de datos

El reporte de la Fig. 42 presenta los primeros resultados de la tarea Ejecutar el instrumento de recolección de datos perteneciente a A2.

Reporte de ejecución del instrumento de recolección de datos	
Fecha de ejecución:	03/01/2024
Proyectos obtenidos inicialmente:	15868
Proyectos en el marco muestral:	890
Duración de la ejecución:	Generación del marco muestral: 03:15:52

Fig. 42. Ejemplo de reporte de ejecución del instrumento de recolección de datos

Referencias

[1] D. L. Parnas, “Some software engineering principles,” in *Software fundamentals: collected papers by David L. Parnas*, 2001, pp. 257–266. Accessed: Feb. 23, 2021. [Online]. Available: https://dl.acm.org/doi/10.5555/376584.376632

[2] M. M. Lehman, “Laws of software evolution revisited,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Verlag, 1996, pp. 108–124. doi: 10.1007/BFb0017737.

[3] B. Kitchenham and S. L. Pfleeger, “Software quality: the elusive target,” *IEEE Softw*, vol. 13, no. 1, pp. 12–21, Jan. 1996, doi: 10.1109/52.476281.

[4] B. Kitchenham, T. Dyba, and M. Jorgensen, “Evidence-based software engineering,” in *Proceedings. 26th International Conference on Software Engineering*, IEEE Comput. Soc, 2004, pp. 273–281. doi: 10.1109/ICSE.2004.1317449.

[5] A. Alazba and H. Aljamaan, “Code smell detection using feature selection and stacking ensemble: An empirical investigation,” *Inf Softw Technol*, vol. 138, p. 106648, Oct. 2021, doi: 10.1016/j.infsof.2021.106648.

[6] Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, and J. Too, “Boosted Whale Optimization Algorithm With Natural Selection Operators for Software Fault Prediction,” *IEEE Access*, vol. 9, pp. 14239–14258, 2021, doi: 10.1109/ACCESS.2021.3052149.

[7] C. Ni, X. Xia, D. Lo, X. Chen, and Q. Gu, “Revisiting Supervised and Unsupervised Methods for Effort-Aware Cross-Project Defect Prediction,” *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 786–802, Mar. 2022, doi: 10.1109/TSE.2020.3001739.

[8] E. Tempero *et al.*, “The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies,” in *2010 Asia Pacific Software Engineering Conference*, IEEE, Nov. 2010, pp. 336–345. doi: 10.1109/APSEC.2010.46.

[9] M. M. Rahman and C. K. Roy, “Improving IR-based bug localization with context-aware query reformulation,” in *ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*,

Association for Computing Machinery, Inc, Oct. 2018, pp. 621–632. doi: 10.1145/3236024.3236065.

[10] M. Jureczko and L. Madeyski, “Towards identifying software project clusters with regard to defect prediction,” in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE ’10*, New York, New York, USA: ACM Press, 2010, p. 1. doi: 10.1145/1868328.1868342.

[11] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: Some comments on the NASA software defect datasets,” *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013, doi: 10.1109/TSE.2013.11.

[12] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, “On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation,” *Empir Softw Eng*, vol. 23, no. 3, pp. 1188–1221, Jun. 2018, doi: 10.1007/S10664-017-9535-Z/TABLES/11.

[13] C. Wohlin and A. Aurum, “Towards a decision-making structure for selecting a research design in empirical software engineering,” *Empir Softw Eng*, vol. 20, no. 6, pp. 1427–1455, Dec. 2015, doi: 10.1007/s10664-014-9319-7.

[14] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, vol. 9783642290. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-29044-2.

[15] “Artifact Review and Badging,” ACM. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-and-badging-current

[16] E. Barr, J. Bell, M. Hilton, S. Mechtaev, and C. Timperley, “Continuously Accelerating Research,” in *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, IEEE, May 2023, pp. 123–128. doi: 10.1109/ICSE-NIER58687.2023.00028.

[17] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, “Curating GitHub for engineered software projects,” *Empir Softw Eng*, vol. 22, no. 6, pp. 3219–3253, Dec. 2017, doi: 10.1007/s10664-017-9512-6.

[18] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, “The promises and perils of mining GitHub,” in *Proceedings of the 11th Working Conference on Mining Software Repositories*, New York, NY, USA: ACM, May 2014, pp. 92–101. doi: 10.1145/2597073.2597074.

