ISSN 2007-9737

Mapeo sistemático sobre estudios empíricos realizados con colecciones de proyectos software

Juan Andrés Carruthers[1], Jorge Andrés Diaz-Pace[2],
Emanuel Agustín Irrazábal[1]

[1] Universidad Nacional del Nordeste,
Departamento de Informática, Corrientes,
Argentina

[2] Universidad Nacional del Centro de la Provincia de Buenos Aires,
Instituto Superior De Ingeniería Del Software, Buenos Aires,
Argentina

{jacarruthers, eairrazabal}@exa.unne.edu.ar, adiaz@exa.unicen.edu.ar

Resumen. Contexto: los proyectos software son insumos comunes en los experimentos de la Ingeniería del Software, aunque estos muchas veces sean seleccionados sin seguir una estrategia específica, lo cual disminuye la representatividad y replicación de los resultados. Una opción es el uso de colecciones preservadas de proyectos software, pero estas deben ser vigentes y con reglas explícitas que aseguren su actualización a lo largo del tiempo. Objetivo: realizar un estudio secundario sistematizado sobre las estrategias de selección de los proyectos software en estudios empíricos para conocer las reglas tenidas en cuenta, el grado de uso de colecciones de proyectos, los meta-datos extraídos y los análisis estadísticos posteriores realizados. Método: se utilizó un mapeo sistemático para identificar estudios publicados desde enero de 2013 a diciembre de 2020. Resultados: se identificaron 122 estudios de los cuales el 72% utilizó reglas propias para la selección de proyectos y un 27% usó colecciones de proyectos existentes. Asimismo, no se encontraron evidencias de un marco estandarizado para la selección de proyectos, ni la aplicación de métodos estadísticos que se relacionen con la estrategia de recolección de las muestras.

Palabras clave. Colecciones, proyectos software, experimentación, ingeniería del software basada en evidencia.

A Systematic Mapping Study of Empirical Studies Performed with Collections of Software Projects

Abstract. Context: software projects are common resources in Software Engineering experiments, although these are often selected without following a specific strategy, which reduces the representativeness and replication of the results. An option is the use of preserved collections of software projects, but these must be current, with explicit guidelines that guarantee their updating over a long period of time. Goal: to carry out a systematic secondary study about the strategies to select software projects in empirical studies to discover the guidelines taken into account, the degree of use of project collections, the meta-data extracted and the subsequent statistical analysis conducted. Method: A systematic mapping study to identify studies published from January 2013 to December 2020. Results: 122 studies were identified, of which the 72% used their own guidelines for project selection and the 27% used existent project collections. Likewise, there was no evidence of a standardized framework for the project selection process, nor the application of statistical methods that relates with the sample collection strategy.

Keywords. Collections, software projects, experimentation, evidence based software engineering.

1. Introducción

El desarrollo software actual trabaja con la construcción de aplicaciones multi versión [50] que crecen, tanto en complejidad como en funcionalidad, siendo necesario conservar su calidad actual [39].

Por ello, es necesario obtener métodos empíricos para demostrar la calidad del producto software [37] y utilizar evidencia directamente relacionada con el producto software resultante a

partir de mediciones que se vinculen con los atributos de calidad del código fuente [19].

En este sentido, la Ingeniería del Software Basada en Evidencia (ISBE) proporciona los medios para que la evidencia actual de la investigación pueda integrarse con la experiencia práctica y los valores humanos en el proceso de toma de decisiones para el desarrollo y mantenimiento de software [32].

En el caso de los experimentos cuyo objeto de estudio es el código fuente, una colección *ad hoc* de proyectos software muchas veces no es suficiente para lograr representatividad o replicación en los resultados. Con la intención de mejorar los resultados surgen las colecciones preservadas de proyectos software, que se utilizan para reducir el costo de recopilar los proyectos software y para que los estudios sean replicables y comparables [60].

Estas colecciones son un insumo para los grupos de trabajo y sirven como mecanismo de comparación para los experimentos en la disciplina de Ingeniería del Software. Existen distintas colecciones, como las realizadas por Barone y Sennrich [5], Allamanis y Sutton [2] o Keivanloo [29], que se diferencian por la cantidad, calidad de proyectos que lo componen o los criterios y métodos de agrupamiento. Por ejemplo, Zerouali y Mens [66] investigaron el uso de las bibliotecas y frameworks de pruebas en Java mientras que Goeminne y Mens [21] trabajaron con frameworks de bases de datos.

En este y otros ejemplos las reglas para la selección de los proyectos muchas veces no son explícitas, se corresponden con selecciones anteriores o son al azar. La definición de un modelo de procedimientos a partir de reglas estándar y herramientas es fundamental para garantizar la capacidad de preservar sistemáticamente la colección a lo largo del tiempo por integrantes del mismo equipo de trabajo o externos, es decir, que la conservación de la misma sea independiente del grupo que la construyó.

Así, por ejemplo, la última versión del Qualitas es del año 2013 [60], lo que hace necesaria la revisión de los proyectos y sus versiones y la generación de los meta-datos necesarios. En otras colecciones, con mayor cantidad de proyectos [2], su diseño no ha tenido en cuenta la extracción de meta-datos necesarios para el estudio de la calidad del producto software.

Por lo antes indicado, el objetivo de este trabajo es determinar los criterios de selección de los proyectos software que son insumos de estudios empíricos, las características de estos proyectos, qué meta-datos se extraen, qué herramientas se utilizan para obtener los meta-datos y qué análisis estadísticos se realizan con los meta-datos recopilados. Se eligió el enfoque de mapeo sistemático para obtener una visión general del desarrollo técnico actual o el nivel de práctica de un área de investigación [51].

Este estudio pretende brindar una visión general sobre las prácticas seguidas por los grupos de investigación para la experimentación con colecciones de proyecto, exponiendo los problemas encontrados que comprometen a la representatividad de las muestras, la replicación de los experimentos y la generación de los resultados, y de esta manera evitar que siga sucediendo en futuros trabajos.

Además de esta introducción, el trabajo se encuentra organizado de la siguiente manera. La sección 2 describe los estudios relacionados con la temática. En la sección 3 se detalla la metodología empleada para el estudio y se presentan las preguntas de investigación. En la sección 4 se reportan las actividades llevadas a cabo. En la sección 5 se discuten y se responden las preguntas de investigación. Finalmente, en la sección 6 se incluye la conclusión del mapeo sistemático desarrollado.

2. Trabajos relacionados

El uso de proyectos software para realización de estudios empíricos no resulta una práctica novedosa en la Ingeniería de Software. En 1971, Knuth publicó su artículo [38] en el cual recolectó aleatoriamente programas en FORTRAN de la Universidad de Stanford y de la compañía Lockheed Missiles and Space. Chidamber y Kemerer en [7] desarrollaron un trabajo para validar las métricas orientadas a objetos creadas por ellos mismos tres años antes [8], una parte del mismo buscaba demostrar la factibilidad de estas métricas en dos sistemas comerciales: uno

codificado en el lenguaje C++ y el otro en Smalltalk.

Harrison et al. [22] también condujeron un estudio con cinco proyectos software en tecnología C++ para evaluar y comparar la métrica acoplamiento entre objetos y la métrica número de asociaciones entre una clase y sus pares.

El advenimiento de plataformas para compartir código como SourceForge, aumentó la accesibilidad a proyectos de fuente abierta [60]. Con estos nuevos repositorios públicos fue posible el acceso al código fuente versionado de proyectos software con distintos tamaños, tecnología o perfiles de trabajo de los equipos.

Esto hizo más necesaria la construcción de colecciones de proyectos software que aumente la replicabilidad de los experimentos, se agreguen los resultados, se reduzcan los costos y se mejore la representatividad de las muestras. En la literatura existe una gran variedad de ejemplos de estas colecciones. Barone y Sennrich [5] crearon una colección de más de 100K funciones en Python con sus respectivos cuerpos y descripciones para estudios.

Allamanis y Sutton [2] construyeron el Github Java Corpus con todos los proyectos Java en Github Archive que no fuesen repositorios duplicados. Similar al anterior, Keivanloo et al. [29] incluyeron 24824 proyectos Java, alojados en SourceForge y Google Code.

Zerouali y Mens [66] actualizaron el Github Java Corpus y analizaron el uso de bibliotecas y frameworks de pruebas como JUnit, Spring o TestNG. En esta actualización agregaron los repositorios creados en Github no presentes en la colección original y descartaron aquellos que no estuvieran disponibles y los que no utilizaron la herramienta Maven para gestionar las dependencias, teniendo como resultado una colección de 4532 proyectos. De igual manera, Goeminne y Mens [21] realizaron cambios al Github Java Corpus y estudiaron cinco frameworks de base de datos.

Tempero et al. [60] en el año 2013 constituyeron la colección Qualitas Corpus donde incluyeron sistemas desarrollados en Java con sus archivos fuente y binarios disponibles públicamente en formato “.jar”. El objetivo del Qualitas es reducir sustancialmente el costo de los equipos de investigación para desarrollar grandes estudios empíricos del código fuente.

En algunas colecciones, además de la documentación, código fuente y binarios del proyecto, también se proporcionan meta-datos relacionados. Por ejemplo, FLOSSmole [26] brinda los nombres de los proyectos, lenguajes de programación, plataformas, tipo de licencia, sistema operativo y datos de los desarrolladores involucrados. FLOSSMetrics [24] calcula, extrae y almacena información de sistemas de control de versiones, sistemas de rastreo de defectos, archivos de listas de correos y métricas de código. Qualitas.class [61] por su parte, contiene medidas relativas a los proyectos tales como métricas de tamaño (cantidad de líneas de código, número de paquetes, clases e interfaces) y métricas de diseño.

Finalmente, mapeos sistemáticos como los de Falessi et al. [15] y Cosentino et al. [9] estudiaron los conjuntos de datos de proyectos usados en la Ingeniería del Software, debido a la gran diversidad de criterios considerados para seleccionar los proyectos y los meta-datos obtenidos. En ambos trabajos reportaron un bajo nivel replicabilidad de las metodologías empleadas para su recolección.

3. Metodología

Se llevó a cabo un estudio de mapeo sistemático siguiendo las pautas identificadas en [52] para obtener una visión general del uso de colecciones de proyectos en la ISBE. Se seleccionó esta técnica por centrarse en la “clasificación y análisis temático de un tema de la Ingeniería del Software” [31].

En este caso, aunque la recolección de proyectos sea una práctica estándar para la experimentación en ISBE, los criterios de selección de los proyectos software, sus características, los meta-datos que se extraen de estos proyectos, qué herramientas se utilizan para obtener los meta-datos, así como también qué análisis estadísticos se realizan con los meta-datos recopilados no han sido abordados ampliamente por otros estudios secundarios.

Por lo tanto, es necesario identificar, categorizar y analizar la investigación disponible

Tabla 1. Preguntas de investigación

Pregunta de investigación	Motivación
RQ1: ¿Cuáles son los criterios de selección de los proyectos software objeto de estudios empíricos?	Identificar cuáles son los criterios tenidos en cuenta por los investigadores para la selección de proyectos en la realización de estudios empíricos.
RQ2: ¿Con qué tipo de proyectos trabajan los grupos de investigación para realizar estudios empíricos?	Conocer qué características tienen los proyectos seleccionados en términos del tipo de software y lenguaje de programación.
RQ3: ¿Cuáles son los datos/meta-datos que se extraen de estos proyectos?	Determinar los meta-datos y métricas que son extraídos de cada uno de estos proyectos para su posterior experimentación.
RQ4: ¿Qué herramientas se utilizan para obtener estos datos/meta-datos?	Determinar cuáles son las herramientas usadas en los diferentes estudios para recolectar los meta-datos.
RQ5: ¿Qué análisis estadísticos se realizan con los datos/meta-datos recopilados?	Definir los tipos de análisis estadísticos que se desarrollan sobre los experimentos hechos con los proyectos y sus respectivos meta-datos para interpretar los resultados obtenidos.

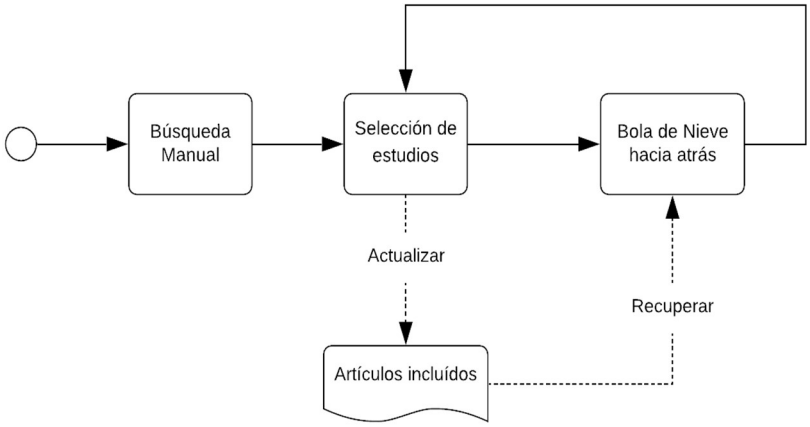

Fig. 1. Proceso de búsqueda y selección de artículos

sobre el tema para describir las prácticas y obtener una visión general de su estado de arte.

