electronics

MDPI

Article

Enhancing Software Project Monitoring with Multidimensional Data Repository Mining

Łukasz Reszka, Janusz Sosnowski * and Bartosz Dobrzyński

Institute of Computer Science, Warsaw University of Technology, 00-665 Warsaw, Poland; lukasz.reszka.stud@pw.edu.pl (Ł.R.); dobrzynski.b@gmail.com (B.D.)
* Correspondence: janusz.sosnowski@pw.edu.pl

Abstract: Software project development and maintenance activities have been reported in various repositories. The data contained in these repositories have been widely used in various studies on specific problems, e.g., predicting bug appearance, allocating issues to developers, and identifying duplicated issues. Developed analysis schemes are usually based on simplified data models while issue report details are neglected. Confronting this problem requires a deep and wide-ranging exploration of software repository contents adapted to their specificities, which differs significantly from classical data mining. This paper is targeted at three aspects: the structural and semantic exploration of repositories, deriving characteristic features in value and time perspectives, and defining the space of project monitoring goals. The considerations presented demonstrate a holistic image of the project development process, which is useful in the assessment of its efficiency and identification of imperfections. The original analysis introduced in this work was verified using open source and some commercial software project repositories.

Keywords: software project monitoring; software repositories; issue-tracking systems; data exploration; project lifecycle

Citation: Reszka, Ł.; Sosnowski, J.; Dobrzyński, B. Enhancing Software Project Monitoring with Multidimensional Data Repository Mining. *Electronics* **2023**, *12*, 3774. https://doi.org/10.3390/electronics12183774

Academic Editors: Iouliia Skliarova, Jinhyun Kim, Seonah Lee and Suwon Lee

Received: 27 June 2023
Revised: 14 August 2023
Accepted: 16 August 2023
Published: 6 September 2023

1. Introduction

Software project development and maintenance need tracing of relevant reports in software repositories, in particular those generated via issue tracking (e.g., Bugzilla, Jira, Mantis) and software version control systems (e.g., GitHub). This has been widely discussed in the literature, e.g., [1–3]. An issue-tracking system (ITS) manages the running of reports on recorded problems, their processing status and progress, and their final resolution (e.g., completed, fixed, rejected, not a problem, and duplicated). Each issue is uniquely labelled by the relevant identifier (ticket). In many cases, it can be correlated with code corrections, updates, or changes documented in software version control systems (SVC). Software repositories are useful in managing project development and maintenance activities; however, due to project time pressure, the information contained in them is not sufficiently explored by involved actors and available tools. On the other hand, they have been studied in many research papers, mostly with respect to three aspects: bug predictions [4,5], problem resolution allocation (triaging) [6], and identifying duplicated requests [7]. Many studies have focused on mining issue textual descriptions to derive issue features, e.g., [8–10] and references therein. Diverse classification schemes have been proposed to identify issue types (bug, non-bug, code improvement, new functionality, and technical tasks) and categorize issue comments (outlined in Section 2).

Most publications only deal with a single problem (as listed above) and do not examine development processes and their context from a wider perspective. Typically, they are based on general (coarse-grained) repository data features. This flat model of issues mostly uses issue inflow, timing, and description features. Having analyzed a wide scope of software project repositories (open source and commercial ones), we noticed that available rich data

are partially used in research and practice. In software projects, most issues comprise at least a few dozen attributes with specified meaningful and valuable information. Hence, we found a need for a holistic and extensive investigation of these data with respect to software development and maintenance processes. Our research covered two questions:

RQ1—what kind of information is available in issue repository attributes; what the range of its richness, diversity, and accuracy are; and what the scope of its usage in software engineering studies is.

RQ2—what kind of metrics can be used to assess the quality of issue tracking systems and how they characterize software project development processes.

When dealing with these questions, we surveyed the literature relevant to issue tracking and verified the scope of the issue attributes in the studies we performed. This resulted in the development of a methodology for deeper analysis, followed by data crawling from ITS repositories and their storage in a practical database. It included data parsing of raw data, extracting useful semantics (characteristic features), and defining investigation topics. It was supported with metrics introduced to assess the scope and information accuracy of repository contents, identify anomalies, and suggest possible improvements. The scope of the research is presented gradually in successive sections according to a top-down (general to detailed) problem investigation scheme. The filling of the research gap in the literature is outlined in the subsequent section and summarized in the final discussion of the paper, based on the results of the studies performed.

We developed data models with appropriate exploration schemes, and we split our studies into four interrelated tasks:

- The taxonomy of issue-reporting dimensions (structural, semantic, and observation perspectives) relevant to available attributes, based on a generalized ITS model.
- Deriving an analysis methodology enhanced with profiles and metrics characterizing the contents of the issue reports (the filling ratios of attributes and distribution of assigned values), project actor activities, time features, etc.
- Investigating issue-reporting dependencies in time and cross-attribute correlations.
- The identification of software repository deficiencies.

In this study, we adopted and extended various statistical and text-mining algorithms considering the specificities of repositories. They differed significantly from classical data-mining approaches. The evaluations performed related to different perspectives (global and local), revealed correlations in repository data, and provided the capability for comparing activities within an analyzed project, company, or larger project domain. Data extraction and analysis were supported with specially developed tools and relevant databases, which provided illustrative results for real projects. We outlined the impact of diverse issue features on the accuracy of assessing issue-handling processes. The methodology and results presented would be especially useful for project managers; nevertheless, the knowledge derived from issue reports could have a positive impact on other project stakeholders, as it could help to improve their interactions with repositories. It could also be useful in software engineering research, since it improved the resiliency of the studies performed.

Section 2 outlines the background of our research with respect to other publications in the literature. Section 3 provides an overview of software repositories and their syntactic, semantic, and temporal features; it presents the methodology of the multidimensional analysis developed. A systematic assessment of issue-reporting contents is presented in Section 4. Section 5 provides the metrics for assessing project actor activities. Issue dependencies (correlations) are considered in Section 6. Sections 7 and 8 discuss and summarize the results of the research, respectively.

2. Literature Review and Problem Background

Software development, maintenance, and quality issues can be evaluated by tracing diverse software repositories, such as issue tracking, version control, test, or recommendation ones. These repositories provide the capability for communication between project stakeholders (developers and users) to report or comment on bugs and request improvements

or feedback. Issue reports reflect problems and the progress of the development process. Software developers use ITS systems to manage software bugs, improvements, code change requests, development tasks, release planning, etc. Publications related to reliability and bug handling dominate in the literature. On the basis of bug reports (the time between reported defects or the number of detected ones in subsequent periods), we can predict unrevealed defects. For this purpose, many reliability growth models have been developed and examined, e.g., [11]. Other approaches consider product features (correlation of code complexity metrics and defect density), process, and issue features, e.g., [12].