[19] D. J. Sheskin, *Parametric and Nonparametric Statistical Procedures*, 2nd ed. 2000. [Online]. Available: www.crcpress.com
[20] K. Munger, "The Limited Value of Non-Replicable Field Experiments in Contexts With Low Temporal Validity," *Soc Media Soc*, vol. 5, no. 3, p. 205630511985929, Apr. 2019, doi: 10.1177/2056305119859294.
[21] S. Baltes and P. Ralph, "Sampling in software engineering research: a critical review and guidelines," *Empir Softw Eng*, vol. 27, no. 4, p. 94, Jul. 2022, doi: 10.1007/s10664-021-10072-8.
[22] V. Cosentino, J. Luis, and J. Cabot, "Findings from GitHub," in *Proceedings of the 13th International Conference on Mining Software Repositories*, New York, NY, USA: ACM, May 2016, pp. 137–141. doi: 10.1145/2901739.2901776.
[23] A. J. Albrecht and J. E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 639–648, Nov. 1983, doi: 10.1109/TSE.1983.235271.
[24] K. Maxwell, *Applied Statistics for Software Managers*. 2002.
[25] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki, "Robust regression for developing software estimation models," *Journal of Systems and Software*, vol. 27, no. 1, pp. 3–16, Oct. 1994, doi: 10.1016/0164-1212(94)90110-4.
[26] S. Amasaki, H. Aman, and T. Yokogawa, "An extended study on applicability and performance of homogeneous cross-project defect prediction approaches under homogeneous cross-company effort estimation situation," *Empir Softw Eng*, vol. 27, no. 2, p. 46, Mar. 2022, doi: 10.1007/s10664-021-10103-4.
[27] A. Jadhav, S. K. Shandilya, I. Izonin, and M. Gregus, "Effective Software Effort Estimation Leveraging Machine Learning for Digital Transformation," *IEEE Access*, vol. 11, pp. 83523–83536, 2023, doi: 10.1109/ACCESS.2023.3293432.
[28] P. Phannachitta, "On an optimal analogy-based software effort estimation," *Inf Softw Technol*, vol. 125, p. 106330, Sep. 2020, doi: 10.1016/j.infsof.2020.106330.
[29] B. Curtis, M. I. Kellner, and J. Over, "Process modeling," *Commun ACM*, vol. 35, no. 9, pp. 75–90, Sep. 1992, doi: 10.1145/130994.130998.
[30] Ó. Marbán, G. Mariscal, E. Menasalvas, and J. Segovia, "An engineering approach to data mining projects," *Lecture Notes in Computer Science (including subseries Lecture Notes in*

Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 4881 LNCS, pp. 578–588, 2007, doi: 10.1007/978-3-540-77226-2_59/COVER.

[31] V. K. Vaishnavi and W. Kuechler, *Design Science Research Methods and Patterns*, 2nd ed. CRC Press, 2015. doi: 10.1201/b18448.

[32] B. Oates, "Philosophical Paradigms - Positivism," in *Researching Information Systems and Computing*, 2006, pp. 281–290.

[33] G. Walsham, "The Emergence of Interpretivism in IS Research," *Information Systems Research*, vol. 6, no. 4, pp. 376–394, Dec. 1995, doi: 10.1287/isre.6.4.376.

[34] P. Johannesson and E. Perjons, *An Introduction to Design Science*, vol. 9783319106. Cham: Springer International Publishing, 2014. doi: 10.1007/978-3-319-10632-8.

[35] B. Kitchenham, D. Budgen, and P. O. Brereton, "Using mapping studies as the basis for further research - A participant-observer case study," *Inf Softw Technol*, vol. 53, no. 6, pp. 638–651, Jun. 2011, doi: 10.1016/j.infsof.2010.12.011.

[36] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," in *Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, Jun. 2008, pp. 68–77. doi: 10.14236/ewic/EASE2008.8.

[37] J. D. Singer and J. B. Willett, *Applied Longitudinal Data Analysis*. Oxford University PressNew York, 2003. doi: 10.1093/acprof:oso/9780195152968.001.0001.

[38] M. Vidoni, "A systematic process for Mining Software Repositories: Results from a systematic literature review," *Inf Softw Technol*, vol. 144, p. 106791, Apr. 2022, doi: 10.1016/j.infsof.2021.106791.

[39] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir Softw Eng*, vol. 14, no. 2, pp. 131–164, Apr. 2009, doi: 10.1007/s10664-008-9102-8.