Las preguntas de investigación que guiaron el desarrollo del estudio se presentan en la Tabla 1.

3.1. 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. 1.

Búsqueda manual: se realizó una búsqueda manual en las principales conferencias y revistas enfocadas en ISBE. 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 utilizadas en otros estudios, como en [67, 47].

De acuerdo a estrategias planteadas en trabajos como [59, 67] y las buenas prácticas de búsquedas manuales de [52], en primera instancia se seleccionó el período de tiempo comprendido entre el 1 de enero del 2013 hasta el 30 de noviembre del 2018 de ESE y ESEM, de forma análoga a [67].

Posteriormente se complementó el rango de búsqueda hasta el 31 de diciembre del 2020 conforme se fueron publicando nuevos números de la revista y el congreso. Después se

incorporaron las ediciones del congreso EASE como en [47] considerando el mismo período de tiempo. De esta manera se tomaron las publicaciones de EMSE, ESEM y EASE de los últimos 8 años.

En particular, el instrumento de recolección fue una hoja de cálculo, donde se escribieron los nombres y autores de los artículos de la revista y las conferencias.

Selección de estudios: esta fase consistió en una revisión dual que se realiza de forma iterativa. Los artículos candidatos, después de ser recopilados se incluyeron o excluyeron de acuerdo con los criterios presentes en la Tabla 2.

Los trabajos fueron analizados considerando resumen, introducción, metodología, resultados y conclusión. En cada iteración, se seleccionaron 15 estudios al azar que fueron revisados por dos investigadores.

Los mismos anotaron sus decisiones de incluir o excluir cada estudio, junto con el CI o CE en que se basó la decisión.

Para medir el grado de acuerdo entre los investigadores se utilizó el estadístico Kappa de Cohen, como sugieren [2, 33]. El valor registrado de Kappa de Cohen fue 0.77, lo que demuestra un nivel de acuerdo alto.

Muestreo "bola de nieve": se complementó la búsqueda manual con el método de bola de nieve hacia atrás con un muestreo de los artículos incluidos. Las referencias proporcionaron 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 anteriormente. Este proceso se realizó en dos iteraciones.

Evaluación de la calidad: en este trabajo se decidió no realizar una evaluación de calidad de los estudios primarios, al contrario de lo que se propone en las guías de buenas prácticas de revisiones sistemáticas [25, 34].

A diferencia de las revisiones sistemáticas, en los mapeos no es necesario determinar el rigor y la relevancia de los estudios primarios porque su objetivo es proporcionar una visión general del alcance del tema investigado [52]. Finalmente, los artículos seleccionados fueron importados y procesados con la herramienta de gestión de referencias bibliográficas Mendeley[1].

Tabla 2. Criterios de inclusión y exclusión de estudios

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	Los estudios empíricos se realizan en base a la selección de un conjunto de proyectos software.
CE1	Artículos no técnicos (guías, artículos de introducción, editoriales, etc.)
CE2	Artículos duplicados.

3.2. Extracción de datos

Se utilizó un cuestionario de extracción de datos basado en las preguntas de investigación para recopilar la información relevante de los estudios primarios. La extracción fue realizada por dos investigadores y revisada por un tercero, comprobando la información presente en el formulario con cada uno de los artículos para verificar la consistencia.

Los datos recolectados incluyeron información general (título, autores, año de publicación y fuente) e información relativa a las preguntas de investigación (RQ1 – RQ5), como se ilustra en la Tabla 3.

La información se extrajo exactamente como los autores la mencionan en los artículos y los conflictos se discutieron y resolvieron internamente por los investigadores. Se adoptó este enfoque para evitar la subjetividad y facilitar la replicación del estudio.

Para dar soporte a este proceso se utilizó la herramienta FRAMEndeley [18], una extensión que permite codificar con tablas el texto resaltado en el gestor de referencias Mendeley valiéndose de los colores de subrayado para distinguir y extraer fragmentos de texto relevantes. En este caso se le asignó un color identificativo a cada RQ.

Previo a la extracción, se realizó una prueba piloto con 15 artículos para calibrar el instrumento de extracción, refinar la estrategia y evitar diferencias entre los investigadores.

[1] https://www.mendeley.com

Tabla 3. Formulario de extracción de datos

	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
RQ1	D5	Criterio	Criterio de selección de los proyectos utilizados en el estudio primario
RQ2	D6	Lenguaje	Lenguaje de programación utilizado en los proyectos seleccionados
	D7	Tipo de proyecto	Tipo de proyecto de acuerdo a su funcionalidad
RQ3	D8	Meta-datos	Meta-datos extraídos de los proyectos seleccionados en los estudios primarios
RQ4	D9	Herramientas	Herramientas de recolección de meta-datos
RQ5	D10	Análisis	Tipo de análisis estadísticos que se realiza con los meta-datos extraídos

Tabla 4. Resultados de búsqueda

Año	EMSE	ESEM	EASE
2013	37	65	31
2014	61	72	61
2015	55	41	30
2016	74	58	30
2017	91	70	50
2018	105	58	26
2019	119	54	43
2020	146	43	69
Total	688	461	340

3.3. Análisis de los datos

Las respuestas a las RQ fueron analizadas de forma cualitativa mediante la codificación abierta de los textos encontrados en los estudios primarios recolectados [54]. La selección de esta estrategia se basó en la realizada por Janssen y Van der Voort en [27].

El primer y segundo autor realizaron el proceso de forma separada e identificaron los desacuerdos, que se resolvieron sumando un tercer investigador que visitaba nuevamente la

Tabla 5. Detalle de cada fase de selección de estudios primarios

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	-

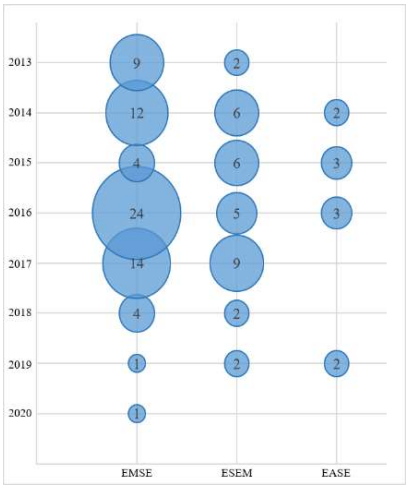

Fig. 2. Distribución de los artículos seleccionados de cada revista por año

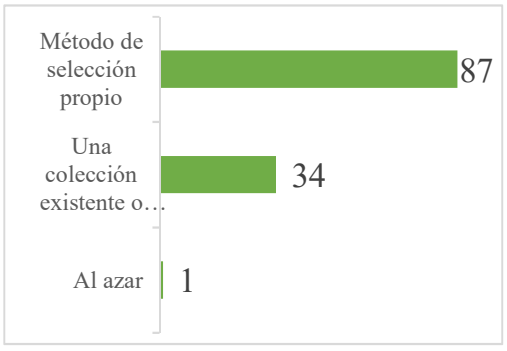

Fig. 3. Método utilizado para elegir los proyectos

fuente de información y tenía en cuenta las justificaciones dadas por los dos integrantes originales.

Como siguiente paso se utilizó la codificación cerrada [10] para identificar y resignificar las respuestas a las taxonomías encontradas. Nuevamente los dos primeros autores del estudio realizaron la codificación inicial y los desacuerdos fueron resueltos mediante la presentación a un tercer investigador.

En la sección 4 de análisis y resultados se pueden encontrar las descripciones de las clasificaciones empleadas.

4. Análisis y resultados

En esta sección, se presentan los datos recopilados y se responde a las preguntas de investigación con los datos extraídos. Los mismos se organizaron y resumieron en tablas y gráficos para una mejor visualización. Para tener un mayor nivel de detalle, en el Anexo[2] están disponibles las planillas originales con los datos extraídos.

4.1. Selección de artículos

Se obtuvo un total de 122 estudios primarios aplicando la estrategia descrita en la Sección 3.1.

La búsqueda se efectuó en la revista ESE y las conferencias EASE y ESEM empleando título, resumen y palabras claves indexadas. Los resultados se observan en la Tabla 4.

En la búsqueda manual fueron recolectados 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 fueron rechazados por no cumplir alguno de los criterios de inclusión. A los 111 artículos aceptados se le sumaron 11 en la fase 4, quedando finalmente la suma de 122. Los resultados se observan en la Tabla 5.

En la Fig. 2 se puede observar un gráfico de burbujas con la cantidad de estudios seleccionados por año en EMSE, ESEM y EASE. La revista EMSE aportó la mayor cantidad de artículos con un total de 69, seguido por ESEM con 32 y EASE con 10.

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

Desde el año 2013 al 2018 se seleccionaron en promedio casi 20 artículos por año registrándose el punto máximo en el 2016 con 32. Desde el año 2018 en adelante este promedio se reduce a 4 artículos por año siendo el punto mínimo en el 2020 con un solo artículo.

4.2. RQ1: ¿Cuáles son los criterios de selección de los proyectos software objeto de estudios empíricos?

Los estudios recolectados fueron clasificados de acuerdo a la estrategia de selección escogida teniendo en cuenta dos enfoques, uno general y otro particular. En la primera clasificación (ver Fig. 3), que abarca 3 categorías, el 72% de los artículos optaron por un método de selección propio, el 27% utilizaron una colección existente o subgrupo de esta y el 1% los recolectaron al azar.

En la Tabla 6 se encuentran las colecciones que fueron utilizadas en los 33 estudios que implementaron esta estrategia. Algunos ejemplos son: los datasets de PROMISE, el SourceForge 100 (SF100), el Qualitas Corpus y el dataset del Grupo de Estándares y Benchmarking Internacional (ISBSG), entre otros.

Para la segunda clasificación (ver Tabla 7) se definieron 13 categorías. La clasificación fue elaborada en función de los datos recopilados tal y como se indica en el punto 3.3. En total, se extrajeron datos de 94 estudios primarios.

De esta manera, el 36% de los estudios describieron criterios específicos del caso de estudio por el cual se realizaba el experimento. Por ejemplo, repositorios cuyo primer commit no pareciera una migración en [P86], proyectos que incluyan una matriz que indique las fallas que cubren los tests en [P110], sistemas de fuente abierta que sean similares a soluciones industriales en [P22], entre otros.

El 34% de los estudios construyeron la colección de proyectos seleccionando aquellos que estuvieran disponibles públicamente con sus meta-datos asociados. Se mencionaron sitios de alojamiento de proyectos software como: SourceForge en [P16, P25, P94, P107, P119]; ohloh.net en [P86]; Squeaksource en [P23]; Google Play en [P7, P73]; Github en [P1, P9], [P14, P18, P21, P29, P54, P64, P72, P102, P104, P122] o Apple Store en [P73]; entre otros.

Tabla 6. Colecciones de proyectos

#	Nombre de la colección	Artículos
17	PROMISE [1, 4, 6, 12, 28, 30, 35, 42, 44, 55, 56]	P20 P31 P37 P41 P49 P61 P65 P80 P81 P85 P88 P89 P91 P92 P97 P101 P118
2	Qualitas Corpus [60]	P12 P115
2	SF100 [P43]	P42 P99
2	ISBSG [40]	P61 P93
1	FlossMole [26]	P4
1	Vasilescu, et al [63]	P14
1	Centro de desarrollo de un banco de China	P27
1	Tukutuku [43]	P31
1	Yu et al. [65]	P40
1	D'Ambros et al [11]	P41
1	Sistemas de defensa de Corea del Sur	P66
1	Dataset de Finlandia	P68
1	Qualitas Corpus de aplicaciones en Python [49]	P70
1	CVS-Vintage [48]	P71
1	Mockus [45]	P75
1	Mkaouer et al. [45]	P78
1	Departamento de defensa de Estados Unidos	P100
1	Hamasaki et al [22]	P111

El 29% consideraron la actividad del proyecto a lo largo del tiempo como factor de selección. La actividad puede ser medida por cantidad de años de desarrollo o cantidad de commits realizados.

Así se encuentran casos como en [P94] que selecciona proyectos con 3 o más años, en [P58] recolecta proyectos con más de 10K commits, o en [P119] que excluye aquellos sistemas que tengan menos de 32 commits y un año de desarrollo.

El 28% eligieron la popularidad en repositorios públicos como criterio de recolección. Dependiendo del trabajo se tomaron enfoques diferentes para cuantificarla. Por ejemplo, la

Tabla 7. Criterios tenidos en cuenta para seleccionar los proyectos

#	Criterios	Artículos
34	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
32	Proyectos y meta-datos disponibles públicamente	P1 P6 P7 P9 P13 P14 P16 P18 P21 P23 P25 P26 P29 P30 P54 P64 P71 P72 P73 P76 P83 P86 P94 P95 P102 P104 P107 P113 P114 P119 P120 P122
27	Actividad en el proyecto a lo largo del tiempo	P3 P26 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
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
10	Disponibilidad Información de defectos	P13 P34 P78 P90 P94 P95 P107 P114 P119 P120
8	Equipo de Desarrollo	P16 P26 P51 P72 P87 P104 P113 P114
8	Calidad	P9 P15 P16 P61 P68 P105 P108 P114
7	Disponibilidad datos históricos	P26 P31 P44 P51 P94 P106 P121
5	Documentación	P14 P16 P79 P106 P114

popularidad se expresó términos de: el número de descargas [P16]; la cantidad de usuarios como [P8, P26, P106, P109, P121] el número de visitas [P29]; el número de duplicaciones del repositorio [P104]; o la cantidad de reseñas de usuarios [P73], entre otras.