When analyzing issue reports we can trace project development processes, roles of stakeholders, project lifecycle, issue inflow [13], and relevant timing statistics. The issue tracker described in [14] allows us to correlate issues with relevant commits, code branches, and file paths. Bug handling processes are analyzed in publications targeted at specific aspects such as: (i) efficiency of these processes [15], supported via tools to estimate costs (human resources, time, and the identification of risks using simulation models); and (ii) detection of duplicated defects (to eliminate redundant handling actions [7]); (iii) allocating issues for resolution to appropriate developers (bug triaging). An important aspect in issue handling is identifying duplicated issue reports at the time of filing an issue report—just-in-time duplicate retrieval [16]. In [17], the software development risk is assessed using simulation modelling (targeted at an agile scheme) based on data from the Jira tool and derived features, e.g., development activity and the number of reported issues. Typically, bug triaging is performed by creating recommendation lists or rankings of issue fixers [18], creating project teams [19], and mapping developers with bug reports [6]. The problem of matching project contributors' skills to tasks for resolution is discussed in [20]. Bug triaging uses API-domain labels to indicate the skills involved in an issue. The labels represent possible libraries of the source code linked to an issue. An automated selection of a suitable software developer to resolve a bug in an efficient and timely way is presented in [21]. It refers to diverse parameters having an impact on finding the optimal developer selection task. Experiment results have confirmed the following sequence of tracing attributes in bug reports: component, priority, and operating system. In [22], the process of assigning bugs to the right developer or team uses the bug summary, description, and responses to bug reports at the time of their closing. These textual attributes are enhanced with additional derived features such as product, customer, site, priority, issue reporter, configuration, and project generation.

Reported issues may need different treatment (processing), depending on their types and specified explicitly in relevant repository fields or labels. Issue labelling can be used to provide specific information or suggest decisions [23] for further processing and faster resolution. Quite often, issue specifications can be incorrect or skipped by the issue reporter. It can be completed by exploring the description included. For this purpose, text mining techniques targeted at the detection of the searched issue categories have been developed. They relate to discriminating bug and non-bug issues [24], duplicated issues [7], security problem issues [25,26], issues that do not need fixing (the so called "won't fix" issues [27], e.g., due to postponing to another version, negligible impact, etc.), bugs that are related to performance degradation, or software aging [28]. The predictions of some bug properties (e.g., severity, assignee, and duplicated reports) support software evolution and maintenance.

The neural network RoBERT is used in [8] to classify issue reports into three categories: bug, enhancement, and question. However, this is not consistent with the scope of reported issues in various projects. This is in opposition to keyword-based approaches neglecting text context. In [29], the authors deal with the problem of approving or rejecting issues for further processing. They use natural language processing techniques for the sentiment analysis (positive and negative) of the issue summary, based on words frequently used in reports. Each pre-processed report and its sentiment are converted into a feature vector used in the classification. In [30], issue descriptions and comments are explored to identify software feature requests of the users. This study is based on natural language processing

and machine learning algorithms. Bug classification based on the location of their fixes is presented in [2] and includes bugs fixed via modifying a single location in the code, those fixed in more than one location, multiple bugs fixed in the same location, or multiple bugs fixed in the same set of locations. Here, ITS issue reports are correlated with software version control issue reports. Issue classification facilitates an understanding of issues and is helpful in their handling [31]. ITS systems provide the capability to tag issues with default customized labels; however, many issues may not be labelled or may be tagged incorrectly. This can be supported with text mining classifiers focused on the issue title, description, and comments. In practice, the misclassification of issue types may impede an efficient bug-fix process.

Issue reports are enhanced with textual comments that comprise relevant questions or explanations. Relations between the comments and their communication functions are discussed in [32], e.g., referential, imperative, interrogative, expressive, or emotional sentences. Comments are accumulated into discussion threads, which are useful in the activities of project stakeholders. In [33], a supervised classification technique was used to distinguish 16 information types provided by the comments.

Some papers propose various prediction schemes. The model presented in [34] predicts questions raised by developers in an issue report. In [35], the correlation of software modules relevant to the resolution of an issue is based on a predictive model learned from past issue reports. Bug localization based on the textual similarity of the bug report description and the source code files is discussed in [36]. The proposed LaProb method [37] transforms the problem of bug localization into a multi-label distribution learning problem. It uses intra-relations and inter-relations among bug reports and source files (considering their structures and contents). Another model (fine-tuned Transformer) is proposed in [4] to predict both the objective of opening an issue and its priority level. A prediction of bug severity is discussed in [38]; it confirms the usefulness of unstructured text features combined with text mining methods. In practice, diverse imperfections appear in issue reporting; they mostly result from time pressure, the overloading of the project stakeholders, their negligence, etc. The deficiencies (called bad smells) of bug reports are outlined in [39], based on a survey of literature and questionnaire results from several companies, e.g., a lacking link to the bug-fixing commit, an incomplete resolution, a lacking bug assignee, severity level, or environmental information.

This survey confirmed a multitude of project development aspects which could be explored with software repositories. However, the methods proposed were restricted to selected data from software repositories targeted at analyzing specific problems in isolation from the others. Studies on issue inflow and processing were superficial and were restricted to basic statistics.

Dominant research problems are summarized in Table 1 with relevant references and basic data used in the studies. It is worth noting that issue attributes were limited to a few ones (mostly issue timing features, descriptions, and types). Extracted data from issue reports could be used in predicting or recommending software development practices, detecting duplicated issues, localizing bugs, and predicting issue fix time. A survey of issue attribute usage is presented in [40]. The most influential attributes were as follows: summary, component, priority, and assignee. The number of reported attributes can be quite large; in the survey paper [41], 25 attributes are listed in relevance to 16 Jira open-source projects. In our studies of diverse projects, we observed many more attributes. They could help in deriving additional statistics; nevertheless, there is a gap in their exploration as an additional source of information on project development and maintenance processes or actors' involvement. Hence, we dealt with this problem in subsequent sections, focusing on the quality of issue reporting and their information value/significance. The importance of assessing software quality is discussed in [42]; product/process factors such as code quality, issue velocity, pending issues, performance metrics, etc., are surveyed. The need for a detailed quality model is highlighted; however, issue reporting contents is not studied there. The analysis of issue repositories relevant to artificial intelligence (AI) systems is presented

in [43]. It focuses on some general issue properties, e.g., language of descriptions, text sentiment, the existence of issue assignee, code snippets, and the number of contributors. More detailed issue specification has been recommended. However, the authors have not analyzed issue attribute values and statistics.

Table 1. Literature taxonomy.

Research Problem	Literature References	Basic Data
Project lifecycle tracing	[1–3,12–17,42,43]	Issue inflow, timing features
Issue classification	[2,8–10,23–33,39]	Issue description
Issue duplication	[7,32,40,41]	Issue description
Task allocation, bug triaging, and localization	[6,18–22,36,37,40]	Issue description, actor skills, component, severity, priority, operating system attributes
Predictions	[4,5,11,12,24,29,34,35,38]	Issue type, handling times, code features

After analyzing a wide scope of repositories, we identified a great deal of reported details, which were neglected in the studies performed (Section 3). They provided deeper insight into issue processing schemes and appearing dependencies. Hence, the need for a holistic (fine-grained) exploration of software repositories aimed at enhancing systematic project assessment in various perspectives arose. We developed a multidimensional analysis of issue features based on the accuracy of their reporting and correlations (Sections 3 and 4). This facilitated the tracing and interpreting of issue handling schemes with diverse metrics useful in assessing development and maintenance processes, as well as the identification of their imperfections or anomalies. This gave rise to the problem of selecting repositories for studies. In our analysis, we checked diverse open-source and some commercial projects, which offered a wider view on repository contents, their quality, and limitations. For this purpose, a special tool was developed to extract data from repositories and create a uniform database for the analysis purposes specified in Section 1.