[40] B. Flyvbjerg, "Five Misunderstandings About Case-Study Research," *Qualitative Inquiry*, vol. 12, no. 2, pp. 219–245, Apr. 2006, doi: 10.1177/1077800405284363.

[41] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, London: Springer London, 2008, pp. 285–311. doi: 10.1007/978-1-84800-044-5_11.

[42] B. Kitchenham, L. Pickard, and S. L. Pfleeger, “Case Studies for Method and Tool Evaluation,” *IEEE Softw*, vol. 12, no. 4, pp. 52–62, 1995, doi: 10.1109/52.391832.

[43] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” in *Information and Software Technology*, Elsevier, Aug. 2015, pp. 1–18. doi: 10.1016/j.infsof.2015.03.007.

[44] L. Zhang, J.-H. Tian, J. Jiang, Y.-J. Liu, M.-Y. Pu, and T. Yue, “Empirical Research in Software Engineering — A Literature Survey,” *J Comput Sci Technol*, vol. 33, no. 5, pp. 876–899, Sep. 2018, doi: 10.1007/s11390-018-1864-x.

[45] J. S. Molléri, K. Petersen, and E. Mendes, “Survey Guidelines in Software Engineering,” in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA: ACM, Sep. 2016, pp. 1–6. doi: 10.1145/2961111.2962619.

[46] M. A. Storey, N. A. Ernst, C. Williams, and E. Kalliamvakou, “The who, what, how of software engineering research: a socio-technical framework,” *Empir Softw Eng*, vol. 25, no. 5, pp. 4097–4129, Sep. 2020, doi: 10.1007/s10664-020-09858-z.

[47] N. Bin Ali and K. Petersen, “Evaluating strategies for study selection in systematic literature studies,” in *International Symposium on Empirical Software Engineering and Measurement*, New York, New York, USA: IEEE Computer Society, Sep. 2014, pp. 1–4. doi: 10.1145/2652524.2652557.

[48] B. Kitchenham and P. Brereton, “A systematic review of systematic review process research in software engineering,” Dec. 01, 2013, *Elsevier B.V.* doi: 10.1016/j.infsof.2013.07.010.

[49] J. Saldaña, *The Coding Manual for Qualitative Researchers*. SAGE Publications Ltd, 2015.

[50] M. Janssen and H. van der Voort, “Agile and adaptive governance in crisis response: Lessons from the COVID-19 pandemic,” *Int J Inf Manage*, vol. 55, p. 102180, Dec. 2020, doi: 10.1016/j.ijinfomgt.2020.102180.

[51] B. F. Crabtree and W. L. Miller, *Doing Qualitative Research*, 2nd ed. SAGE Publications, Inc, 1999. [Online]. Available: https://us.sagepub.com/en-us/nam/doing-qualitative-research/book9279

[52] A. Forward and T. C. Lethbridge, “A taxonomy of software types to facilitate search and evidence-based software engineering,” in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research meeting*

of minds - CASCON '08, New York, New York, USA: ACM Press, 2008, p. 179. doi: 10.1145/1463788.1463807.

[53] S. Elmidaoui, L. Cheikhi, and A. Idri, "Towards a Taxonomy of Software Maintainability Predictors," in *Advances in Intelligent Systems and Computing*, vol. 930, Springer Verlag, 2019, pp. 823–832. doi: 10.1007/978-3-030-16181-1_77.

[54] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. 2002.

[55] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empir Softw Eng*, vol. 21, no. 3, pp. 1143–1191, Jun. 2016, doi: 10.1007/s10664-015-9378-4.

[56] J. P. T. Higgins *et al.*, *Cochrane Handbook for Systematic Reviews of Interventions*. Wiley, 2019. doi: 10.1002/9781119536604.

[57] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth, and T. Gyimóthy, "An automatically created novel bug dataset and its validation in bug prediction," *Journal of Systems and Software*, vol. 169, p. 110691, Nov. 2020, doi: 10.1016/j.jss.2020.110691.

[58] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 11, pp. 736–743, 1997, doi: 10.1109/32.637387.

[59] A. Chávez, I. Ferreira, E. Fernandes, D. Cedrim, and A. Garcia, "How does refactoring affect internal quality attributes?: A multi-project study," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Sep. 2017, pp. 74–83. doi: 10.1145/3131151.3131171.

[60] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "The technical debt dataset," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Sep. 2019, pp. 2–11. doi: 10.1145/3345629.3345630.