En el 26% de los estudios primarios consideraron como criterio el dominio o tipo de software, es decir, la funcionalidad o ámbito de aplicación del mismo. En 21 artículos ([P1, P4, P6, P10, P12, P22, P31, P32, P33, P50, P55, P56, P57, P67, P73, P77, P84, P86, P99, P106, P121]) mencionan que los sistemas seleccionados deben provenir de múltiples dominios de aplicación.

En los tres casos restantes recolectaron proyectos de usos menos generales, como bases de datos en [P53], aplicaciones de propósito general en [P8], y ámbitos más específicos como sistemas de un banco comercial en China en [P27].

El 26% tuvieron en cuenta el tamaño de los proyectos cuantificado en términos de clases como en [P6, P16 P57, P102]; módulos en [P35]; líneas de código [P18, P55, P87, P88, P113]; puntos de función IFPUG en [P61] o puntos de función FiSMA en [P68].

En el 17% de los casos una condición fue el uso de herramientas para dar soporte a procesos dentro del desarrollo del proyecto. Esto incluyó procesos tales como: gestión de dependencias en [P2, P3, P18, P48, P64, P75, P102]; gestión de versiones en [P10, P17, P44, P58 , P69 P107, P117] o revisión de código en [P74, P111].

El 13% buscaron proyectos en los que se haya registrado actividad reciente al momento de recolección, es decir, que los mismos sigan siendo mantenidos o actualizados por sus colaboradores. La actividad suele ser medida en base a la cantidad de commits realizados en un periodo de tiempo [P14].

El 11% de los estudios establecieron como criterio de selección que hubiera disponibilidad de información de defectos en sistemas de rastreo de defectos. En el 9% de los artículos recolectaron

proyectos según la cantidad de participantes en el equipo de desarrollo o si existiera una comunidad que ofreciera soporte.

También con un 9% fue considerada la calidad del proyecto, donde se utilizaron diversos métodos para establecerla. En [P108] tomaron proyectos orientados a objetos bien diseñados con evidencias de prácticas agiles; en [P9, P105, P114] filtraron los proyectos de baja calidad con la herramienta Reaper; en [P61, P68] evaluaron la calidad según los métodos de IFPUG y FiSMA respectivamente.

El 7% buscaron aquellos proyectos en los que existiera datos históricos del proceso de desarrollo, en algunos casos para recuperar información evolutiva del sistema. Y en el 5% de los casos, la documentación del proyecto fue un factor determinante, en medios como los comentarios en el código fuente ([P14, P16, P79]), documentación de desarrollo en [P106] y licencia del software en [P114].

4.3. RQ2: ¿Con qué tipo de proyectos trabajan los grupos de investigación para realizar estudios empíricos?

Cada uno de los proyectos software en las colecciones utilizados en los estudios seleccionados fueron clasificados según el lenguaje de programación principal (Fig. 4) y el tipo de software (Fig. 5). En total se registraron 110 artículos que describieron los lenguajes de programación principales de los proyectos. En un 79% fueron construidos en Java, seguido por C con un 27% y C++ con un 24%.

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 manuales.

Cada proyecto fue clasificado según la taxonomía publicada en [16], de donde se emplearon cinco categorías de segundo nivel: diseño y construcción de software, servidor, redes y comunicaciones, sistema operativo y sistema embebido (ver Fig. 6).

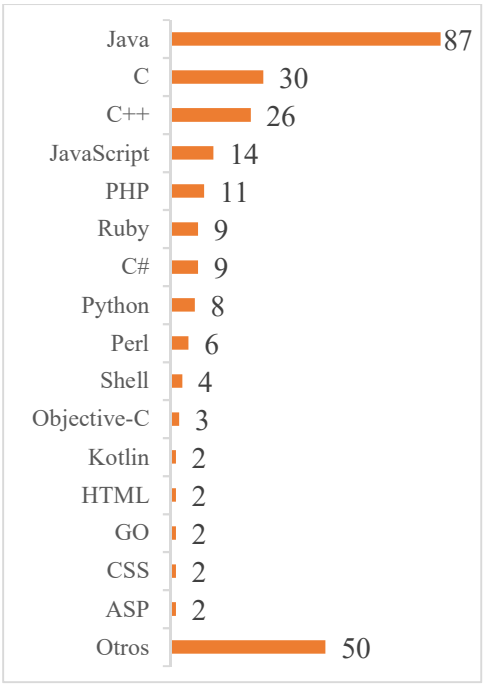

Fig. 4. Lenguajes de programación de los proyectos

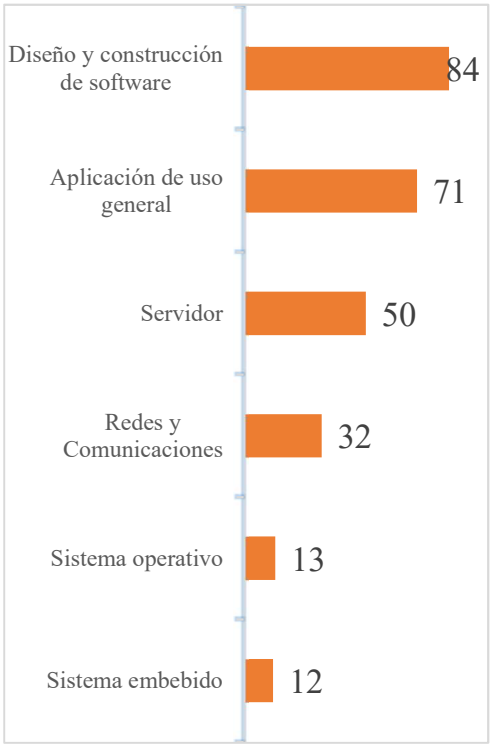

Fig. 5. Tipos de software

Para comprender las demás categorías dentro de la taxonomía original que abarcan al software guiado por datos, orientado al consumidor y de propósito general se usó el término “aplicación de uso general”.

De los estudios primarios seleccionados 96 aportaron esta información.

El 88% de los artículos trabajaron con proyectos software para el diseño y construcción de sistemas.

En este sentido se consideran compiladores como: Clang en [P47], frameworks de desarrollo como Spring Framework en [P86], entornos de desarrollo integrado como Eclipse en [P36], interfaces de programación de aplicaciones como Apache iBatis en [P43], herramientas de construcción como Ant en [P86], herramientas para el modelado del sistema como ArgoUML en [P43] o librerías para la generación de casos de prueba como jUnit en [P59], entre otros.

El 74% experimentaron con aplicaciones de uso general. En esta clasificación entran: procesadores de texto como jEdit en [P94], videojuegos como Freecol en [P12], navegadores como Firefox en [P109], aplicaciones cliente de mensajería como Pidgin en [P94], sitios web como phpMyAdmin en [P94], aplicaciones android en [P30] o reproductores de música como aTunes en [P103], entre otras.

El 52% se identificó con servidores, es decir, sistemas preparados para la recepción de solicitudes de clientes. Por ejemplo, servidores web como Apache en [P43], de base de datos como MySQL en [P53], de transferencia de archivos como FileZilla en [P94], para computación distribuida como Hadoop en [P26] o de instancias de virtualización como Apache Cloudstack en [P58].

En el 33% utilizaron software de sistema para tareas de redes y comunicaciones entre aplicaciones y dispositivos, por ejemplo: Apache Synapse en [P85], dnsjava en [P71], Quagga en [P47], ActiveMQ en [P58]. El 14% trabajaron con sistemas operativos, tales como Linux en [P109] y Android en [P75].

En el 13% estudiaron sistemas embebidos, es decir, software embebido en un dispositivo mecánico o eléctrico diseñado para realizar una función específica.

Tabla 8. Categorías de meta-datos

#	Categorías de Meta-datos	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	Code Smells	P2 P3 P7 P10 P12 P18 P19 P21 P29 P37 P44 P56 P63 P78 P79 P85 P105

Por ejemplo, en [P66] utilizaron proyectos software armamentísticos de Corea del Sur, y en [P101] sistemas controladores de una compañía de Turquía. En [P55] sistemas industriales embebidos como controladores de motores de combustión o soluciones de procesamiento de audio.

4.4. RQ3: ¿Cuáles son los datos/meta-datos que se extraen de estos proyectos?

Dependiendo del objeto de estudio del artículo se recolectaron diferentes meta-datos del código de cada uno de los proyectos.

En la Tabla 8 se agruparon los meta-datos siguiendo la taxonomía elaborada en [14] debido a la gran variedad existente.

Las categorías consideradas de esta taxonomía son aquellas que abarcan métricas obtenibles por medio del análisis estático del código.

De los 54 estudios que extrajeron meta-datos, las métricas presentes son de tamaño del código fuente en un 67%, por ejemplo, líneas de código (LOC) en [P6], volumen de Halstead en [P46] o líneas de comentarios también en [P46].

En el 63% de artículos se utilizan métricas de diseño que miden propiedades inherentes del software y se puede disgregar en otras 7 subcategorías: complejidad, acoplamiento, diagrama de clases, herencia, cohesión, encapsulamiento y polimorfismo (ver Tabla 9).

Las métricas de complejidad calculan la cantidad de caminos existentes en el flujo de control del programa; como la complejidad ciclomática de McCabe en [P25], o el peso de métodos por clase en [P85].

Las métricas de acoplamiento cuantifican la asociación o interdependencia entre los módulos, como el acoplamiento entre objetos en [P85] o la respuesta por clase en [P6].

Las medidas de diagrama de clases representan la estructura estática del sistema y pueden clasificarse en términos de clases, métodos y atributos (Tabla 10), por ejemplo: el número de métodos en [P6], el número de clases en [P12] o el número de atributos en [P6].

Las métricas de herencia miden los atributos y métodos compartidos entre clases en una relación jerárquica, dentro de las cuales están el número de hijos en [P85], la profundidad de árbol de herencia en [P25].

Las métricas de cohesión establecen el grado en que métodos y atributos de una misma clase están conectados, como la falta de cohesión en los métodos, la cohesión de clases ajustada o la cohesión de clases suelta todas en [P6].

Las medidas de encapsulamiento calculan los datos y funciones empaquetadas en una sola unidad, por ejemplo, el número de métodos accessors en [P12], la métrica de acceso a datos en [P78].

Y las medidas de polimorfismo se usan para cuantificar la habilidad de un componente de tomar múltiples formas, como el número de métodos polimórficos en [P78].

La tercera categoría es code smells. Los code smells son estructuras en el código fuente que a menudo indican la existencia de un problema de calidad o estructural, y plantean la necesidad de realizar una refactorización. Es una forma disciplinada de limpiar el código que reduce las chances de insertar defectos [17].

Si bien esta categoría no se encuentra presente en la taxonomía original [14], es razonable incluirla en la misma porque se ha establecido como un método efectivo para descubrir problemas en el código fuente y por medio de la refactorización, mejorar la calidad y mantenimiento del software [P12]. Dicho esto, el 32% de los estudios recolectan code smells.

Tabla 9. Subcategorías de meta-datos de diseño

#	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

Tabla 10. Subcategorías de meta-datos de diagrama de clases

#	Subcategorías de Diagrama de Clases	Artículos
16	Métodos	P6 P9 P12 P20 P25 P29 P49 P50 P65 P69 P78 P80 P81 P113 P118 P119
9	Atributos	P6 P12 P20 P49 P69 P78 P80 P81 P118
5	Clases	P9 P12 P78 P96 P102

4.5. RQ4: ¿Qué herramientas se utilizan para obtener estos datos/meta-datos?

Es común en estos estudios hallar diferentes herramientas para dar soporte a tareas específicas para lograr precisión y repetitividad [41]. Dada la variedad de herramientas encontradas, se realizó una clasificación en cinco categorías en base a las funciones que las mismas desempeñan.

En la Fig. 6 se puede observar un gráfico de burbujas con la clasificación antes mencionada (eje vertical), donde están descripta la cantidad de herramientas encontradas, cuántas de ellas siguen vigentes, es decir, si la última versión estable sigue en funcionamiento y cuantas han sido actualizadas después del 1º de enero del 2020 (eje horizontal). Dentro de las herramietas halladas se encuentran: Understand en [P25], CKJM en [P69] o PMD en [P85].

4.6. RQ5: ¿Qué análisis estadísticos se realizan con los datos/meta-datos recopilados?

Los investigadores necesariamente realizan un análisis previo a los datos recabados para exponer los hallazgos encontrados. Este proceso permite identificar las características que posee la muestra y a su vez seleccionar los estudios apropiados para generar los resultados. De los artículos recolectados se extrajeron los análisis estadísticos utilizados y se clasificaron en las dos áreas generales de la estadística: estadística inferencial y estadística descriptiva (ver Tabla 11), tomando como referencia las clasificaciones de [57]. En total 88 estudios informaron los procedimientos estadísticos realizados.

La estadística inferencial emplea los datos para sacar conclusiones o hacer predicciones. En el 78% de los artículos seleccionaron procedimientos inferenciales que se clasificaron en 2 categorías, test no paramétrico y test paramétrico (ver Tabla 12).