3. Software Repository Features and Research Methodology

Issue tracking systems (ITSs) are commonly used for document creation and the updating and resolving of reported issues by stakeholders of software projects. Project stakeholders, also called actors, are involved in project development, testing, and usage. They have the capabilities of accessing and filling ITS repositories according to assigned permissions allocated by project managers. There are many open-source and commercial ITSs, which differ in details of issue reporting (e.g., issue type, resolution category and severity, and activities of involved actors). The essential characteristics of ITS systems are outlined in Section 3.1; they formed the basis for specifying a generalized issue model and relevant investigation methodology (Section 3.2).

3.1. Outline of ITS Systems

Typically, ITS systems specify mandatory, recommended, and optional data fields (attributes) in issue reports and allow for the introducing of flexible projects or company-oriented data. This results in some diversity in the contents of a software repository. Nevertheless, some abstract or generalized data mappings can be introduced to facilitate project comparisons. Moreover, in long-time projects, we can observe some changes in data reporting style, used names/categories, etc. We illustrated this in relevance to four ITS repositories, referring to the most popular one—Jira.

A reported issue should be processed until a decisive resolution (e.g., completed, fixed, rejected as not being a problem, identified as duplicated, etc.) is reached. Issue handling may involve several phases (states), e.g., issue analysis, diagnosis, corrections, testing, validation, and completion. This processing progress is documented in the repository, and

it depends upon the type of issue, its localization, and other specifications. In general, we can distinguish 6 categories of data fields (attributes) in repositories corresponding to Jira:

Issue identifier: name of the project and unique issue key/tag.

Issue portrayal: issue title, summary, detailed description of the problem requiring a resolution, issue type, priority, and severity.

Issue processing progress: progress status (processing phase), final resolution (e.g., code fixing, rejecting duplicated or invalid issues), and generated comments (presenting an image of actor discourse during issue processing).

Actors correlated with the issue: reporter (person who created the issue), assignees (persons attributed to issue handling), and watchers (list of persons informed on the handling of progress).

Issue localization: affects versions (e.g., the code version number of issue appearance), fix versions (application versions with the resolved issue), components (software components related to the reported issue), environment (e.g., the operating system, used libraries), and links to the issue (e.g., URLs).

Basic timestamps: the creation, update, resolution dates, and due date (date of expected resolution).

Supplementary information: estimated resolution effort (e.g., man hours), total issue handling time, and appendices complementing problem descriptions (e.g., screenshots, snippets of the suspected code, and test suites activating detected defect)—they can be combined with a description or portrayal of comments in the issue.

In Bugzilla (limited to tracking bugs), the following additional interesting data fields appear: *blocks* (links between mutually blocking tasks), *depends on* (list of issues which should be resolved before the one considered), *product* (issue classification for further filtering), *see also* (references to linked issues), and *votes* (number of user votes supporting the resolution of an issue). In Mantis Bug Tracker, the *reproducibility* field defines the level of defect repetition. In commercial project repositories, other specific fields appear, e.g., in projects considered in [3], over 100 fields were used. This included information related to testing (test source, category, group, and path), which, in another project [44], were stored in a special test repository. On the other hand, many fields were rarely used (filled sporadically or assuming a single value). The scope of comprised data in ITS repositories is potentially quite large, and it is reasonable to check its usefulness in a holistic and deeper project assessment. Hence, we initiated our studies to obtain better insight into these data and explore their significance, as well as to identify reporting deficiencies.

A preliminary survey of software project repositories proved that they comprised a lot of diverse data, which were used in research and development practices in a very restrictive way. Most of them were neglected without checking their potential significance in a holistic assessment of projects. Hence, there is a need for a deep and systematic study of these repositories, while considering various observation perspectives and introducing some classification taxonomies.

The syntactic and semantic properties of data fields are specified with different precisions sometimes freely interpreted by actors; moreover, some fluctuations may appear in time in the case of long-term projects, fluctuations of employed actors, etc. Sometimes, issue categories are changed during their processing, e.g., type, priority, or severity. Moreover, project actors may not be sufficiently motivated to update their activity in an ITS repository by filling all possible data fields. Included texts can use various jargons, mixtures of words from several languages, variances in spelling, and differing grammatical flows. The exploration and interpretation of repository imperfections and inconsistencies requires more effort. Here, we also faced the problem of evaluating the value of comprised data, their redundancy and orthogonality, consistency of reports, etc.

An important issue is to derive characteristic features that are helpful in the evaluation of project development. Some of them can be extracted directly from appropriate data fields in an ITS repository; others need the tracing of data correlations involving several fields or even other repositories, e.g., software version control. When confronting these problems,

we proposed an original taxonomy of issue attributes, supported with assessment metrics and feature profiles related to value and timing characteristics. These are derived either directly from attribute content or involve cross-sectional correlations (e.g., value and timing aspects).

3.2. Methodology of Issue Exploration

Our analysis of issue features is based on the following issue model relevant to the considered software project P:

$$IS(P) = \{I_i \mid R\};$$

IS(P) is the set of all issues related to project P satisfying restrictions R. Restrictions can specify the features of considered issues, e.g., time perspective of the project, issue type or priority, or issues correlated with reporter classes. Each issue is characterized by the relevant set of attributes:

$$I_i = \{a_{i1}, a_{i2}, \ldots\ a_{ij}, ., a_{in(i))}\}$$

where $I_i \subset IS(P)$, a_{ij} is the j-th attribute value of the i-th issue, n(i) is the number of attributes for the i-th attribute. To facilitate project comparisons, we can perform a mapping of attribute values into compatible ones as well as aggregate them. Some complex attributes can be modelled as lists, e.g.:

$$L(a_i) = <a_{i1},/s_{i1}, a_{ij},/s_{ij}, \ldots\ a_{in(i)}/s_{in(i)}>$$

where a_{ij} is the attribute value and s_{ij} is its context specification, e.g., a timestamp, or a reporting person. They mostly relate to the history of attributes and can be stored as additional data in the repository, while the main attribute field may comprise only the recent attribute value.

The issue analysis methodology developed for this study is composed of four steps:

(1) Issue acquisition from external ITS repositories;
(2) Data preprocessing and structuring;
(3) Issue feature extraction and quantitative/qualitative assessment;
(4) Result visualization.