[61] S. Bauersfeld, T. E. J. Vos, and K. Lakhotia, "Unit Testing Tool Competitions – Lessons Learned," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer, Cham, Nov. 2013, pp. 75–94. doi: 10.1007/978-3-319-07785-7_5.

[62] X. Zeng *et al.*, "Automated test input generation for android: Are we really there yet in an industrial case?," in *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Association for Computing Machinery, Nov. 2016, pp. 987–992. doi: 10.1145/2950290.2983958.

[63] L. Deng, J. Offutt, and D. Samudio, “Is mutation analysis effective at testing android apps?,” in *Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017*, Institute of Electrical and Electronics Engineers Inc., Aug. 2017, pp. 86–93. doi: 10.1109/QRS.2017.19.

[64] G. Fraser and A. Arcuri, “Sound empirical evidence in software testing,” in *Proceedings - International Conference on Software Engineering*, IEEE, Jun. 2012, pp. 178–188. doi: 10.1109/ICSE.2012.6227195.

[65] Y. Gu *et al.*, “Does the fault reside in a stack trace? Assisting crash localization by predicting crashing fault residence,” *Journal of Systems and Software*, vol. 148, pp. 88–104, Feb. 2019, doi: 10.1016/J.JSS.2018.11.004.

[66] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in *Proceedings - ICSE 2007 Workshops: Third International Workshop on Predictor Models in Software Engineering, PROMISE’07*, IEEE, May 2007, pp. 9–9. doi: 10.1109/PROMISE.2007.10.

[67] H. Do, S. Elbaum, and G. Rothermel, “Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact,” in *Empirical Software Engineering*, Kluwer Academic Publishers PUB879 Norwell, MA, USA , Oct. 2005, pp. 405–435. doi: 10.1007/s10664-005-3861-2.

[68] T. J. Mccabe, “A Complexity Measure,” *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.

[69] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994, doi: 10.1109/32.295895.

[70] J. Whitehead, I. Mistrík, J. Grundy, and A. van der Hoek, “Collaborative Software Engineering: Concepts and Techniques,” in *Collaborative Software Engineering*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–30. doi: 10.1007/978-3-642-10294-3_1.

[71] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison, “Self-organization of teams for free/libre open source software development,” *Inf Softw Technol*, vol. 49, no. 6, pp. 564–575, Jun. 2007, doi: 10.1016/j.infsof.2007.02.004.

[72] B. Gezici, A. Tarhan, and O. Chouseinoglou, “Internal and external quality in the evolution of mobile software: An exploratory study in open-source market,” *Inf Softw Technol*,

vol. 112, pp. 178–200, Aug. 2019, doi: 10.1016/j.infsof.2019.04.002.

[73] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?," *Inf Softw Technol*, vol. 74, pp. 204–218, Jun. 2016, doi: 10.1016/j.infsof.2016.01.004.

[74] G. Gousios, M. Pinzger, and A. van Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA: ACM, May 2014, pp. 345–355. doi: 10.1145/2568225.2568260.

[75] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, 2001, doi: 10.1109/32.895984.

[76] D. J. Kim, T.-H. Chen, and J. Yang, "The secret life of test smells - an empirical study on test smell evolution and maintenance," *Empir Softw Eng*, vol. 26, no. 5, p. 100, Sep. 2021, doi: 10.1007/s10664-021-09969-1.

[77] C. Laaber, M. Basmaci, and P. Salza, "Predicting unstable software benchmarks using static source code features," *Empir Softw Eng*, vol. 26, no. 6, pp. 1–53, Aug. 2021, doi: 10.1007/S10664-021-09996-Y.

[78] C. Macho, S. Beyer, S. McIntosh, and M. Pinzger, "The nature of build changes," *Empir Softw Eng*, vol. 26, no. 3, pp. 1–53, May 2021, doi: 10.1007/S10664-020-09926-4/FIGURES/13.

[79] Z. A. Kermansaravi, M. S. Rahman, F. Khomh, F. Jaafar, and Y. G. Guéhéneuc, "Investigating design anti-pattern and design pattern mutations and their change- and fault-proneness," *Empir Softw Eng*, vol. 26, no. 1, pp. 1–47, Jan. 2021, doi: 10.1007/S10664-020-09900-0.

[80] L. P. Lima, L. S. Rocha, C. I. M. Bezerra, and M. Paixao, "Assessing exception handling testing practices in open-source libraries," *Empir Softw Eng*, vol. 26, no. 5, p. 85, Sep. 2021, doi: 10.1007/s10664-021-09983-3.