Una de las diferencias entre estos grupos es que los test paramétricos hacen suposiciones específicas con respecto a uno o más parámetros de la población que caracterizan la distribución subyacente para la cual el test está siendo empleado. Ejemplos de test paramétricos son: test T en [P23], test chi cuadrado de Wald en [P74], y test de análisis de varianza (ANOVA) en [P94].

Tabla 12. Procedimientos estadísticos inferenciales

#	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

Tabla 13. Procedimientos estadísticos descriptivos

#	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

Dentro de los no paramétricos hay ejemplos como test Mann-Whitney en [P36], test de rangos con signo de Wilcoxon en [P31], y test de normalidad Shapiro-Wilk en [P103].

Por otra parte, el 73% de los artículos trabaja con procedimientos descriptivos utilizados para presentar y resumir los datos. Estos procedimientos fueron clasificados en cinco categorías: tamaño del efecto, medida de variabilidad, medida de tendencia central, medida de asimetría y medida de curtosis (ver Tabla 13).

El tamaño de efecto mide la magnitud de fuerza de un fenómeno o efecto. En Kitchenham et al. [36] remarcan su utilidad porque proporcionan una medida objetiva de la importancia que tiene un fenómeno en un experimento, independientemente de la significación estadística de la prueba de hipótesis conducida.

Además, le afecta menos el tamaño de la muestra que a la significación estadística. Por ejemplo, podemos mencionar: el coeficiente de correlación Spearman en [P86], el tamaño de efecto de Cliff en [P103] o el coeficiente de correlación de Pearson en [P29].

Las medidas de variabilidad y las de tendencia central son los estadísticos más básicos empleados tanto en la estadística inferencial como en la descriptiva. En esta clasificación fueron incluidos solamente dentro de los procedimientos descriptivos porque en cada caso los usaron con el fin de describir la muestra recolectada. Como ejemplos de medidas de variabilidad se encuentran la desviación estándar en [P13], los cuartiles en [P6] y la varianza en [P24]. En el caso de medidas de tendencia central están la media en [P74], la mediana en [P43] y la moda en [P24].

5. Discusión

En esta sección se discuten los resultados obtenidos y como se relacionan con las preguntas de investigación identificadas en la sección 2. Sin embargo, es posible que no se hayan localizado todos los estudios relevantes, por esa razón el proceso fue desarrollado metodológicamente siguiendo un protocolo bien definido y múltiples investigadores revisaron la calidad de la información extraída. El mapeo comprendió 122 artículos publicados en la revista EMSE y las conferencias ESEM y EASE durante el período 2013 – 2020.

5.1. RQ1: ¿Cuáles son los criterios de selección de los proyectos software objeto de estudios empíricos?

Para abordar esta pregunta se buscaron evidencias de la selección de proyectos software con dos enfoques, uno a nivel general y otro más específico. A nivel general, en la mayoría de los casos (72%) los investigadores siguen lineamientos propios para seleccionar sus proyectos y en menor medida (27%) utilizan una colección de proyectos existentes.

De los 94 estudios que reportaron información al respecto, los criterios de selección predominantes son: los específicos del caso de estudio (36%), junto con la disponibilidad de los proyectos y meta-datos (34%), el período de actividad en el proyecto (29%), la popularidad (28%), el dominio de aplicación (26%) el cual muchas veces no se encuentra descrito con precisión, el tamaño del proyecto (26%) que tiene en cuenta diferentes dimensiones, desde la cantidad de líneas de código hasta medidas de punto función.

La Tabla 7 evidencia la diversidad de estrategias existentes para recolectar las muestras. En particular, algunos estudios optan por automatizar el proceso utilizando herramientas para explorar repositorios públicos como Github o SourceForge.

Así, en [P9, P83, P105] aplican un framework, que permite seleccionar repositorios Github que contengan “proyectos que aprovechan las prácticas sólidas de ingeniería de software en una o más de sus dimensiones, como la documentación, pruebas y gestión de proyectos” [P83].

O en [P24, P107] que utilizan un lenguaje de programación de dominio específico para el análisis y minado de repositorios software de gran escala [13].

En resumen, se evidencias estrategias diversas, pocos descritas o justificadas y en ocasiones sesgadas para los estudios.

5.2. Q2: ¿Con qué tipo de proyectos trabajan los grupos de investigación para realizar estudios empíricos?

Para caracterizar la muestra de los proyectos software recolectados por los grupos de investigación se describieron el lenguaje de programación principal y el tipo de actividad o función que desempeña. De 110 artículos que reportan el lenguaje de programación principal de los proyectos, la mayoría (79%) experimentan con Java y en menor grado (31%) los lenguajes C o C++. Esto puede tener una relación directa con la cantidad de proyectos en los repositorios de ambos lenguajes [20].

Otra razón es la gran cantidad de herramientas de análisis estático del código fuente compatibles disponibles para el caso del lenguaje Java. Aún así, trabajos como [P31, P60, P66, P72, P75, P83, P104, P114, P119, P121] recolectaron proyectos con 5 o más lenguajes.

Finalmente, de los 96 artículos que fue posible determinar el tipo de software empleado la mayoría era para el diseño y construcción de sistemas (88%) y, en segundo lugar, aplicaciones de uso general (74%).

5.3. RQ3: ¿Cuáles son los datos/meta-datos que se extraen de estos proyectos?

Para responder esta pregunta se buscaron los meta-datos extraídos de los proyectos software para la realización de los experimentos. Así se encontraron 54 estudios primarios que reúnen meta-datos del código en forma de métricas, indicadores o medidas.

De esta manera en este conjunto de artículos se pueden encontrar métricas que cuantifican el tamaño (67%), diseño (63%) y los code smells (32%). Esto evidencia el uso de meta-datos básicos en primera instancia y la posibilidad de incluir interpretaciones en una segunda instancia.

El resto de los trabajos se valieron de otras fuentes de información. Por ejemplo, en 13 artículos utilizaron los reportes de defectos para calcular su tiempo de resolución en [P13, P26, P94]; clasificarlos en [P52, P106, P109, P112] o detectar reportes duplicados en [P95]; entre otros estudios.

En 6 artículos recolectaron la información de la ejecución del programa (análisis dinámico) para localizar funciones específicas en [P36], clasificar los procesos en [P27], estudiar la asignación de memoria en [P28], analizar cómo trabajan las excepciones en [P30] o las dependencias del sistema en ejecución en [P4].

En 50 artículos extrajeron los datos contenidos en el repositorio del proyecto para medir el acoplamiento y dependencias en la evolución del sistema en [P16, P51, P59]; clasificar los commits en [P38, P53]; predecir defectos en el software en [P101]; estudiar el uso de patrones en la historia del proyecto en [P57] o el aporte de los desarrolladores [P74], entre otros.

Para estos casos es necesario que los proyectos tengan un conjunto de meta-datos registrados a lo largo de su tiempo de desarrollo. Esto se consigue muchas veces en la gestión del proyecto con una herramienta de control de versiones y desarrollo colaborativo.

5.4. RQ4: ¿Qué herramientas se utilizan para obtener estos datos?

El objetivo de esta pregunta es conocer las herramientas utilizadas en la Ingeniería del Software para la recolección de meta-datos de proyectos. Es notable que solamente 61 artículos mencionan de manera explícita los nombres de las herramientas utilizadas. De los cuales 44 estudios informan herramientas que estrictamente generen meta-datos de los proyectos, siendo este un requisito indispensable para la replicabilidad de los experimentos.

En muchas ocasiones se informa el modelo, técnica, procedimiento o algoritmo utilizado, pero no así la herramienta utilizada. Por ejemplo, en [P36] utilizan un modelo de fusión de datos compuesto por técnicas de recuperación de información, análisis dinámicos y minado Web para la localización de métodos que desarrollan funciones específicas del sistema.

En [P31] implementan un algoritmo de búsqueda meta-heurístico en un modelo de aprendizaje automatizado para estimar el esfuerzo de desarrollo. En [P27] declararon el uso de un “método propio” para la refactorización de artefactos software.

En [P6] trabajaron con una herramienta con la que calculan 29 métricas de código fuente pero no reportan su nombre, ni como acceder a la misma.

Dicho esto, en los 44 estudios (ver Tabla 15) usaron herramientas para el cálculo de métricas de software (57%), análisis de la estructura y dependencias del código fuente (37%), detección de code smells o vulnerabilidades (37%), refactorización de código (9%), generación automática de tests (9%) y detección patrones de diseño (5%). En la Tabla 14 y en la Tabla 15 se encuentran las cuatro herramientas más repetidas en los artículos seleccionados.

Existen herramientas que se ubican en más de una categoría de la clasificación presentada en la Fig. 6 y Tabla 14. Understand, Alitheia Core y Analizo calculan métricas de software y analizan la estructura y dependencias del código fuente. PMD, iPlasma, SonarQube, Designite e InCode también calculan métricas y además detectan code smells y vulnerabilidades del código.

5.5. RQ5: ¿Qué análisis estadísticos se realizan con los datos recopilados?

Esta pregunta pretende determinar qué técnicas o procedimientos estadísticos son elegidos por los investigadores para validar los resultados de los experimentos realizados.

En este sentido, la selección y aplicación apropiada de los métodos de análisis es una de las recomendaciones de Wohlin y Rainer en [63] para garantizar que la evidencia generada se presente correctamente y evitar que existan interpretaciones erróneas de los resultados.

De los 88 artículos que se registraron procedimientos estadísticos el 78% son inferenciales, de los cuales el 25% contiene pruebas o tests estadísticos paramétricos para el análisis de los datos recopilados.

Esto implica supuestos, como que los datos obtenidos sigan una distribución normal, lo que podría no coincidir con la forma de selección de los proyectos o los mecanismos de extracción de los datos. Así, por ejemplo, muchos estudios utilizan reglas basadas en la disponibilidad del proyecto o su popularidad, pero no en la representatividad de los meta-datos con respecto a la población.

Tabla 14. Tipo de herramientas

#	Tipo de Herramientas	Artículos
25	Cálculo 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 code 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 tests	P42 P43 P62 P99
2	Detección patrones de diseño	P57 P78

Tabla 15. Herramientas más utilizadas

#	Herramientas	Artículos
8	Understand[3]	P2 P4 P9 P21 P25 P74 P116 P119
4	Evo Suite[4]	P42 P43 P62 P99
3	CKJM[5]	P20 P25 P69
3	PMD[6]	P12 P67 P85

[3] https://www.scitools.com
[4] https://www.evosuite.org
[5] https://www.spinellis.gr/sw/ckjm
[6] https://pmd.github.io

Finalmente, en 51 estudios incorporaron un análisis de tamaño del efecto, siendo esta una recomendación indicada para los casos en que las muestras sean poco representativas y se utilicen técnicas paramétricas [36].

5.6. Amenazas a la validez

A continuación, se discuten las amenazas a la validez del estudio siguiendo el enfoque propuesto por [53]. Se consideraron cinco aspectos de validez.

Validez de constructo. La validez del constructo refleja hasta qué punto la metodología de la investigación representa a la estrategia del investigador y lo que se busca estudiar en las preguntas de investigación. Esta amenaza está presente al diseñar el instrumento de extracción de datos. La misma disminuyó implementando una prueba piloto de la tabla, tomando artículos al azar para completar una primera versión, y modificándola iterativamente según sea necesario hasta lograr la versión final.

Validez interna. Este aspecto de validez analiza los riesgos cuando se estudian relaciones causales [58]. Elegir los artículos y el período de tiempo adecuados son factores importantes que afectan la validez interna. La cantidad de artículos relacionados con la ISBE ha crecido mucho recientemente y las revistas de Ingeniería del Software tienen diferentes grados de aceptación para investigaciones empíricas. Para mitigar esta amenaza, se seleccionaron artículos de revistas y conferencias ampliamente aceptados en el ámbito de la Ingeniería del Software en términos de alcance y reputación. La subjetividad en la selección disminuyó mediante el proceso descrito en la Sección 3.1, realizando tantas iteraciones como fueron necesarias hasta obtener un grado de acuerdo alto.

Validez externa. La validez externa se refiere hasta qué punto es posible generalizar los resultados más allá del estudio. Los hallazgos aquí presentados se basan en las publicaciones pertenecientes a EMSE, EASE y ESEM. Si bien se desconoce si los resultados se pueden generalizar a artículos de otras revistas o conferencias, la investigación se basó en el análisis de 122 artículos y, por tanto, puede considerarse representativa.

Fiabilidad. La fiabilidad evidencia hasta qué punto los resultados de la investigación son independientes de los investigadores. Es decir, si otro autor llevase a cabo el mismo estudio, los resultados deberían ser iguales o similares [53]. En este artículo, los métodos y procesos de investigación se describen en detalle para garantizar su reproducibilidad y en el anexo se incluyen las planillas originales con los datos extraídos.

Sesgo de publicación. El sesgo de publicación se refiere al "problema de que es más probable que se publiquen los resultados positivos de la investigación que los negativos" [62]. Este problema ocurre en cualquier revisión de literatura o estudio de mapeo. Sin embargo, en este caso, su efecto fue moderado porque nuestro estudio no pretende comparar resultados de investigación.