Data was acquired using the appropriate REST APIs of the ITSs. They usually provide access to a limited number of issues (e.g., 1000), so they were extended with a view of completing a specified number of issues (or relevant observation period) and storing them in a uniform structure in a separate database (MongoDB) for each project. To download more issues, we generated subsequent data packet requests with the specified number of issues in the packet and a time offset (related to the initial timestamp of the first acquired packet). Using multithreading (based on the *Concurrent.Futures* module and *Process Pool Executor* class from the Python library), we can simultaneously acquire data from several projects to speed this step up. The specifications of many fields (attributes) may have comprised surplus information irrelevant to the performed analysis; therefore, appropriate data cleaning (purification) was included. The data structures of some attributes were complex and nested. The simplest ones comprised single values (e.g., issue id and reporting date), while others might have comprised objects or lists of objects (e.g., comments attribute). Hence, it was important to simplify these structures by deriving interesting data and storing them in a structural format, e.g., identifiers of commenting authors, time of posting the comments, etc. This was realized with a *JSON* packet (https://docs.python.org/3/library/json.html, accessed on 15 August 2023); moreover, due to diverse specifications of time (e.g., dates) we used a generic date/time string parser (https://dateutil.readthedocs.io/en/stable/parser.html, accessed on 15 August 2023). The historical values of attributes are available via additional data requests with appropriate specifications. In Jira, the history of all attributes is provided in a single file, so extracting the interesting ones needs appropriate parsing.

Extracted data from raw record issues were filled in appropriate collections of MongoDB (operation on port 21414). We distinguished three main collections: (i) *issue_synt*—which comprised the issue id, the date of reporting, and the reporter's name; (ii) *issue_full*—which comprised the values of all the attributes of the issue relevant to the date of data acquisition, the number of relevant comments, and the length of the issue description; and (iii) *comments*—which comprised the comment text, the relevant issue identifier, the comment identifier, the comment author, the comment length, and the comment creation date/time. Additional collections were related to extracted historical data, e.g., subsequent processing state data (date/times and the involved actor). After all the required data was stored, appropriate indices were fixed for each collected component (element) to speed up inquiry handling.

MongoDB, used here, is a popular document database, which offers an intuitive and flexible data model, adaptable to potential extensions or changes. A document may comprise a variety of values of types and structures, including strings, numbers, dates, arrays, or objects. Documents can be stored in formats like JSON, BSON, and XML, which are used in repositories and facilitate data processing, including machine learning algorithms. Data processing involves deriving diverse statistics on, the visualization of, and presentation of results. MongoDB was also used in other papers on issue tracking, e.g., [41].

For data processing, the following supplementary data collections were created in the database: (i) *comments_stats*—comment statistics comprising issue id, the number of comments, the number of unique commenters, and commenting timing features; and (ii) *issue_timing*—the date/time of final issue resolution and the number of days devoted to issue resolution. Issue attributes may assume specific category values (e.g., issue type, priority), lists of historical values with relevant timestamps (e.g., subsequent processing states), and textual contents (e.g., issue descriptions, comments). Hence, their analysis had to be adapted to relevant categories and needs through introducing appropriate evaluation metrics and statistical profiles. We have defined four classes of them: general characteristics (e.g., filling ratio and scope of assumed values); issue value distributions; features focused on specific problems (e.g., project actors' activities in issue contribution and time involvement, features of comments, etc.); and issue dependencies (correlations in time, value, etc.). The results were presented in individual or aggregated perspectives, depending on the analysis of the considered problem. For this purpose, diverse profile vectors (including vectors of vectors), multiparameter tables, and timing plots were used. They are specified in subsequent sections and illustrated with results relevant to illustrative projects. We analyzed issue handling processing (timing issues and processing states) using our previously developed PHG model and the tool described in [15]. Textual attributes were analyzed with text mining algorithms; we confronted this problem in Section 4.1.

After all the summarizing of the processing was completed, statistics were derived, e.g., to compare diverse projects or to assess selected projects. The analysis that was developed was focused on the following aspects:

- Attribute filling ratio and the scope of assumed values;
- Distributions of assumed values in diverse perspectives;
- Correlations considering diverse dependencies;
- Project actors' activities (in reporting contribution and time);
- Analysis of textual attributes;
- Timing features (issue inflow considering types, priorities, resolution time, etc.).

The results were stored in .csv files and could be visualized in graphical plots and tables. For this purpose, we used data analysis and the *pandas* manipulation tool (https://pandas.pydata.org/, accessed on 15 August 2023), as well as the *Matplotlib* library (https://matplotlib.org/, accessed on 15 August 2023). Visualization could be performed in two perspectives: a single project one and multi-project one (cross project analysis). Data processing was performed separately for each project; however, some cross project analysis could also be performed.

4. Exploring Issue Repository Content

Issue report attributes assumed diverse value types: numerical, category, or textual. Numerical values could be viewed in the form of an integer, real numbers, formats of dates or time, a program version, etc. Their range, granularity, and ordering depended upon their purpose and were useful for correlating other features with the project phase, code locations, etc. Category values were characterized by a specified list of acceptable names (e.g., priority), usually limited to several or a few dozen elements; sometimes default values were defined here. They specified detailed features of issues and are analyzed in Section 4.1. Textual attributes comprised texts composed of natural language words and other textual objects related to code snippets, e-mail addresses, technical names, acronyms, etc. They could be characterized by text size (number of words, characters), as well as syntactic and semantic features. They are discussed in Section 4.2.

4.1. Category Attributes

The analysis of category attributes focused on the following problems:

(i) The identification of used attributes and their filling ratio in the project lifecycle;
(ii) The distribution of used attribute values, checking their consistency over reporters and time;
(iii) Deriving attribute value profiles conditioned by issue types, projects, reporter classes (developer, user), issue severity level, etc.

This involved tracing all reported issues to extract general parameters for subsequent statistical analysis. To facilitate project comparison, attribute values were grouped in consistent classes (e.g., priority classes) and aggregated in semantic groups (e.g., issue resolution). Such an analysis provided better insight into repository data and facilitated the detection of anomalies.

For most of the open access projects, the number of attributes used was in the range of 15–25; however, about 10 attributes were typically filled in at more than 90%. Most repository recommended attributes were commonly used; nevertheless, some specific ones appeared, albeit scarcely. In commercial projects, a much higher diversity could be observed. For example, in the projects of disc controllers [3], over 100 attributes were used: a number of 22 with an over 90% filling ratio and 55 with a more than 10% filling ratio. They included such useful attributes as: *suspected area*, *source_of_error* (requirement, implementation, environment), *error_characteristic* (incorrect, incomplete, missing), *acceptance criteria*, *test-source*, *test category*, *test_group*, and *test_path*. On the other hand, in another commercial project (C1), 25 attributes were used, including the following not encountered in other projects: *attachments* (comprising screenshots, application logs, project documentation, schemes of API services, etc.), *participants* (list of persons involved in issue pre-processing), and *watchers* (list of persons notified by e-mail during issue changes).