[81] G. A. A. Prana *et al.*, "Out of sight, out of mind? How vulnerable dependencies affect open-source projects," *Empir Softw Eng*, vol. 26, no. 4, p. 59, Jul. 2021, doi: 10.1007/s10664-021-09959-3.

[82] E. A. AlOmar, M. W. Mkaouer, A. Ouni, and M. Kessentini, "On the Impact of Refactoring on the Relationship between Quality Attributes and Design Metrics," in *International*

Symposium on Empirical Software Engineering and Measurement, 2019.

[83] L. Grammel, H. Schackmann, A. Schröter, C. Treude, and M.-A. Storey, "Attracting the community's many eyes," in *Human Aspects of Software Engineering*, New York, NY, USA: ACM, Oct. 2010, pp. 1–6. doi: 10.1145/1938595.1938601.

[84] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, New York, NY, USA: ACM, Nov. 2008, pp. 308–318. doi: 10.1145/1453101.1453146.

[85] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, "GitHub Projects. Quality Analysis of Open-Source Software," 2014, pp. 80–94. doi: 10.1007/978-3-319-13734-6_6.

[86] H. Borges and M. T. Valente, "What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform," *Journal of Systems and Software*, vol. 146, pp. 112–129, Dec. 2018, doi: 10.1016/j.jss.2018.09.016.

[87] S. Zamani and H. Hemmati, "A pragmatic approach for hyper-parameter tuning in search-based test case generation," *Empir Softw Eng*, vol. 26, no. 6, pp. 1–35, Sep. 2021, doi: 10.1007/S10664-021-10024-2.

[88] H. Jahanshahi, K. Chhabra, M. Cevik, and A. Baþar, "DABT: A Dependency-aware Bug Triaging Method," in *Evaluation and Assessment in Software Engineering*, New York, NY, USA: ACM, Jun. 2021, pp. 221–230. doi: 10.1145/3463274.3463342.

[89] P. Mayer and A. Bauer, "An empirical analysis of the utilization of multiple programming languages in open source projects," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, New York, NY, USA: ACM, Apr. 2015, pp. 1–10. doi: 10.1145/2745802.2745805.

[90] I. Ahmed, C. Brindescu, U. A. Mannan, C. Jensen, and A. Sarma, "An Empirical Examination of the Relationship between Code Smells and Merge Conflicts," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, Nov. 2017, pp. 58–67. doi: 10.1109/ESEM.2017.12.

[91] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980, doi: 10.1109/PROC.1980.11805.

[92] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, New York, NY, USA: ACM, Aug. 2017, pp. 186–196. doi: 10.1145/3106237.3106246.

[93] I. Scholtes, P. Mavrodiev, and F. Schweitzer, "From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects," *Empir Softw Eng*, vol. 21, no. 2, pp. 642–683, Apr. 2016, doi: 10.1007/s10664-015-9406-4.

[94] T. Xia, W. Fu, R. Shu, R. Agrawal, and T. Menzies, "Predicting health indicators for open source projects (using hyperparameter optimization)," *Empir Softw Eng*, vol. 27, no. 6, p. 122, Nov. 2022, doi: 10.1007/s10664-022-10171-0.

[95] A. Ait, J. L. C. Izquierdo, and J. Cabot, "An empirical study on the survival rate of GitHub projects," in *Proceedings of the 19th International Conference on Mining Software Repositories*, New York, NY, USA: ACM, May 2022, pp. 365–375. doi: 10.1145/3524842.3527941.

[96] J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, "Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects," *Inf Softw Technol*, vol. 122, p. 106274, Jun. 2020, doi: 10.1016/j.infsof.2020.106274.

[97] T. Lewowski and L. Madeyski, "Creating Evolving Project Data Sets in Software Engineering," in *Studies in Computational Intelligence*, vol. 851, Springer Verlag, 2020, pp. 1–14. doi: 10.1007/978-3-030-26574-8_1.

[98] O. J. Dunn, "Multiple Comparisons Among Means," *J Am Stat Assoc*, vol. 56, no. 293, p. 52, Mar. 1961, doi: 10.2307/2282330.

[99] S. S. Shapiro and M. B. Wilk, "An Analysis of Variance Test for Normality (Complete Samples)," *Biometrika*, vol. 52, no. 3/4, p. 591, Dec. 1965, doi: 10.2307/2333709.