6. Conclusiones

En este mapeo sistemático se identificaron artículos en los que se realizaron estudios empíricos con colecciones de proyectos. Se abordaron cinco preguntas de investigación de estos estudios, como los criterios de selección de proyectos, su caracterización, los meta-datos recolectados en los estudios empíricos, las herramientas utilizadas para generar u obtener los meta-datos y los análisis estadísticos desarrollados con ellos.

Por medio de una búsqueda manual inicial en la revista EMSE y las conferencias ESEM y EASE se obtuvieron 1496 artículos entre el 1 de enero del 2013 al 31 de diciembre del 2020. De los cuales 122 estudios fueron seleccionados después de aplicar los criterios de inclusión y exclusión definidos. A continuación, se presentan las respuestas a las preguntas de investigación.

Respecto de los criterios de selección del conjunto de proyectos, las practicas más comunes realizadas por los investigadores en este sentido es seguir lineamientos propios para seleccionar los proyectos y utilizar una colección de proyectos existentes. No se ha evidenciado un marco unificado o automatizado para la selección de proyectos debido a la gran diversidad de aspectos considerados en los 94 estudios que reportaron criterios con un mayor nivel de detalle.

En 35 estudios se informaron criterios específicos del caso de estudio y solo en 5 casos se reportaron el uso de herramientas para automatizar el proceso de selección en base a las reglas establecidas.

Con respecto a las características de los proyectos recolectados, el lenguaje de programación principal de los proyectos software con un gran margen de diferencia es Java (79%), el segundo y tercero son C y C++ (31%) de 110 artículos que se registraron respuestas.

Los proyectos software utilizados fueron específicos con un 74% de aplicaciones de uso general y 96% de proyectos en el dominio del diseño y construcción de sistemas, servidores, redes, sistemas operativos o sistemas embebidos de los 96 artículos que fue posible determinar el tipo de software empleado.

Con respecto a los meta-datos de los proyectos, las fuentes de información de donde se extraen varían desde el mismo código fuente, la información en tiempo de ejecución, los reportes de defectos o el repositorio del proyecto, entre otras.

En particular se encontraron 54 estudios primarios que reúnen meta-datos extraídos del código de los proyectos software en forma de métricas. Estas miden aspectos como el tamaño, diseño y code smells del proyecto.

Respecto de las herramientas para obtener los meta-datos, en 44 estudios primarios se utilizan herramientas para la recolección de meta-datos de proyectos obtenidos por medio del análisis estático del código.

Realizan tareas como el cálculo de métricas de software, el análisis de la estructura y dependencias del código fuente, la detección de code smells o vulnerabilidades, la refactorización de código, la generación automática de tests y la detección patrones de diseño. Las herramientas que más se repitieron fueron Understand (8), Evo Suite (4), CKJM (3) y PMD (3).

De los análisis estadísticos desarrollados sobre los meta-datos, en 88 artículos se registraron tanto procedimientos estadísticos inferenciales como descriptivos. La mayoría de los métodos inferenciales fueron pruebas estadísticas no paramétricas (69%) y en menor medida paramétricas (25%). En el caso de los métodos descriptivos, se utilizaron tamaño de efecto, medidas de variabilidad y medidas de tendencia central.

Dicho esto, en un gran número de casos no se observa que se tome en consideración la forma de selección de las muestras en los análisis estadísticos practicados, ignorando la posibilidad de que las mismas no sean representativas de la población.

Finalmente, como aporte de este trabajo se identificaron algunas pautas que ayudan a sistematizar la selección de proyectos en la construcción de colecciones con fines de investigación. Las principales reglas son: código fuente libre tanto su acceso como distribución, la vigencia del proyecto, su popularidad en repositorios y en lenguaje Java.

La colección resultante debería conservar el código fuente de los proyectos, sus métricas obtenidas del análisis estático y los valores de estadísticos descriptivos que caractericen la muestra.

Como trabajo futuro se analizarán las colecciones creadas con fines de investigación presentes en la Tabla 6 y en artículos relacionados, con el objetivo de reconocer cuales fueron los criterios considerados en cada caso, el propósito de su construcción, y la vigencia de estos.

Agradecimientos

Este artículo cuenta con el apoyo de la Universidad Nacional del Nordeste (UNNE) a través del proyecto PI-17F018 “Metodologías y herramientas emergentes para contribuir con la calidad del software”, acreditado por la Secretaría de Ciencia y Técnica de la misma entidad para el periodo 2018-2021, y del Consejo Nacional de Investigaciones Científicas y Técnicas de la Argentina (CONICET) por medio de una beca interna doctoral otorgada por RESOL-2021-154-APN-DIR#CONICET para el período 2021-2025.

Referencias

1. **Albrecht, A. J., Gaffney, J. E. (1983).** Software function, source lines of code, and development effort prediction: A software

science validation. IEEE Transactions on Software Engineering, Vol. 9, No. 6, pp. 639–648. DOI: 10.1109/TSE.1983.235271.

2. **Ali, N. Bin, Petersen, K. (2014).** Evaluating strategies for study selection in systematic literature studies. International Symposium on Empirical Software Engineering and Measurement, pp. 1–4. DOI: 10.1145/2652524.2652557.
3. **Allamanis, M., Sutton, C. (2013).** Mining source code repositories at massive scale using language modeling. 10th Working Conference on Mining Software Repositories MSR´13, pp. 207–216
4. **Bakır, A., Turhan, B., Bener, A. B. (2010).** A new perspective on data homogeneity in software cost estimation: A study in the embedded systems domain. Software Quality Journal, Vol. 18, No. 1, pp. 57–80. DOI: 10.1007/s11219-009-9081-z.
5. **Barone, A. V. M., Sennrich, R. (2017).** A parallel corpus of Python functions and documentation strings for automated code documentation and code generation. http://arxiv.org/abs/1707.02275.
6. **Boehm, B. W. (1981).** Software Engineering Economics. Springer Berlin Heidelberg.
7. **Chidamber, S. R., Kemerer, C. F. (1994).** A metrics suite for object oriented design. IEEE Transactions on Software Engineering, Vol. 20, No. 6
8. **Chidamber, S. R., Kemerer, C. F. (1991).** Towards a metrics suite for object oriented design. Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications OOPSLA'91, pp. 197–211. DOI: 10.1145/117954.117970.
9. **Cosentino, V., Luis, J., Izquierdo, C., Cabot, J. (2016).** Findings from GitHub: methods, datasets and limitations. Proceedings 13th Working Conference on Mining Software Repositories, MSR´16, pp. 137–141. DOI: 10.1145/2901739 .2901776.
10. **Crabtree, B. F., Miller, W. L. (1999).** Doing qualitative research 2nd ed. SAGE Publications, https://us.sagepub.com/enus/nam/doing-qualitative -research/book9279
11. **D'Ambros, M., Lanza, M., Robbes, R. (2012).** Evaluating defect prediction approaches: A benchmark and an extensive comparison. Empirical Software Engineering, Vol. 17, No. 4–5, pp. 531–577. DOI: 10.1007/s10664-011-9173-9.
12. **Desharnais, J. M. (1989).** Analyse statistique de la productivite des projets de developpement en informatique a partir de la technique des points de fonction. Masters Thesis University of Montreal
13. **Dyer, R., Nguyen, H. A., Rajan, H., Nguyen, T. N. (2015).** Boa: Ultra-large-scale software repository and source-code mining. ACM Transactions on Software Engineering and Methodology, Vol. 25, No. 1. DOI: 10.1145/2803171.
14. **Elmidaoui, S., Cheikhi, L., Idri, A. (2019).** Towards a taxonomy of software maintainability predictors. Advances in Intelligent Systems and Computing, Vol. 930, pp. 823–832. DOI: 10.1007/978-3-030-16181-1_77.
15. **Falessi, D., Smith, W., Serebrenik, A. (2017).** STRESS: A semi-automated, fully replicable approach for project selection. International Symposium on Empirical Software Engineering and Measurement, 2017-November, pp. 151–156. DOI: 10.1109/ESEM.2017.22.
16. **Forward, A., Lethbridge, T. C. (2008).** A taxonomy of software types to facilitate search and evidence-based software engineering. Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, No. 14, pp. 179–191. DOI: 10.1145/1463788.1463807.
17. **Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (2002).** Refactoring: Improving the design of existing code. pp. 1–337
18. **FRAMEndeley. (2021).** Chrome web store. https://chrome.google.com/webstore/detail/framendeley/decpeaebklmmgfhnnhggeikfhhlbcjpf?hl=es.
19. **Garvin, D. A. (1984).** What does "Product quality" really mean? MIT Sloan Management Review, pp. 25–43. https://sloanreview.mit.edu/article/what-does-product-quality-really-mean/.

20. **Githut 2.0: A Small Place To Discover Languages In GITHUB. (2021).** GitHut. https://madnight.github.io/githut.
21. **Goeminne, M., Mens, T. (2015).** Towards a survival analysis of database framework usage in Java projects. IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 551–555. DOI: 10.1109/ICSM.2015.7332512.
22. **Hamasaki, K., Gaikovina Kula, R., Yoshida, N., Camargo Cruz, A. E., Fujiwara, K., Naist, H. I. (2013).** Who Does What during a Code Review? Datasets of OSS Peer Review Repositories. 10th Working Conference on Mining Software Repositories (MSR). http://source.android.com/
23. **Harrison, R., Counsell, S., Nithi, R. (1998).** Coupling metrics for object-oriented design. Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No.98TB100262), DOI: 10.1109/METRIC. 1998.731240.
24. **Herraiz, I., Izquierdo-Cortazar, D., Rivas-Hernandez, F., Gonzalez-Barahona, J., Robles, G., Dueñas-Domínguez, S., Garcia-Campos, C., Gato, J. F., Tovar, L. (2009).** Flossmetrics: Free / libre / open source software metrics. Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, pp. 281–284. DOI: 10.1109/CSMR.2009.43.
25. **Higgins, J. P. T., Thomas, J., Chandler, J., Cumpston, M., Li, T., Page, M. J., Welch, V. A. (2019).** Cochrane Handbook for systematic reviews of interventions. J. P. T. Higgins, J. Thomas, J. Chandler, M. Cumpston, T. Li, M. J. Page, & V. A. Welch (Eds.), Cochrane Handbook for Systematic Reviews of Interventions. Wiley. DOI: 10.1002/9781 119536604.
26. **Howison, J., Conklin, M., Crowston, K. (2006).** FLOSSmole: A collaborative repository for FLOSS research data and analyses. International Journal of Information Technology and Web Engineering, Vol. 1, No. 3, pp. 17–26. DOI: 10.4018/jitwe.200 6070102.
27. **Janssen, M., van der Voort, H. (2020).** Agile and adaptive governance in crisis response: Lessons from the COVID-19 pandemic. International Journal of Information Management, Vol. 55, pp. 102180. DOI: 10.1016/j.ijinfomgt.2020.102180.
28. **Jureczko, M., Madeyski, L. (2010).** Towards identifying software project clusters with regard to defect prediction. Promise '10, Towards Identifying Software Project Clusters With Regard to Defect Prediction, No. 9, pp. 1–10. DOI: 10.1145/1868328. 1868342.
29. **Keivanloo, I., Rilling, J., Zou, Y. (2014).** Spotting working code examples. Proceedings International Conference on Software Engineering, Vol. 1, pp. 664–675. DOI: 10.1145/2568225.2568292.
30. **Kemerer, C. F. (1987).** An empirical validation of software cost estimation models. Communications of the ACM, Vol. 30, No. 5, pp. 416–429. DOI: 10.1145/22899.22906.
31. **Kitchenham, B. A., Budgen, D., Brereton, P. O. (2011).** Using mapping studies as the basis for further research - A participant-observer case study. Information and Software Technology, Vol. 53, No. 6, pp. 638–651. DOI: 10.1016/j.infsof.2010.12.011.
32. **Kitchenham, B. A., Dybå, T., Jørgensen, M. (2004).** Evidence-based software engineering. Proceedings of the 26th International Conference on Software Engineering, pp. 273–281. www.eviden cenetwork.org.
33. **Kitchenham, B., Brereton, P. (2013).** A systematic review of systematic review process research in software engineering. Information and Software Technology, Elsevier B.V., Vol. 55, No. 12, pp. 2049–2075. DOI: 10.1016/j.infsof. 2013.07.010.
34. **Kitchenham, B., Charters, S., (2007).** Guidelines for performing Systematic Literature Reviews in Software Engineering.pp. 1–57.
35. **Kitchenham, B., Kansala, K. (1993).** Inter-item correlations among function points. Proceedings First International Software Metrics Symposium, pp. 11–14. DOI: 10.1109/ METRIC.1993.263805.
36. **Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S.,**

Gibbs, S., Pohthong, A. (2017). Robust statistical methods for empirical software engineering. Empirical Software Engineering, Vol. 22, No. 2, pp. 579–630. DOI: 10.1007/s10 664-016-9437-5.