The primary (general overall) profile of the repository was characterized by two parameters: filling ratio and the cardinality of used values in category fields. Some illustrations are provided in Table 2 for commercial (C1) and five open-source projects. Here, we considered the consistent fields of these projects and those which have a filling ratio below 100%. The filling ratio was followed (after/character) by the number (cardinality) of the attribute categories (values). The attributes filled in at 99.5–100% were skipped; for most of the projects, issue id, type, priority, status, summary, reporter, created, etc. were included. In the case of some projects, obligatory attributes showed filling deficiencies, e.g., a description below 100% (89.8–99%); however, this could be compensated by a 100% summary and title filling. The deficiency in priority filling was more important: 92.6%, 99.3%, 2.6%, and 40.1% for Flink, Spark, Mozilla Thunderbird, and Red Hat projects, respectively. The type attribute assured a filling ratio of 100% except for Red Hat—80.0%. In all projects except for Mozilla and RedHat, the *resolution* attribute was complemented with a *resolved* one, which specified the timestamp of the resolution (it comprised about 85% of unique timestamps). In Jira, bug issues comprised an additional field called *environment*

(e.g., operation system version), and the *user story* attribute comprised the number of user story points and a sub-task list.

Table 2. Attribute specification features: filling ratio (%)/the number of category values.

Attributes	C1(17,658)	Flink (28,714)	Ignite (17,396)	Spark (39,500)	Moz (62,207)	Red (22,066)
Resolution	86.5%/19	83.0%/21	74.2%/19	92.7%/19	86.6%/9	88.4%/13
Fix Vers.	65.1%/195	62.1%/388	61.1%/68	59.5%/324	-	43.3%/6605
Aff. Vers.	-	51.0%/836	42.3%/168	76.3%/1176	100%/108	100%/11
Compon.	50.5%/450	89.5%/909	63.0%/261	95.5%/620	100%/26	100%/1387
Environ.	15.4%/2252	3.4%/873	2.2%/355	7.1%/324	-	0.05%/12

We could observe some differences which could be drilled down by deriving conditional statistics (restriction R in Section 3.2) related to specified issue types, observation periods, etc. This aspect is discussed later. Most attributes (fields) were widely accepted in diverse projects, and they could assume different (compatible) values. Nevertheless, some unique fields only appeared in specific projects, e.g., the *Change* attribute in the Cassandra project.

Another important repository aspect was the distribution of attribute values (category names) used in the repositories. We illustrated this for the primary fields in Table 3, and we limited it to the most important values (e.g., appearing sufficiently frequently), such as: issue priority, type, status, and resolution. Some values were specified differently in projects that were considered. For example, priority values in Cassandra and Red Hat were specified as normal, low, urgent, and high; in Mozilla there was P1–P5, with P1 being the most critical (with a skipped P5 = 29.5%). Their ordering was adjusted to the compatible values in the table (Major, Minor, Critical, and Blocker). Other values not shown contributed 100-S%, where S denoted the sum of the percent included in the table. Moreover, some values representing similar meanings were grouped (aggregated); we specified the cardinality of such groups in brackets. For example, *Resolved* comprises {*Fixed*, *Implemented*, *Done*, *Resolved*, and *Workaround*}; *Won't Fix* comprises {*Abandoned*, *Auto Closed*, *Invalid*, *Not a Bug*, *Not a problem*, *Won't do*, *Won't Fix*, and *Works for me*}; and the remaining values include *Later* (typically 0.04–1%), *cannot reproduce*, and *Feedback received*. Their meaning is self-explanatory; however, some ambiguities exist.

Table 3. Distribution of attribute values (%).

Attributes		Cassandra	Flink	Ignite	Spark	Mozilla	Red Hat
Priority	Major	59.1	66.8	75.8	63.7	21	45.2
	Minor	37.1	18.2	10.5	24.9	3.34	13.9
	Critical	3.6	7.9	7.9	4.0	19.7	33.1
	Blocker	0.2	7.1	4.3	3.5	26.7	7.8
Resolut.	Resolved	68.8 (5)	76.3 (5)	81.2 (5)	66.5 (5)	23.6	64.5 (4)
	Won't Fix	15.9 (8)	13.4 (8)	11.0 (7)	15.7 (8)	43.3 (5)	24.5 (5)
	Duplicate	10.3	7.5	6.1	6.7	32.9	8.1
Type	Bug	55.2	38.9	41.7	39.5	82.8	96.3
	Improv	31.5	31.4	23.1	30.2	14.6	1.0
	Task	7.8	21.9	28.5	18.7	2.6	-
	New Feat.	5.5	6.4	6.4	5.2	-	2.2
Status	Resolved	86.5	15.2	46.2	85.5	88.3 (2/)	-
	Closed	-	67.8	27.9	7.3	-	88.5
	Re(open)	10.5	16.1(2)	4.5	4.5 (2)	7.1 (2)	5.5

The distribution of assumed values within the field categories that were analyzed may significantly differ, depending on the specificity of the project, their development schemes, assumed policies in the company, competence of actors, etc. Hence, deeper insight into some distributions could be reasonable, e.g., issue priority. In the case of the MongoDB project, the distribution of priority issues was as follows: Major (P3)—94%, Critical (P2)—2.3%, Minor (P4)—2.3%, Blocking (P1)—1.1%, and Trivial (P5)—0.4%. It seemed that the criteria were not properly assigned, so most issues were qualified as *Major* (probably the default value), or the actors were not motivated to waste time on its evaluation. This was often encountered in open-source projects. In commercial projects, this was usually more restrictive. For example, in C1 project we obtained the following priority distribution (Low, Medium, High, Urgent and Cosmetic): (41.9%, 34.4%, 19.0%, 2.4%, 2.2%) and (45%, 28.5%, 21.5%, 1.9%, 2.4%) for all issues (584) and bugs only (418) within a period of 3 sprints. For subsequent sprints, some fluctuations were observed; however, the general relation was similar.

There were also some other specific fields that were useful, depending on the projects, e.g., *severity* (medium, high, low, urgent) for Mozilla (73.5%, 9.6%, 9.2%, 7.6%) and Red Hat (47.8%, 27.1%, 17.8%, 7.4%), *change category* for Cassandra (operability—36.7%, quality—24.6%, semantics—20.7%, code clarity—9.9%, and performance—7.8%), and *discovered by* (user 49.8%, code tests 39.1%, and code inspection 10.3%).

Values within some issue report fields should not change during the issue handling period, e.g., id, priority, and issue type. The appearance of such changes needs identification and explanation. Usually, the attribute values used (the list of possible names in the issue report field considered) in the whole project repository were stable. Nevertheless, we observed some changes (updates) during the project lifetime. It was important to identify the corresponding time of these changes and consider them in derived statistics as well as explain their reason. This could be caused by a decision of the project manager, using new version of ITS system, reporter experience moved from another project, etc.