[100] W. H. Kruskal and W. A. Wallis, "Use of Ranks in One-Criterion Variance Analysis," *J Am Stat Assoc*, vol. 47, no. 260, pp. 583–621, Dec. 1952, doi: 10.1080/01621459.1952.10483441.

[101] O. J. Dunn, "Multiple Comparisons Using Rank Sums," *Technometrics*, vol. 6, no. 3, pp. 241–252, Aug. 1964, doi: 10.1080/00401706.1964.10490181.

[102] A. Vargha, H. D. Delaney, and A. Vargha, "A Critique and Improvement of the 'CL' Common Language Effect Size Statistics of McGraw and Wong," *Journal of Educational and*

Behavioral Statistics, vol. 25, no. 2, p. 101, 2000, doi: 10.2307/1165329.

[103] M. R. Hess and J. D. Kromrey, "Robust confidence intervals for effect sizes: A comparative study of cohen's d and cliff's delta under non-normality and heterogeneous variances," in *Annual Meeting of the American Educational Research Association*, 2004.

[104] D. R. Cox and D. Oakes, *Analysis of Survival Data*. Chapman and Hall/CRC, 2018. doi: 10.1201/9781315137438.

[105] E. Iannone, R. Guadagni, F. Ferrucci, A. De Lucia, and F. Palomba, "The Secret Life of Software Vulnerabilities: A Large-Scale Empirical Study," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 44–63, Jan. 2023, doi: 10.1109/TSE.2022.3140868.

[106] R. Coelho, L. Almeida, G. Gousios, A. van Deursen, and C. Treude, "Exception handling bug hazards in Android: Results from a mining study and an exploratory survey," *Empir Softw Eng*, vol. 22, no. 3, pp. 1264–1304, Jun. 2017, doi: 10.1007/s10664-016-9443-7.

[107] E. L. Kaplan and P. Meier, "Nonparametric Estimation from Incomplete Observations," *J Am Stat Assoc*, vol. 53, no. 282, p. 457, Jun. 1958, doi: 10.2307/2281868.

[108] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development," in *Proceedings of the 38th International Conference on Software Engineering*, New York, NY, USA: ACM, May 2016, pp. 285–296. doi: 10.1145/2884781.2884826.

[109] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," *Journal of Systems and Software*, vol. 1, pp. 213–221, Jan. 1979, doi: 10.1016/0164-1212(79)90022-0.

[110] M. Caneill, D. M. Germán, and S. Zacchiroli, "The Debsources Dataset: two decades of free and open source software," *Empir Softw Eng*, vol. 22, no. 3, pp. 1405–1437, Jun. 2017, doi: 10.1007/s10664-016-9461-5.

[111] L. Hatton, D. Spinellis, and M. van Genuchten, "The long-term growth rate of evolving software: Empirical results and implications," *Journal of Software: Evolution and Process*, vol. 29, no. 5, p. e1847, May 2017, doi: 10.1002/smr.1847.

[112] G. Rousseau, R. Di Cosmo, and S. Zacchiroli, "Software provenance tracking at the scale of public source code," *Empir Softw Eng*, vol. 25, no. 4, pp. 2930–2959, Jul. 2020, doi: 10.1007/s10664-020-09828-5.

[113] V. Cosentino, J. L. Canovas Izquierdo, and J. Cabot, "A Systematic Mapping Study of Software Development With GitHub," *IEEE Access*, vol. 5, pp. 7173–7192, May 2017, doi: 10.1109/ACCESS.2017.2682323.

[114] W. Cochran, *Sampling Techniques*, 3rd ed. John Wiley & Sons, Inc., 2007.

[115] G. Henry, *Practical Sampling*. 2455 Teller Road, Thousand Oaks California 91320 United States of America: SAGE Publications, Inc., 1990. doi: 10.4135/9781412985451.

[116] R. M. de Mello, P. C. da Silva, and G. H. Travassos, "Investigating probabilistic sampling approaches for large-scale surveys in software engineering," *Journal of Software Engineering Research and Development*, vol. 3, no. 1, p. 8, Dec. 2015, doi: 10.1186/s40411-015-0023-0.

[117] B. Kitchenham and S. L. Pfleeger, "Chapter 3: Personal Opinion Surveys," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds., London: Springer London, 2008, pp. 63–92. doi: 10.1007/978-1-84800-044-5.