37. **Kitchenham, B., Pfleeger, S. L. (1996).** Software quality: the elusive target. IEEE Software, Vol. 13, No. 1, pp. 12–21. DOI: 10.1 109/52.476281.
38. **Knuth, D. E. (1971).** An empirical study of FORTRAN programs. Software: Practice and Experience, Vol. 1, No. 2, pp. 105–133. DOI: 10.1002/spe.4380010203.
39. **Lehman, M. M. (1996).** Laws of software evolution revisited. In: Montangero, C. (eds) Software Process Technology. EWSPT 1996. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg. Vol 1149. pp. 108–124. DOI: 10.1007/BFb0017737.
40. **Lokan, C., Wright, T., Hill, P. R., Stringer, M. (2001).** Organizational benchmarking using the ISBSG data repository. IEEE Software, Vol. 18, No. 5, pp. 26–32. DOI: 10. 1109/52.951491.
41. **Maibaum, T., Wassyng, A. (2008).** A product-focused approach to software certification. Computer, Vol. 41, No. 2, pp. 91–93. DOI: 10.1109/MC.2008.37.
42. **Maxwell, K. (2002).** Applied Statistics for Software Managers.
43. **Mendes, E., Di Martino, S., Ferrucci, F., Gravino, C. (2008).** Cross-company vs. single-company web effort models using the Tukutuku database: An extended study. Journal of Systems and Software, Vol. 81, No. 5, pp. 673–690. DOI: 10.1016/j.jss.20 07.07.044.
44. **Miyazaki, Y., Terakado, M., Ozaki, K., Nozaki, H. (1994).** Robust regression for developing software estimation models. The Journal of Systems and Software, Vol. 27, No. 1, pp. 3–16. DOI: 10.1016/01641212(94) 90110-4.
45. **Mkaouer, W., Kessentini, M., Bechikh, S., Deb, K., Cinnéide, M. Ó. (2014).** High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III. GECCO´14 In: Proceedings of the 2014 Genetic and Evolutionary Computation Conference, pp. 1263–1270. DOI: 10.1145/2576768.2598366.
46. **Mockus, A. (2009).** Amassing and indexing a large sample of version control systems: Towards the census of public source code history. Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR´09, pp. 11–20. DOI: 10.1109/MSR.2009. 5069476.
47. **Molléri, J. S., Petersen, K., Mendes, E. (2016).** Survey guidelines in software engineering: An annotated review. Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, No. 58, pp. 1–6. DOI: 10.1145/2961111.2962619
48. **Monperrus, M., Martinez, M. (2012).** CVS-Vintage: A Dataset of 14 CVS Repositories of Java Software. https://hal.archivesouver tes.fr/hal-0076 9121
49. **Orrú, M., Tempero, E., Marchesi, M., Tonelli, R., Destefanis, G. (2015).** A curated benchmark collection of python systems for empirical studies on software engineering. ACM International Conference Proceeding Series, pp. 1–4. DOI: 10.1 145/2810146. 2810148.
50. **Parnas, D. L. (2001).** Some software engineering principles. Software fundamentals: collected papers by David L. Parnas, pp. 257–266. DOI: 10.5555/376 584.376632.
51. **Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M. (2008).** Systematic Mapping Studies in Software Engineering. Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, pp. 68–77. www.splc.net.
52. **Petersen, K., Vakkalanka, S., Kuzniarz, L. (2015).** Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology, Vol. 64, pp. 1–18. DOI: 10.1016 /j.infsof.2015.03.007.
53. **Runeson, P., Höst, M. (2009).** Guidelines for conducting and reporting case study research in software engineering. Empirical Software

Engineering, Vol. 14, No. 2, pp. 131–164. DOI: 10.1007/s10664-008-9102-8.

54. **Saldaña, J. (2015).** The coding manual for qualitative researchers. SAGE Publications Ltd. https://us.sagepub.com/enus/nam/book/coding-manual-qualitative-researchers-1.
55. **Shepperd, M., Schofield, C. (1997).** Estimating software project effort using analogies. IEEE Transactions on Software Engineering, Vol. 23, No. 11, pp. 736–743. DOI: 10.1109/32.637387.
56. **Shepperd, M., Song, Q., Sun, Z., Mair, C. (2013).** Data quality: Some comments on the NASA software defect datasets. IEEE Transactions on Software Engineering, Vol. 39, No. 9, pp. 1208–1215. DOI: 10.1109/TSE.2013.11.
57. **Sheskin, D. J. (2000).** Parametric and nonparametric statistical procedures second Edition. www.crcpress.com
58. **Siegmund, J., Siegmund, N., Apel, S. (2015).** Views on internal and external validity in empirical software engineering. Proceedings - International Conference on Software Engineering, Vol. 1, pp. 9–19. DOI: 10.1109/ICSE.2015.24.
59. **Storey, M. A., Ernst, N. A., Williams, C., Kalliamvakou, E. (2020).** The who, what, how of software engineering research: a socio-technical framework. Empirical Software Engineering, Vol. 25, No. 5, pp. 4097–4129. DOI: 10.1007/s10664-020-09858-z.
60. **Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., Noble, J. (2010).** The qualitas corpus: A curated collection of Java code for empirical studies. Proceedings - Asia-Pacific Software Engineering Conference, APSEC, pp. 336–345. DOI: 10.1109/APSEC.2010.46.
61. **Terra, R., Miranda, L. F., Valente, M. T., Bigonha, R. S. (2013).** Qualitas.class corpus. ACM SIGSOFT Software Engineering Notes, Vol. 38, No. 5, pp. 1–4. DOI: 10.1145/2507288.2507314.
62. **Unterkalmsteiner, M., Gorschek, T., Islam, A. K. M. M., Cheng, C. K., Permadi, R. B., Feldt, R. (2012).** Evaluation and measurement of software process improvement-A systematic literature review. IEEE Transactions on Software Engineering Vol. 38, No. 2, pp. 398–424. DOI: 10.1109/TSE.2011.26.
63. **Vasilescu, B., Serebrenik, A., Filkov, V. (2015).** A Data Set for Social Diversity Studies of GitHub Teams. MSR ’15: Proceedings of the 12th Working Conference on Mining Software Repositories. https://github.com/bvasiles/ght.
64. **Wohlin, C., Rainer, A. (2021).** Challenges and recommendations to publishing and using credible evidence in software engineering. Information and Software Technology, Vol. 134, pp. 106555. DOI: 10.1016/j.infsof.2021.106555.
65. **Yu, Y., Wang, H., Filkov, V., Devanbu, P., Vasilescu, B. (2015).** Wait for It: Determinants of pull request evaluation latency on GitHub. IEEE International Working Conference on Mining Software Repositories, pp. 367–371. DOI. 10.1109/MSR.2015.42.
66. **Zerouali, A., Mens, T. (2017).** Analyzing the Evolution of Testing Library Usage in Open Source Java Projects. IEEE 24th International Conference on Software Analysis, Evolution and Reengineering SANER, pp. 417–421.
67. **Zhang, L., Tian, J. H., Jiang, J., Liu, Y. J., Pu, M. Y., Yue, T. (2018).** Empirical research in software engineering — A literature survey. Journal of Computer Science and Technology, Vol. 33, No. 5, pp. 876–899. DOI: 10.1007/s11390-018-1864-x.

Referencias estudios primarios

P1. **Accioly, P., Borba, P., Cavalcanti, G. (2018).** Understanding semi-structured merge conflict characteristics in open-source Java projects. Empirical Software Engineering, Vol. 23, No. 4, pp. 2051–2085. DOI: 10.1007/s10664-017-9586-1.

P2. **Ahmed, I., Brindescu, C., Mannan, U. A., Jensen, C., Sarma, A. (2017).** An empirical examination of the relationship between code smells and merge conflicts. International Symposium on Empirical

Software Engineering and Measurement, pp. 58–67. DOI: 10.1109/ESEM.2017.12.

P3. **Ahmed, I., Mannan, U. A., Gopinath, R., Jensen, C. (2015).** An empirical study of design degradation: How software projects get worse over time. International Symposium on Empirical Software Engineering and Measurement, pp. 31–40. DOI: 10.1109/ESEM.2015. 7321186.

P4. **Ajienka, N., Capiluppi, A., Counsell, S. (2017).** Managing hidden dependencies in oo software: A study based on open source projects. International Symposium on Empirical Software Engineering and Measurement, pp. 141–150. DOI: 10.1109/ ESEM. 2017.21.

P5. **Al Alam, S. M. D., Shahnewaz, S. M., Pfahl, D., Ruhe, G. (2014).** Monitoring bottlenecks in achieving release readiness. International Symposium on Empirical Software Engineering and Measurement, pp. 1–4. DOI: 10.1145/2652 524.2652549.

P6. **Al Dallal, J., Morasca, S. (2014).** Predicting object-oriented class reuse-proneness using internal quality attributes. Empirical Software Engineering, Vol. 19, No. 4, pp. 775–821. DOI: 10.1007/s10664-0129239-3.

P7. **Allix, K., Bissyandé, T. F., Jérome, Q., Klein, J., State, R., Le Traon, Y. (2016).** Empirical assessment of machine learning-based malware detectors for Android: Measuring the gap between in-the-lab and in-the-wild validation scenarios. Empirical Software Engineering, Vol. 21, No. 1, pp. 183–211. DOI: 10.1007/s10664-014-9352- 6.

P8. **Alnaeli, S. M., Maletic, J. I., Collard, M. L. (2016).** An empirical examination of the prevalence of inhibitors to the parallelizability of open source software systems. Empirical Software Engineering, Vol. 21, No. 3, pp. 1272–1301. DOI: 10.1007/s10664-015-9385-5.

P9. **AlOmar, E. A., Mkaouer, M. W., Ouni, A., Kessentini, M. (2019).** On the impact of refactoring on the relationship between quality attributes and design metrics. International Symposium on Empirical Software Engineering and Measurement.

P10. **Aman, H., Amasaki, S., Sasaki, T., Kawahara, M. (2015).** Empirical analysis of change-proneness in methods having local variables with long names and comments. International Symposium on Empirical Software Engineering and Measurement, pp. 50–53. DOI: 10.1109/ESEM.2015. 7321197.

P11. **Aman, H., Sasaki, T., Amasaki, S., Kawahara, M. (2014).** Empirical analysis of comments and fault-proneness in methods: Can comments point to faulty methods? International Symposium on Empirical Software Engineering and Measurement.

P12. **Arcelli Fontana, F., Mäntylä, M. V., Zanoni, M., Marino, A. (2016).** Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, Vol. 21, No. 3, pp. 1143–1191. DOI: 10.1007/s10664-015-9378-4.

P13. **Assar, S., Borg, M., Pfahl, D. (2016).** Using text clustering to predict defect resolution time: a conceptual replication and an evaluation of prediction accuracy. Empirical Software Engineering, Vol. 21, No. 4, pp. 1437–1475. DOI: 10.1007/s10664-015-9391-7.

P14. **Aué, J., Haisma, M., Tómasdóttir, K. F., Bacchelli, A. (2016).** Social diversity and growth levels of open source software projects on GitHub. International Symposium on Empirical Software Engineering and Measurement, pp. 1–6. DOI: 10.1145/2961111.2962633.

P15. **Bavota, G., De Lucia, A., Marcus, A., Oliveto, R. (2014).** Automating extract class refactoring: an improved method and its evaluation. Empirical Software Engineering, Vol. 19, No. 6, pp. 1617–1664. DOI: 10.1007/s10664-013-9256-x.

P16. **Beck, F., Diehl, S. (2013).** On the impact of software evolution on software clustering. Empirical Software Engineering, Vol. 18, No. 5, 970–1004. DOI: 10.1007/s10664-012-9225-9.

P17. **Behnamghader, P., Le, D. M., Garcia, J., Link, D., Shahbazian, A., Medvidovic, N. (2017).** A large-scale study of architectural

evolution in open-source software systems. Empirical Software Engineering, Vol. 22, No. 3, 1146–1193. DOI: 10.1007/s10664-016-9466-0.

P18. **Behnamghader, P., Meemeng, P., Fostiropoulos, I., Huang, D., Srisopha, K., Boehm, B. (2018).** A scalable and efficient approach for compiling and analyzing commit history. International Symposium on Empirical Software Engineering and Measurement, No. 27, p.p. 1–10. DOI: 10.1145/3239235.3239237

P19. **Beller, M., Zaidman, A., Karpov, A., Zwaan, R. A. (2017).** The last line effect explained. Empirical Software Engineering, Vol. 22, No. 3, 1508–1536. DOI: 10.1007/s10664-016-9489-6.

P20. **Bennin, K. E., Keung, J., Monden, A., Phannachitta, P., Mensah, S. (2017).** The significant effects of data sampling approaches on software defect prioritization and classification. International Symposium on Empirical Software Engineering and Measurement, pp. 364–373. DOI: 10.1109/ESEM.2017.50.

P21. **Bibiano, A. C., Fernandes, E., Oliveira, D., Garcia, A., Kalinowski, M., Fonseca, B., Oliveira, R., Oliveira, A., Cedrim, D. (2019).** A Quantitative study on characteristics and effect of batch refactoring on code smells. International Symposium on Empirical Software Engineering and Measurement, pp. 1–11. DOI: 10.1109/ESEM.2019.8870183.

P22. **Biggers, L. R., Bocovich, C., Capshaw, R., Eddy, B. P., Etzkorn, L. H., Kraft, N. A. (2014).** Configuring latent Dirichlet allocation based feature location. Empirical Software Engineering, Vol. 19, No. 3, pp. 465–500. DOI: 10.1007/s106 64-012-9224- x.