The reported issues were related to diverse types (e.g., bug, improvement); their handling could involve different project actors and relevant costs. Hence, it was reasonable to trace issue type profiles in projects, e.g., specified as a vector with issue ratios across their types (ITV). As an illustration, we provided such vectors as a sample of open-source projects:

$$\begin{gathered}\text{ITV(Cassandra)} = (0.55, 0.32, 0.13);\ \text{ITV (Flink)} = (0.39,0.31, 0.30);\\ \text{ITV(RedHat)} = 0.77, 0.03, 0.20);\ \text{ITV(Spark)} = (0.40, 0.30, 0.30);\\ \text{ITV(Mozilla)} = (0.83, 0.16, 0.03);\ \text{ITV(Ignite)} = (0.42, 0.23, 0.35).\end{gathered}$$

Subsequent vector elements specified the ratio of bugs, improvement, and other issue types. Depending on the project, we could observe some significant differences, e.g., high ratio of bugs and low ratio of improvements. The distributions presented here covered the whole observation period. They could have changed in time, which could have been presented in timing plots. In the projects considered, the profiles presented fluctuated insignificantly; however, in the case of the Ignite project, other issues dominated within the first 4 years, achieving a 0.45 ratio (see Section 5). For some other open-source projects with issue types such as bugs, a new feature, or improvement, we had:

$$\begin{gathered}\text{ITV(Groovy)} = (0.75, 0.05, 0.20);\ \text{ITV(Lucene)} = (0.46, 0.47, 0.07);\ \text{ITV(log4J1)} = (0.61, 0.10,\\ 0.29);\ \text{ITV(Arrow)} = (0.38, 0.12, 0.49);\ \text{ITV(MongoDB)} = (0.65, 0.10, 0.32).\end{gathered}$$

For the considered C1 commercial project we had:

$$\text{ITV(C1)} = (0.65, 0.10, 0.10, 0.14)$$

where the specified reported issue ratios were related to bugs, new features, user stories, and tasks. Knowledge of the issue type distribution (profile vectors) facilitated optimizing project actor profiles (Section 4.2).

Issue attributes were specified with different amounts of attention, depending on the project team and assumed development practices. This could be evaluated by tracing the filling ratio and the distribution of assigned values. Some attributes were considered mandatory and filled in at 100% (however, negligence up to a few percent sometimes appeared). Depending on the projects, many others demonstrated different levels of negligence or were not used. This is illustrated in Table 2. The values presented were related to all reported issues. We also analyzed the filling ratio in relevance to issue types; some illustrative results are given in Table 4 for four attributes: comment, resolution type, component, and priority. The presented vectors (profiles) separately showed the filling ratio of all issues, bugs, enhancements, and other type issues. In some cases, noticeable differences were visible, e.g., in the case of a resolution attribute. Low values for the component attribute in Cassandra showed problems with diagnostics; low values for priority in the case of Mozilla and Red Hat were worrying (due to the irresponsibility of reporters or imprecise recommendations of project manager). However, the severity attribute was also used in these projects, which could be considered to be complimentary to priority; it had the higher filling ratios (85, 90, 86, 51) and (52, 58, 74, 26) for the Mozilla and Red Hat projects, respectively. Nevertheless, these values were still not satisfactory.

Table 4. Attribute filling ratio (%) profiles referred to issue types (all, bugs, enhancements, and others).

Project	Comments	Resolution	Component	Priority
Cassandra	(93, 95, 90, 90)	(86, 91,81, 81)	(50, 49, 49, 59)	(100, 100, 100, 100)
Flink	(94, 93, 91, 93)	(87, 83, 82, 80)	(86, 90, 95, 89)	(93, 94, 93, 96)
Ignite	(74, 79, 69, 73)	(74, 75, 66, 78)	(63, 62, 60, 60)	(100, 100, 100, 100)
Spark	(93, 93, 90, 90)	(93, 95, 90, 93)	(97, 95, 97, 98)	(100, 100, 100, 100)
Mozilla	(97, 98, 93, 97)	(67, 89, 71, 92)	(100, 100, 100, 100)	(2.6, 2.4, 2.5, 8.8)
Red Hat	(81, 85, 89, 66)	(88, 88, 85, 92)	(100, 100, 100, 100)	(41, 40, 60, 39)

Depending on the projects, the filling ratio of attributes differed; however, for some of them, this fluctuation was relatively low, e.g., the resolution attribute for all issues was filled in the range 74–93%, 75–95% for bugs, and 66–85% for enhancement issues. Higher fluctuation was observed for the component attribute (50–100%), and the Fix version attribute (not shown in the paper) was filled in at 60–73%. In some cases, significant low values appeared, e.g., the component attribute for the MongoDB project was filled in at 35%.

Beyond typical attributes, we encountered additional customized fields specific to individual projects. Their numbers were quite big; for example, in the MongoDB project, we had 52 additional attributes, with 13 of them filled in at 100% in an automatic way (reporting actors' interactions with the repository) and 6 not used. The remaining ones were filled in the range of 0.01–67%. The filling ratio differed on the issue type, with the ratio of the maximum/minimum values of 2/6, while some were used only for a specified issue type (e.g., C10032 in 96% for bugs). In the case of the C2 commercial project (a newer version of C1), we identified 39 additional attributes with 9 of them filled in at 100%, and 6 not used. Depending on the issue (new feature, user story, task, or bug), we identified 26, 34, 20, and 23 additional attributes, respectively. Most of them were common to all types. Their filling ratio ranged from 0.05% to 20% in total; however, for some issue types, maximum values were in the range of 70–85%. They enhanced information on bug localization, assessment of the complexity, and were related to business feature names. The number of possible values varied from 2 to over 20 with different distributions. However, quite often, default values dominated due to some inattentiveness of the actors. Deeper analysis could lead to the cancelling of such attributes. For example, in commercial projects [3] we recommended reducing the list of used attributes by 60%.

4.2. Textual Attributes

Textual fields (or attachments) comprise various texts (usually unique) which may differ significantly, so considering value distribution is irrelevant. Here, it was more reasonable to use other metrics, e.g., distribution of the number of comprised bytes or diverse words (natural, technical, or symbolic language, etc.). Syntactic features were related to natural language words used, other objects, and linguistic deficiencies (incorrect words, grammar flaws). Semantic features could be derived using advanced text mining techniques; this problem was discussed in many papers, including our previous one [45]. Here, we restricted our studies to the following problems:

(i) The statistical assessment of text sizes;
(ii) The analysis of lexicon used in relation to natural language words (NL) and non-NL textual objects (e.g., code snippets, program class names).

Statistics on text sizes, natural words used, and other textual objects offered some view on information value and possible text mining processes. To illustrate that, in Tables 5 and 6, we provide text size statistics (Q1, Q2, and Q3 quartiles in words/tokens) for issue descriptions and comments related to 6 open-source projects.

Table 5. Statistics of issue description text sizes for open-source projects.

Parameter	Cassandra	Spark	Flink	Ignite	Mozilla	Red
Q1	32	18	20	16	54	37
Q2	64	44	42	35	90	92
Q3	124	94	84	72	141	198

Table 6. Statistics of comment text sizes for open-source projects.