[118] C. Le Goues *et al.*, "The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs," *IEEE Transactions on Software Engineering*, vol. 41, no. 12, pp. 1236–1256, Dec. 2015, doi: 10.1109/TSE.2015.2454513.

[119] OMG, "Software & Systems Process Engineering Meta-Model Specification," 2008. Accessed: Dec. 16, 2022. [Online]. Available: http://www.omg.org/spec/SPEM/2.0/PDF

[120] G. Gousios and D. Spinellis, "GHTorrent: Github's data from a firehose," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, IEEE, Jun. 2012, pp. 12–21. doi: 10.1109/MSR.2012.6224294.

[121] O. Dabic, E. Aghajani, and G. Bavota, "Sampling Projects in GitHub for MSR Studies," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, IEEE, May 2021, pp. 560–564. doi: 10.1109/MSR52588.2021.00074.

[122] J. Gorostidi, A. Ait, J. Cabot, and J. L. Canovas Izquierdo, "On the Creation of Representative Samples of Software Repositories," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA: ACM, Oct. 2024, pp. 434–439. doi: 10.1145/3674805.3690747.

[123] Y. Bugayenko *et al.*, "Qualitative Clustering of Software Repositories Based on Software Metrics," *IEEE Access*, vol. 11, pp. 14716–14727, 2023, doi: 10.1109/ACCESS.2023.3244495.
[124] Z. Wu, Z. Wang, J. Chen, H. You, M. Yan, and L. Wang, "Stratified random sampling for neural network test input selection," *Inf Softw Technol*, vol. 165, p. 107331, Jan. 2024, doi: 10.1016/j.infsof.2023.107331.
[125] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means Clustering Algorithm," *Appl Stat*, vol. 28, no. 1, p. 100, 1979, doi: 10.2307/2346830.
[126] N. Fenton and J. Bieman, *Software Metrics*, Third., vol. 55, no. 2. CRC Press, 2014. doi: 10.1201/b17461.
[127] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. doi: 10.1007/3-540-39538-5.
[128] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*. Prentice-Hall, 1994.
[129] B. A. Nejmeh, "NPATH: a measure of execution path complexity and its applications," *Commun ACM*, vol. 31, no. 2, pp. 188–200, Feb. 1988, doi: 10.1145/42372.42379.
[130] T. L. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," in *2010 IEEE International Conference on Software Maintenance*, IEEE, Sep. 2010, pp. 1–10. doi: 10.1109/ICSM.2010.5609747.
[131] K. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *Journal of Systems and Software*, vol. 85, no. 2, pp. 244–257, Feb. 2012, doi: 10.1016/j.jss.2011.05.044.
[132] S. Herbold, J. Grabowski, and S. Waack, "Calculation and optimization of thresholds for sets of software metrics," *Empir Softw Eng*, vol. 16, no. 6, pp. 812–841, Dec. 2011, doi: 10.1007/s10664-011-9162-z.
[133] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, "A public unified bug dataset for java and its assessment regarding metrics and bug prediction," *Software Quality Journal*, vol. 28, no. 4, pp. 1447–1506, Dec. 2020, doi: 10.1007/s11219-020-09515-0.
[134] M. Mahdieh, S.-H. Mirian-Hosseinabadi, K. Etemadi, A. Nosrati, and S. Jalali, "Incorporating fault-proneness estimations into coverage-based test case prioritization methods," *Inf Softw Technol*, vol. 121, p. 106269, May 2020, doi: 10.1016/j.infsof.2020.106269.