P23. **Callaú, O., Robbes, R., Tanter, É., Röthlisberger, D. (2013).** How (and why) developers use the dynamic features of programming languages: The case of smalltalk. Empirical Software Engineering, Vol. 18, No. 6, pp. 1156–1194. DOI: 10.1007/s10664-012-9203-2.

P24. **Campos, E. C., Maia, M. D. A. (2017).** Common Bug-Fix Patterns: A large-scale observational study. International Symposium on Empirical Software Engineering and Measurement, pp. 404–413. DOI: 10.1109/ESEM.2017.55.

P25. **Ceccato, M., Capiluppi, A., Falcarin, P., Boldyreff, C. (2015).** A large study on the effect of code obfuscation on the quality of java code. Empirical Software Engineering, Vol. 20, No. 6, pp. 1486–1524. DOI: 10.1007/s10664-014-9321-0.

P26. **Chen, B., Jack-Jiang, Z. M. (2017).** Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation. Empirical Software Engineering, Vol. 22, No. 1, pp. 330–374. DOI: 10.1007/s10664-016-9429-5.

P27. **Chen, N., Hoi, S. C. H., Xiao, X. (2014).** Software process evaluation: a machine learning framework with application to defect management process. Empirical Software Engineering, Vol. 19, No. 6, pp. 1531–1564. DOI: 10.1007/s10664-013-9254-z.

P28. **Chen, X., Slowinska, A., Bos, H. (2016).** On the detection of custom memory allocators in C binaries. Empirical Software Engineering, Vol. 21, No. 3, pp. 753–777. DOI: 10.1007/s10664-015-9362-z.

P29. **Cheung, W. T., Ryu, S., Kim, S. (2016).** Development nature matters: An empirical study of code clones in JavaScript applications. Empirical Software Engineering, Vol. 21, No. 2, pp. 517–564. DOI: 10.1007/s10664-015-9368-6.

P30. **Coelho, R., Almeida, L., Gousios, G., van Deursen, A., Treude, C. (2017).** Exception handling bug hazards in Android: Results from a mining study and an exploratory survey. Empirical Software Engineering, Vol. 22, No. 3, pp. 1264–1304. DOI: 10.1007/s10664-016-9443-7.

P31. **Corazza, A., Di Martino, S., Ferrucci, F., Gravino, C., Sarro, F., Mendes, E. (2013).** Using tabu search to configure support vector regression for effort estimation. Empirical Software Engineering, Vol. 18, No.

3, pp. 506–546. DOI: 10.1007/s10664-011-9187-3.

P32. **Corazza, Anna, Di Martino, S., Maggio, V., Scanniello, G. (2016).** Weighing lexical information for software clustering in the context of architecture recovery. Empirical Software Engineering, Vol. 21, No. 1, pp. 72–103. DOI: 10.1007/s10664-014-9347-3.

P33. **da Costa, D. A., McIntosh, S., Kulesza, U., Hassan, A. E., Abebe, S. L. (2018).** An empirical study of the integration time of fixed issues. Empirical Software Engineering, Vol. 23, No. 1, pp. 334–383. DOI: 10.1007/s10664-017-9520-6.

P34. **Dashevskyi, S., Brucker, A. D., Massacci, F. (2019).** A screening test for disclosed vulnerabilities in FOSS components. IEEE Transactions on Software Engineering, Vol. 45, No. 10, pp. 945–966. DOI: 10.1109/TSE. 2018.2816033.

P35. **De O. Barros, M. (2014).** An experimental evaluation of the importance of randomness in hill climbing searches applied to software engineering problems. Empirical Software Engineering, Vol. 19, No. 5, pp. 1423–1465. DOI: 10.1007/s10664-013-9294-4.

P36. **Dit, B., Revelle, M., Poshyvanyk, D. (2013**). Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. Empirical Software Engineering, Vol. 18, No. 2, pp. 277–309. DOI: 10.1007/s10664-011-9194-4.

P37. **Elish, M. O., Al-Ghamdi, Y. (2015).** Fault density analysis of object-oriented classes in presence of code clones. International Conference on Evaluation and Assessment in Software Engineering, pp. 1–7. DOI: 10.1 145/2745802. 2745811.

P38. **Eyolfson, J., Tan, L., Lam, P. (2014).** Correlations between bugginess and time-based commit characteristics. Empirical Software Engineering, Vol. 19, No. 4, pp. 1009–1039. DOI: 10.1007/s10664-013-9245-0.

P39. **Fagerholm, F., Guinea, A. S., Münch, J., Borenstein, J. (2014).** The role of mentoring and project characteristics for onboarding in open source software projects. International Symposium on Empirical Software Engineering and Measurement, No. 55, pp. 1–10. DOI: 10.1145/2652524.2652540.

P40. **Fan, Q., Yu, Y., Yin, G., Wang, T., Wang, H. (2017).** Where Is the road for issue reports classification based on text mining? International Symposium on Empirical Software Engineering and Measurement, pp. 121–130. DOI: 10.1109/ESEM.2017.19.

P41. **Foucault, M., Falleri, J. R., Blanc, X. (2014).** Code ownership in open-source software. International Conference on Evaluation and Assessment in Software Engineering, pp. 1–9. DOI: 10.1145/26012 48.2601283.

P42. **Fraser, G., Arcuri, A. (2015).** 1600 faults in 100 projects: automatically finding faults while achieving high coverage with evosuite. Empirical Software Engineering, Vol. 20, No. 3, pp. 611–639. DOI: 10.1007/s10664-013-9288-2.

P43. **Fraser, G., Arcuri, A. (2012).** Sound empirical evidence in software testing. Proceedings International Conference on Software Engineering, pp. 178–188. DOI: 10.1109/IC SE.2012.6227195.

P44. **Fu, S., Shen, B. (2015**). Code bad smell detection through evolutionary data mining. International Symposium on Empirical Software Engineering and Measurement, pp. 1–9. DOI: 10.1109/ESEM.2015. 7321194.

P45. **Gallaba, K., Mesbah, A., Beschastnikh, I. (2015).** Don't call us, we'll call you: characterizing Callbacks in Javascript. International Symposium on Empirical Software Engineering and Measurement, pp. 1–10. DOI: 10.1109/ESEM.2015. 7321196.

P46. **Gousios, G., Spinellis, D. (2014).** Conducting quantitative software engineering studies with Alitheia Core. Empirical Software Engineering, Vol. 19, No. 4, pp. 885–925. DOI: 10.1007/ s10664-013-9242-3.

P47. **Haller, I., Slowinska, A., Bos, H. (2016**). Scalable data structure detection and classification for C/C++ binaries. Empirical Software Engineering, Vol. 21, No. 3, pp. 778–810. DOI: 10.1007/s10664-015-9363- y.

P48. **Hassan, F., Mostafa, S., Lam, E. S. L., Wang, X. (2017).** Automatic building of Java projects in software repositories: A study on feasibility and challenges. International Symposium on Empirical Software Engineering and Measurement, pp. 38–47. DOI: 10.1109/ESEM .2017.11.

P49. **He, Z., Peters, F., Menzies, T., Yang, Y. (2013).** Learning from open-source projects: An empirical study on defect prediction. International Symposium on Empirical Software Engineering and Measurement, pp. 45–54. DOI: 10.1109 /ESEM.2013.20.

P50. **Hebig, R., Derehag, J., Chaudron, M. R. V. (2015).** Identifying metrics' biases when measuring or approximating size in heterogeneous languages. International Symposium on Empirical Software Engineering and Measurement, pp. 1–4. DOI: 10.1109/ ESEM.2015.7321201.

P51. **Herzig, K., Just, S., Zeller, A. (2016).** The impact of tangled code changes on defect prediction models. Empirical Software Engineering, Vol. 21, No. 2, pp. 303–336. DOI: 10.1007/s10664-015-9376-6.

P52. **Hindle, A., Alipour, A., Stroulia, E. (2016).** A contextual approach towards more accurate duplicate bug report detection and ranking. Empirical Software Engineering, Vol. 21, No. 2, pp. 368–410. DOI: 10.1007/s10664-015-9387-3.

P53. **Hindle, A., Ernst, N. A., Godfrey, M. W., Mylopoulos, J. (2011).** Automated topic naming to support cross-project analysis of software maintenance activities. Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 163–172. DOI: 10.1145/1985 441.1985466.

P54. **Hora, A., Robbes, R. (2020).** Characteristics of method extractions in Java: a large scale empirical study. Empirical Software Engineering, Vol. 25, pp. 1798–1833. DOI: 10.1007/s10664-020-09809-8.

P55. **Hunsen, C., Zhang, B., Siegmund, J., Kästner, C., Leßenich, O., Becker, M., Apel, S. (2016).** Preprocessor-based variability in open-source and industrial software systems: An empirical study. Empirical Software Engineering, Vol. 21, No. 2, pp. 449–482. DOI: 10.1007/s10664-015-9360-1.

P56. **Islam, M. R., Zibran, M. F., Nagpal, A. (2017).** Security vulnerabilities in categories of clones and non-cloned code: an empirical study. International Symposium on Empirical Software Engineering and Measurement, pp. 20–29. DOI: 10.1109/ESEM.2017.9.

P57. **Jaafar, F., Guéhéneuc, Y. G., Hamel, S., Khomh, F., Zulkernine, M. (2016).** Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults. Empirical Software Engineering, Vol. 21, No. 3, pp. 896–931. DOI: 10.1007/s10664-015-9361-0.

P58. **Kabinna, S., Bezemer, C. P., Shang, W., Syer, M. D., Hassan, A. E. (2018).** Examining the stability of logging statements. Empirical Software Engineering, Vol. 23, No. 1, pp. 290–333. DOI: 10.1007/s10664-017-9518-0.

P59. **Kagdi, H., Gethers, M., Poshyvanyk, D. (2013).** Integrating conceptual and logical couplings for change impact analysis in software. Empirical Software Engineering, Vol. 18, No. 5, pp. 933–969. DOI: 10.1007/s10664-012-9233-9.

P60. **Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., Hassan, A. E. (2016).** Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering, Voo. 21, No. 5, pp. 2072–2106. DOI: 10.1007/s10664-015-9400-x.

P61. **Khatibi Bardsiri, V., Jawawi, D. N. A., Hashim, S. Z. M., Khatibi, E. (2014).** A flexible method to estimate the software development effort based on the classification of projects and localization of comparisons. Empirical Software

Engineering, Vol. 19, No. 4, pp. 857–884. DOI: 10.1007/ s10664-013-9241-4.

P62. **Kifetew, F. M., Tiella, R., Tonella, P. (2017).** Generating valid grammar-based test inputs by means of genetic programming and annotated grammars. Empirical Software Engineering, Vol. 22, No. 2, pp. 928–961. DOI: 10.1007/s10664-015-9422-4.

P63. **Kim, S., Kim, D. (2016).** Automatic identifier inconsistency detection using code dictionary. Empirical Software Engineering, Vol. 21, No. 2, pp. 565–604. DOI: 10.1007/s 10664-015-9369-5.

P64. **Kula, R. G., German, D. M., Ouni, A., Ishio, T., Inoue, K. (2018).** Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration. Empirical Software Engineering, Vol. 23, No. 1, pp. 384–417. DOI: 10.1007/s10664-017-9521-5.

P65. **Lavazza, L., Morasca, S. (2016).** Identifying thresholds for software faultiness via optimistic and pessimistic estimations. International Symposium on Empirical Software Engineering and Measurement, pp. 1–10. DOI: 10.1145/2961 111.2962595.

P66. **Lee, T., Gu, T., Baik, J. (2014).** MND-SCEMP: An empirical study of a software cost estimation modeling process in the defense domain. Empirical Software Engineering, Vol. 19, No. 1, pp. 213–240. DOI: 10.1007/s10664-012-9220-1.

P67. **Linares-Vásquez, M., McMillan, C., Poshyvanyk, D., Grechanik, M. (2014).** On using machine learning to automatically classify software applications into domain categories. Empirical Software Engineering, Vol. 19, No. 3, 582–618. DOI: 10.1007/s 10664-012-9230-z.

P68. **Lokan, C., Mendes, E. (2017).** Investigating the use of moving windows to improve software effort prediction: a replicated study. Empirical Software Engineering, Vol. 22, No. 2, pp, 716–767. DOI: 10.1007/s10664-016-9446-4.

P69. **Malhotra, R., Khanna, M. (2017).** An empirical study for software change prediction using imbalanced data. Empirical Software Engineering, Vol. 22, No. 6, pp. 2806–2851. DOI: 10.1007/s10664-016-9488-7.

P70. **Malloy, B. A., Power, J. F. (2017).** Quantifying the transition from python 2 to 3: An empirical study of python applications. International Symposium on Empirical Software Engineering and Measurement, pp. 314–323. DOI: 10.1109/ ESEM.2017.45.

P71. **Martinez, M., Monperrus, M. (2013).** Mining software repair models for reasoning on the search space of automated program fixing. Empirical Software Engineering, Vol. 20, No. 1, pp. 176–205. DOI: 10.1007/s106 64-013-9282-8.

P72. **Mayer, P., Bauer, A. (2015).** An empirical analysis of the utilization of multiple programming languages in open source projects. International Conference on Evaluation and Assessment in Software Engineering, No. 4, pp. 1–10. DOI: 10.1145/ 2745802.2745805.