Parameter	Cassandra	Spark	Flink	Ignite	Mozilla	Red
Q1	10	9	11	7	13	13
Q2	26	10	37	12	26	39
Q3	61	32	86	35	58	64

Some issue description sizes significantly differed from median (Q2) values; the minimum values were 0 and the maximum ones ranged from 5684 (Mozilla) to 79,479 (Spark), with average values of 84–191. It is worth noting that very long texts comprised code snippets, event logs, etc. Those without these objects were in the range of 300–900 words. On the other hand, short texts comprised jargon or informal phrases. They could express approval, support, negation, performed action, etc. Here are some examples: +1—approval, LGTM (acronym for looks good for me), tested, fixed, committed, working on this, and FYI (for your information). Null texts resulted from some negligence or a context with other attributes. Comment texts had similar properties; however, lower text sizes were noticed here (Q1–Q3 quartiles). The minimum values were 0 and the maximum ones ranged from 4834 (Mozilla) to 107,000 (Flink), with average values of 32–98.

Deeper analysis of words and other textual objects used offers a wider view on the textual context. Namely, the vocabulary of natural language words was relatively limited up to several thousand; non-NL words were also limited. We illustrated this for the MongoDB project in Figure 1. The lower plot showed the number of used unique NL words in issue descriptions covering subsequent months (typically in the range 2000–3000). The lexicon used changed in time, hence the number of unique NL words in a longer timing perspective systematically increased; however, it demonstrated a saturation trend. In 2 years, it increased to almost 9000. In the case of the C1 commercial project, the size of lexicon used was lower (4000 NL words in 2 years). We observed similar properties in other projects. Hence, text mining of issue descriptions and comments could not be

directly based on classical techniques used in text analysis targeted at publications, web recommendations, etc. Deriving semantic significance from these texts needed a special adaptation. We proposed an original approach in [45], which considered specific features of these texts, namely non-NL objects. Moreover, they could be enhanced by correlating them with other attributes.

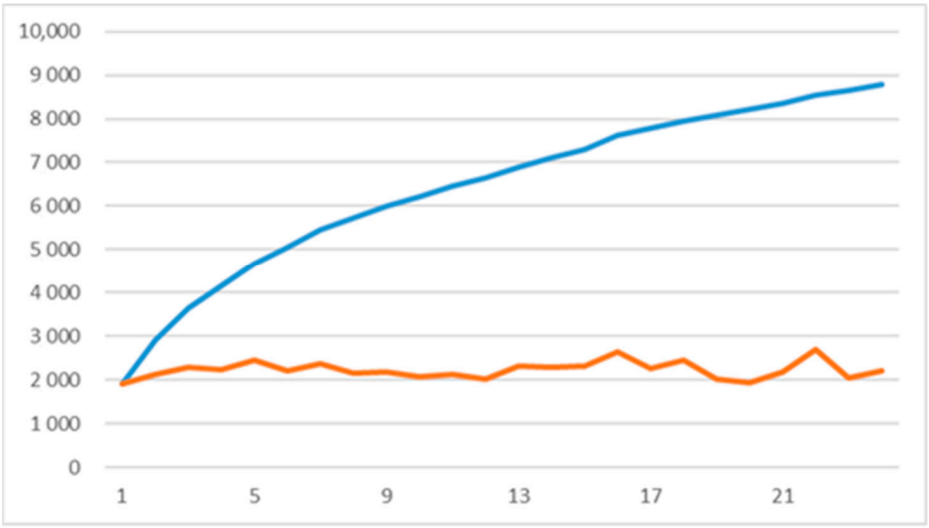

Figure 1. NL lexicon size (*y*-axis—the number of unique words) used in MongoDB issue descriptions: monthly (**lower** plot) and accumulated (**upper** plot) statistics (*x*-axis—subsequent months).

A clustering of issues based on textual properties could facilitate their investigation. We could distinguish the following: short descriptions, descriptions comprising non-NL objects, and remaining ones. Short descriptions were relatively simple for classification, all the more that used phrases are usually significantly limited in size and vocabulary and could be checked manually. Similarly, descriptions with non-NL objects could be classified by taking into account their context; the classification of the remaining descriptions was more complex. But considering their relatively short size and limited vocabulary, we could identify characteristic keyword phrases. This could be combined with machine learning schemes—a hybrid approach. Moreover, such morphological analysis allowed us to assess the quality of issue/comment reporting and propose appropriate improvements for identified deficiencies, e.g., incorrect words, unclear abbreviations, and a low percentage of technical objects. They could be further correlated with project actors or issue types. Non-NL words (text objects) also provided important information, which is not considered in classical text mining approaches. To illustrate that, in Table 7, we provide some statistics of such objects in issue descriptions of MongoDB and C1 projects. They were related to e-mail addresses (E-mail), code snippets (CS), program class/packet names (CN), binary (BA) and image (IA) appendices, links (L), change code references (CR), and Jira panels (JP). Most of them constituted a significant fraction of the lexicon used (except from IA, PR, and JP). Such objects appeared in other repositories; however, their distribution differed. Hence, it was reasonable to consider them in text mining algorithms. We developed such algorithms for issue and comment classification, which was presented in [45].

Table 7. Distribution of non-NL textual objects.

	E-mail	**CS**	**CN**	**BA**	**IA**	**L**	**PR**	**JP**
MongoDB	1%	28%	51%	5%	2%	9%	3%	1%
C1	51%	1%	14%	22%	0%	12%	0%	0%

5. The Activities of Project Actors

Some issue attributes could draw special interest for the repository and require deeper studying. This holds for the analysis of project actors' (stakeholders') activity. General project statistics analyzed the number of actors and the distribution of reported issues by these actors. We distinguished two groups of project actors: those generating issues (issue reporters) and those discussing their resolution in comment exchanges. Activities in these aspects are studied in Sections 5.1 and 5.2, respectively.

5.1. Reporters' Activity

The analysis of reporters' activity was focused on two observation perspectives:

(1) Issue reporting contribution profiles in aggregated or personal views;
(2) The time involvement of issue reporters.

The first point required deriving a list of active reporters (based on reporter attribute values) and summarizing relevant issues. The second point required tracing reporters in correlation with issue timestamps (explained in continuation) and the aggregation of relevant activity time periods. Deeper statistics involved cross-sectional profiles correlated with the issue type or reporter category. We illustrated this in derived actor profiles for a sample of projects.

The number of issue reporters and their contributions differed in developed projects. It was typical that a small group of actors had a dominating contribution. Hence, it was worth introducing aggregated reporting profiles, showing the number of reporters contributing a specified percentage of issues, e.g., 50%, 80%, and 100%. These were so-called aggregated reporting profiles (over the whole project time). Quite often we could observe a high fluctuation of actors in time (e.g., project phases). Hence, it was also reasonable to explore the reporting profiles conditioned by observation periods. This was illustrated in Table 8, which showed the number of reporters that contributed 50%, 80% and 100% of issues within 13 years (year-long report profiles) for two OS projects. The last row provides aggregated statistics covering the entirety of the period considered. The number of contributing and most active reporters increased in subsequent years (code bugs and new functionalities increased as well) and, after achieving a maximum, it slowly decreased (matured code, however, which involved longer product usage, generated more issues). The long time distribution of the number of reporters contributing 50, 80 and 100% of the issues for other projects was as follows: Flink (30, 176, 2619), Ignite (24, 75, 775), Mozilla (2043, 15,632, 28,073), and Red Hat (154, 735, 3252); they covered 8, 8, 22.6, and 19 years, respectively.