[135] B. Mehboob and C. Y. Chong, "A metadata driven process for assessing stability and reusability based on risk of change of software systems," *Softw Pract Exp*, vol. 53, no. 5, pp. 1218–1248, May 2023, doi: 10.1002/spe.3187.
[136] A. Boucher and M. Badri, "Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison," *Inf Softw Technol*, vol. 96, pp. 38–67, Apr. 2018, doi: 10.1016/J.INFSOF.2017.11.005.
[137] D. Spadini, M. Schvarcbacher, A.-M. Oprescu, M. Bruntink, and A. Bacchelli, "Investigating Severity Thresholds for Test Smells," in *Proceedings of the 17th International Conference on Mining Software Repositories*, New York, NY, USA: ACM, Jun. 2020, pp. 311–321. doi: 10.1145/3379597.3387453.
[138] G. Vale, E. Fernandes, and E. Figueiredo, "On the proposal and evaluation of a benchmark-based threshold derivation method," *Software Quality Journal*, vol. 27, no. 1, pp. 275–306, Mar. 2019, doi: 10.1007/s11219-018-9405-y.
[139] V. Khatibi Bardsiri, D. N. A. Jawawi, S. Z. M. Hashim, and E. Khatibi, "A flexible method to estimate the software development effort based on the classification of projects and localization of comparisons," *Empir Softw Eng*, vol. 19, no. 4, pp. 857–884, Aug. 2014, doi: 10.1007/s10664-013-9241-4.
[140] V. Terragni, P. Salza, and M. Pezzè, "Measuring Soft-ware Testability Modulo Test Quality," vol. 11, no. 20, 2020, doi: 10.1145/3387904.3389273.
[141] A. Capiluppi, N. Ajienka, and S. Counsell, "The effect of multiple developers on structural attributes: A Study based on java software," *Journal of Systems and Software*, vol. 167, p. 110593, Sep. 2020, doi: 10.1016/j.jss.2020.110593.
[142] N. Ajienka, P. Vangorp, and A. Capiluppi, "An empirical analysis of source code metrics and smart contract resource consumption," *Journal of Software: Evolution and Process*, May 2020, doi: 10.1002/smr.2267.
[143] U. Iftikhar, N. Bin Ali, J. Börstler, and M. Usman, "A tertiary study on links between source code metrics and external quality attributes," *Inf Softw Technol*, vol. 165, p. 107348, Jan. 2024, doi: 10.1016/j.infsof.2023.107348.
[144] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering*. Wiley, 2012. doi: 10.1002/9781118181034.
[145] J. M. Gonzalez-Barahona and G. Robles, "Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories," *Inf Softw*

Technol, vol. 164, p. 107318, Dec. 2023, doi: 10.1016/j.infsof.2023.107318.

[146] J. C. Carver, N. Juristo, M. T. Baldassarre, and S. Vegas, "Replications of software engineering experiments," *Empir Softw Eng*, vol. 19, no. 2, pp. 267–276, Apr. 2014, doi: 10.1007/s10664-013-9290-8.

[147] A. Tommasel and I. Assent, "Recommendation fairness and where to find it: An empirical study on fairness of user recommender systems," in *2023 IEEE International Conference on Big Data (BigData)*, IEEE, Dec. 2023, pp. 4195–4204. doi: 10.1109/BigData59044.2023.10386616.

[148] L. Montgomery, C. Lüders, and W. Maalej, "An alternative issue tracking dataset of public Jira repositories," in *Proceedings of the 19th International Conference on Mining Software Repositories*, New York, NY, USA: ACM, May 2022, pp. 73–77. doi: 10.1145/3524842.3528486.

[149] B. Kitchenham and S. L. Pfleeger, "Principles of survey research," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 5, pp. 17–20, Sep. 2002, doi: 10.1145/571681.571686.

Esta red se constituyó formalmente en noviembre de 1996 y actualmente 51 universidades argentinas son miembros activos. Sus objetivos son:
"Coordinar actividades académicas relacionadas con el perfeccionamiento docente, la actualización curricular y la utilización de recursos compartidos en el apoyo al desarrollo de las carreras de Ciencia de la Computación y/o Informática en Argentina".
"Establecer un marco de colaboración para el desarrollo de las actividades de posgrado en Ciencia de la Computación y/o Informática de modo de optimizar la asignación y el aprovechamiento de recursos".

Las muestras de proyectos de software son vitales en la Ingeniería del Software Empírica (ISE), pero extraerlas de plataformas de código abierto presenta problemas: contienen datos irrelevantes y las colecciones compartidas a menudo carecen de rigor metodológico. Para solucionar tal reto, se desarrolló el modelo de proceso SUM4SOFT, diseñado para construir, actualizar y curar colecciones de proyectos de software. Su creación siguió la metodología de ciencia de diseño, abarcando cinco estudios empíricos: dos mapeos sistemáticos (que revelaron fallas metodológicas previas), un estudio longitudinal (que demostró la obsolescencia temporal de las muestras), minería de repositorios y un estudio de caso. Como resultado de aplicar SUM4SOFT, se construyó el instrumento JavaQ, el cual implementa estrategias de muestreo aleatorio estratificado mediante algoritmos k-means. Finalmente, se demostró que SUM4SOFT y JavaQ generan muestras representativas, replicables y generalizables para la ISE.

edulp
EDITORIAL DE LA UNLP

ISBN 978-631-6568-83-0