P73. **McIlroy, S., Ali, N., Khalid, H., E. Hassan, A. (2016).** Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. Empirical Software Engineering, Vol. 21, No. 3, pp. 1067–1106. DOI: 10.1007/s10664-015-9375-7.

P74. **McIntosh, S., Kamei, Y., Adams, B., Hassan, A. E. (2016).** An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering, Vol. 21, No. 5, pp. 2146–2189. DOI: 10.1007/s10664-015-9381-9.

P75. **McIntosh, S., Nagappan, M., Adams, B., Mockus, A., Hassan, A. E. (2015).** A large-scale empirical study of the relationship between build technology and build maintenance. Empirical Software Engineering, Vol. 20, No. 6, pp. 1587–1633. DOI: 10.1007/s10664-014-9324-x.

P76. **Medeiros, F., Lima, G., Amaral, G., Apel, S., Kästner, C., Ribeiro, M., Gheyi, R. (2019).** An investigation of misunderstanding code patterns in C open-source software projects. Empirical Software Engineering, Vol. 24, No. 4, pp.

1693–1726. DOI: 10.1007/s10664-018-9666-x.

P77. **Misirli, A. T., Shihab, E., Kamei, Y. (2016).** Studying high impact fix-inducing changes. Empirical Software Engineering, Vol. 21, No. 2, pp. 605–641. DOI: 10.1007/s10664-015-9370-z.

P78. **Mkaouer, M. W., Kessentini, M., Bechikh, S., Ó Cinnéide, M., Deb, K. (2016).** On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. Empirical Software Engineering, Vol. 21, No. 6, pp. 2503–2545. DOI: 10.1007/s10664-015-9414-4.

P79. **Mkaouer, M. W., Kessentini, M., Cinnéide, M., Hayashi, S., Deb, K. (2017).** A robust multi-objective approach to balance severity and importance of refactoring opportunities. Empirical Software Engineering, Vol. 22, No. 2, pp. 894–927. DOI: 10.1007/s10664-016-9426-8.

P80. **Morasca, S., Lavazza, L. (2019).** Comparing the effectiveness of using design and code measures in software faultiness estimation. International Conference on Evaluation and Assessment in Software Engineering, pp. 112–121. DOI: 10.114 5/3319008.3319026.

P81. **Morasca, S., Lavazza, L. (2016).** Slope-based fault-proneness thresholds for software engineering measures. International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10. DOI: 10.1145/29 15970.2915997.

P82. **Mori, T., Tamura, S., Kakui, S. (2013).** Incremental estimation of project failure risk with Naïve bayes classifier. International Symposium on Empirical Software Engineering and Measurement, pp. 283–286. DOI: 10.1109/ES EM.2013.40.

P83. **Munaiah, N., Kroh, S., Cabrey, C., Nagappan, M. (2017).** Curating GitHub for engineered software projects. Empirical Software Engineering, Vol. 22, No. 6, pp. 3219–3253. DOI: 10.1007/s10664-017-9512-6.

P84. **Cinnéide, M. O., Hemati-Moghadam, I., Harman, M., Counsell, S., Tratt, L. (2017).** An experimental search-based approach to cohesion metric evaluation. Empirical Software Engineering, Vol. 22, No. 1, pp. 292–329. DOI: 10.1007/s10664-016-9427- 7.

P85. **Okutan, A., Yıldız, O. T. (2014).** Software defect prediction using Bayesian networks. Empirical Software Engineering, Vol. 19, No. 1, pp. 154–181. DOI: 10.1007/s10664-012-9218-8.

P86. **Parnin, C., Bird, C., Murphy-Hill, E. (2013).** Adoption and use of Java generics. Empirical Software Engineering, Vol. 18, No. 6, pp. 1047–1089. DOI: 10.1007/s10664-012-9236-6.

P87. **Parsai, A., Murgia, A., Demeyer, S. (2016).** Evaluating random mutant selection at class-level in projects with non-adequate test suites. International Conference on Evaluation and Assessment in Software Engineering, No. 11, pp. 1–10. DOI: 10.1145 /2915970.2915992.

P88. **Petrić, J., Bowes, D., Hall, T., Christianson, B., Baddoo, N. (2016).** Building an ensemble for software defect prediction based on diversity selection. International Symposium on Empirical Software Engineering and Measurement, 08-09-Sept, No. 46, pp. 1–10. DOI: 10.1145/ 2961111. 2962610.

P89. **Petrić, J., Bowes, D., Hall, T., Christianson, B., Baddoo, N. (2016).** The Jinx on the NASA software defect data sets. International Conference on Evaluation and Assessment in Software Engineering, No. 13, pp. 1–5. DOI: 10.1145/2915970.2 916007.

P90. **Petrić, J., Galinac Grbac, T. (2014).** Software structure evolution and relation to system defectiveness. International Conference on Evaluation and Assessment in Software Engineering, No. 34, pp. 1–10. DOI: 10.1145/2 601248.2601287.

P91. **Phannachitta, P., Keung, J., Monden, A., Matsumoto, K. (2017).** A stability assessment of solution adaptation techniques for analogy-based software

effort estimation. Empirical Software Engineering, Vol. 22, No. 1, pp. 474–504. DOI: 10.1007/s10664-016-9434-8.

P92. **Phannachittay, P., Mondeny, A., Keungz, J., Matsumotoy, K. (2015).** Case consistency: A necessary data quality property for software engineering data sets. International Conference on Evaluation and Assessment in Software Engineering, No. 19, pp. 1–10. DOI: 10.1145/274580 2.2745820.

P93. **Quesada-López, C., Jenkins, M. (2014).** Function point structure and applicability validation using the ISBSG dataset: A replicated study. International Symposium on Empirical Software Engineering and Measurement, No. 66, pp.1. DOI: 10.1145/2652524.2652595.

P94. **Raja, U. (2013).** All complaints are not created equal: Text analysis of open source software defect reports. Empirical Software Engineering, Vol. 18, No. 1, pp. 117–138. DOI: 10.1007/s10 664-012-9197-9.

P95. **Rakha, M. S., Shang, W., Hassan, A. E. (2016).** Studying the needed effort for identifying duplicate issues. Empirical Software Engineering, Vol. 21, No. 5, pp. 1960–1989. DOI: 10.1007/s10664-015-9404-6.

P96. **Ramírez, A., Romero, J. R., Ventura, S. (2016).** A comparative study of many-objective evolutionary algorithms for the discovery of software architectures. Empirical Software Engineering, Vol. 21, No. 6, pp. 2546–2600. DOI: 1007/s10664-015-9399-z.

P97. **Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., Riquelme, J. C. (2014).** Preliminary comparison of techniques for dealing with imbalance in software defect prediction. International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10. DOI: 10.1145/2601248.2601294.

P98. **Rodriguez, I., Wang, X. (2017).** An empirical study of open source virtual reality software projects. In: International Symposium on Empirical Software Engineering and Measurement, pp. 474–475. DOI: 10.1109/ESEM.2017.65.

P99. **Rojas, J. M., Vivanti, M., Arcuri, A., Fraser, G. (2017).** A detailed investigation of the effectiveness of whole test suite generation. Empirical Software Engineering, Vol. 22, No. 2, pp. 852–893. DOI: 10.1007 /s10664-015-9424-2.

P100. **Rosa, W., Madachy, R., Boehm, B., Clark, B. (2014).** Simple empirical software effort estimation model. International Symposium on Empirical Software Engineering and Measurement, No. 43, pp. 1–4. DOI: 10.1145 /2652524.2652558.

P101. **Ryu, D., Choi, O., Baik, J. (2016**). Value-cognitive boosting with a support vector machine for cross-project defect prediction. Empirical Software Engineering, Vol. 21, No. 1, pp. 43–71. DOI: 10.1007/s10664014 9346-4.

P102. **Sawant, A. A., Bacchelli, A. (2017).** fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage. Empirical Software Engineering, Vol. 22, No. 3, pp. 1348–1371. DOI: 10.1007/s10 664-016-9444-6.

P103. **Scanniello, G., Marcus, A., Pascale, D. (2015).** Link analysis algorithms for static concept location: an empirical assessment. Empirical Software Engineering, Vol. 20, No. 6, pp. 1666–1720. DOI: 10.1007/s10664-014-9327-7.

P104. **Scholtes, I., Mavrodiev, P., Schweitzer, F. (2016).** From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. Empirical Software Engineering, Vol. 21, No. 2, pp. 642–683. DOI: 10.1007/ s10664-015-9406-4.

P105. **Sharma, T., Fragkoulis, M., Spinellis, D. (2017).** House of cards: code smells in open-source c# repositories. International Symposium on Empirical Software Engineering and Measurement, pp. 424–429. DOI: 10.1109/ ESEM.2017.57.

P106. **Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E., Matsumoto, K. I. (2013).** Studying re-

opened bugs in open source software. Empirical Software Engineering, Vol. 18, No. 5, PP. 1005–1042. DOI: 10.1007/s10664-012-9228-6.

P107. Shippey, T., Hall, T., Counsell, S., Bowes, D. (2016). So you need more method level datasets for your software defect prediction? Voilà! International Symposium on Empirical Software Engineering and Measurement, pp. 1–6. DOI: 10.1145/29611 11.2962620.

P108. Soetens, Q. D., Demeyer, S., Zaidman, A., Pérez, J. (2016). Change-based test selection: an empirical evaluation. Empirical Software Engineering, Vol. 21, No. 5, pp. 1990–2032. DOI: 10.1007/s10664-015-9405-5.

P109. Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., & Zhai, C. (2014). Bug characteristics in open source software. Empirical Software Engineering, Vol. 19, No. 6, pp. 1665–1705. DOI: 10.1007/s10664-013-9258-8.

P110. Thomas, S. W., Hemmati, H., Hassan, A. E., Blostein, D. (2014). Static test case prioritization using topic models. Empirical Software Engineering, Vol. 19, No. 1, pp. 182–212. DOI: 10.1007/s10664-012-9219- 7.

P111. Thongtanunam, P., McIntosh, S., Hassan, A. E., Iida, H. (2017). Review participation in modern code review: An empirical study of the android, Qt, and OpenStack projects. Empirical Software Engineering, Vol. 22, No. 2, pp. 768–817. DOI: 10.1007/s10664-016-9452-6.

P112. Tian, Y., Ali, N., Lo, D., Hassan, A. E. (2016). On the unreliability of bug severity data. Empirical Software Engineering, Vol. 21, No. 6, pp. 2298–2323. DOI: 10.1007/s10664-015-9409-1.

P113. Vidal, S. A., Bergel, A., Marcos, C., Díaz-Pace, J. A. (2016). Understanding and addressing exhibitionism in Java empirical research about method accessibility. Empirical Software Engineering, Vol. 21, No. 2, pp. 483–516. DOI: 10.1007/s10664-015-9365-9.

P114. Walkinshaw, N., Minku, L. (2018). Are 20% of files responsible for 80% of defects? International Symposium on Empirical Software Engineering and Measurement. No. 2, pp. 1–10. DOI: 10.1145/3239235.3239244.

P115. Wood, M. I., Ivanov, L., Lamprou, Z. (2019). An analysis of inheritance hierarchy evolution. International Conference on Evaluation and Assessment in Software Engineering, pp. 24–33. DOI: 10.1145/3319008.3319023.

P116. Wu, D., Chen, L., Zhou, Y., Xu, B. (2015). An empirical study on C++ concurrency constructs. International Symposium on Empirical Software Engineering and Measurement, pp. 257–266. DOI: 10.1109/ESEM.2015.7321187.

P117. Xia, X., Shihab, E., Kamei, Y., Lo, D., Wang, X. (2016). Predicting crashing releases of mobile applications. International Symposium on Empirical Software Engineering and Measurement, pp. 1–10. DOI: 10.1145/29611 11.2962606.

P118. Yan, M., Fang, Y., Lo, D., Xia, X., Zhang, X. (2017). File-level defect prediction: unsupervised vs. Supervised models. International Symposium on Empirical Software Engineering and Measurement, pp. 344–353. DOI: 10.1109/ESEM.2017.48.

P119. Zhang, F., Mockus, A., Keivanloo, I., Zou, Y. (2016). Towards building a universal defect prediction model with rank transformed predictors. Empirical Software Engineering, Vol. 21, No. 5, pp. 2107–2145. DOI: 10.1007/s10664-015-9396-2.

P120. Zhao, Y., Zhang, F., Shihab, E., Zou, Y., Hassan, A. E. (2016). How are discussions associated with bug reworking? An empirical study on open source projects. International Symposium on Empirical Software Engineering and Measurement, pp. 1–10. DOI: 10.1145/296 1111.2962591.

P121. Zhou, B., Neamtiu, I., Gupta, R. (2015). A cross-platform analysis of bugs and bug-fixing in open source projects: Desktop vs. Android vs. iOS. In: International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10. DOI: 10.1145/2745802.2745808.

P122. Zhu, J., Zhou, M., Mockus, A. (2014). Patterns of folder use and project popularity: A case study of github repositories. International Symposium on Empirical Software Engineering and Measurement. No. 30, pp. 1–4. DOI: 10.11 45/2652524.2 652564.

Article received on 31/05/2021; accepted on 18/08/2022.
Corresponding author is Juan Andrés Carruther.