Table 8. Excerpt of reporter contribution profiles.

Project Period	Cassandra			Spark		
	50%	80%	100%	50%	80%	100%
1	4	16	99	1	2	4
2	9	51	245	2	7	18
5	29	151	336	24	175	911
6	29	174	533	25	282	1473
12	11	57	198	20	145	828
13	10	37	176	15	82	295
Total	51	429	2917	64	940	6752

We could also generate statistics showing the contribution of individual reporters or specified groups. We illustrated this with derived activity profiles. The most active 10 reporters provided 23.2, 20.2, 30.0, 32.1, 10.9, and 21.4% of issues for Cassandra, Spark, Flink, Ignite, Mozilla and Red Hat, respectively. Here, the contributions of individual reporters were in the following ranges: 1.2–6.4, 1.5–3.2, 1.6–5.8, 1.6–7.6, 0.6–2.2, and 0.7–5.6%, respectively. In

commercial projects [3], the number of reporters ranged from 200 to over 500 (high fluctuation), 4–12% of them contributed 50% issues, and 12–30% contributed 80% issues. A range of 50–60% of reporters produced no more than 5 issues, and they contributed 5–7% of all reported issues. However, the maximum number of issues per reporter (the most active one) obtained values of 300–1000. In some projects, the dominant reporters showed higher stability in time, e.g., in MongoDB, a group of the 15 most active reporters was stable at 50% for an over one-year period. Here it is worth mentioning the automatic issue reporting by the *Coverity collector user* service. It analyzed code control and data flow by following possible code paths that the program could take, and detected such issues as memory leaks, null dereferencing, dead code, buffer overflows, and uninitialized variables. In the C1 Scrum project, the reporter profile was more balanced due to low staff fluctuation (presented in continuation).

A different metric of reporter activity was the time of involvement in the project; it was especially important in long-term projects and in the high fluctuation of project actors. This time could be specified in two ways:

(i) Absolute—taking into consideration the first and last timestamps correlated with the reporter interaction that was considered, together with the ITS repository (within all registered issues);
(ii) Aggregated—summarized periods of activity in which subsequent actors' interactions with ITS did not differ by more than a specified delay DT.

The distribution of reporters' activity times for six projects is given in Table 9. Here, we present median (Q2), Q3 quartile, maximum and average values. When calculating activity periods, we counted the number of days within the time periods T_i, specified by the first and last issue announced by the reporter. In the case of successive reports that appeared after a period of inactivity that are higher than a specified DT period, we started a new T_{i+1} period. In Table 9, we provide the values calculated for DT that were equal to 4 and 52 weeks and denoted with the parameter suffixes -a and -b, respectively. The minimum values, Q1, and dominant ones were assumed as 1.0 for all projects; this resulted from many reporters (users) providing a single issue. We could also derive normalized reporting profiles in reference to activity periods. Here, reporter contribution was equal to the number of the reported issues divided by the number of days within the reporter activity period. This was a metric of the individual reporting efficiency for an issue and its distribution.

Table 9. Distribution of reporters' activities in time (days) for DT = 4 weeks (suffix—a), and DT = 52 weeks (suffix—b).

Parameter	Cassandra	Spark	Flink	Ignite	Mozilla	Red Hat
Med—a	2.0	2.0	3.0	2.0	1.0	4.0
Med—b	3.0	2.0	4.0	4.0	1.0	29.0
Q3—a	15.0	9.0	22.0	28.0	2.0	34.0
Q3—b	85.0	60.0	70.0	161.0	3.0	410.5
Max—a	3501	3094	2809	2399	6451	1871
Max—b	4474	3605	2937	2646	8229	3842
Av—a	45.7	26.5	47.9	114.5	11.4	62.8
Av—b	142.2	104.4	137.9	209.3	76.9	302.1

We could also derive actors' activity plots in time; they could be processed using time series analysis algorithms to derive characteristic features (e.g., high and low activity period distribution) and correlate them with issue handling features. The number of unique reporters within each month was relatively stable: for Cassandra—30 in the initial phase, 90 in the middle, and 20 at the end; for Flink, it increased from 10 to 150. Similar values were observed for other projects; however, for Red Hat it was over 250. This resulted in

the accumulative plots of S and exponential shapes. They showed the number of reporters contributing to the project up to the moment on the time axis (resulting in big numbers at the end of the considered period). For Cassandra and Red Hat, it exceeded 2500; for Ignite, 800; and for Mozilla, over 25,000 (due to a very long project duration). These figures showed the high fluctuation of issue reporters. This was also observed in some commercial projects. The speed of these fluctuations was in the range of 100–500 new reporters per year and 1000 for Mozilla. High fluctuation had a significant impact on various prediction accuracies (e.g., in SRGM modelling). Unfortunately, this was neglected in publications that were based on coarse-grained analysis. Activity plots for most active reporters (providing many issues) typically showed a highly fluctuated number of created issues per month within limited periods of up to several months. Nevertheless, some people sporadically reported issues in a longer period.

A more detailed analysis should distinguish the roles (responsibilities) of reporters and commentators or their competence. Furthermore, we could consider their activities in relevance to issue types. Usually, such data were available in commercial projects; more ambiguities appeared in open-source projects. Therefore, we could use text mining techniques to derive such information. To illustrate that, we assembled the following five groups of reporters in a C1 commercial project (their cardinality is given in the brackets): Testers (6), Business Analysts (5), the Product owner (1), Developers (1 of 15), and UX Designers (1 of 5), which published 364 bugs, 164 issues (new features, user stories, tasks, and rarely bugs), 22 bugs, 5, and 5 bugs, respectively (in three subsequent sprints). The statistics covering three subsequent sprints of the project were as follows: most reports were provided by Testers (16–111) and Business Analysts (16–54). The reporters were involved not only in announcing issues, but also in commenting on them, resolving and analyzing them, etc. Their activities in relation to these roles requires deeper analysis. The diversity of reporter responsibilities, their activity profiles, and types of issues revealed the problem of prediction accuracy, which is not considered in the literature (the homogeneous distribution of these features is considered by default).

The general reporters' activity profiles that were presented (Tables 8 and 9) did not distinguish between their responsibilities/competence in the project. Deeper insight may be needed to trace their activities, taking into account their roles in the projects. To illustrate that, we provided such a profile (RAP) for a C1 project, showing the percentage of reported issues by five groups of actors: Testers, Programmers (Developers), Architects, Analysts, the product owner, and the User interface designer. They covered two recent years (2378 issues):

$$\text{RAP(C1)} = (57.7\%/9,\ 2.2\%/4,\ 2.3\%/2,\ 36\%/5,\ 1.5\%/1,\ 0.3\%/1)$$

The number of involved actors is given after/character. The contributions of subsequent actors within the considered groups showed some dominating ones and less active ones. The dominant group was related to Testers and Analysts; the remaining ones contributed in a range of 1.5–2.3% (in total 5.8%). However, their contribution differed in relevance to issue types. This is illustrated by profile vectors specifying the percentage of reported issues within 4 categories: New Feature, User story, Task, and Bug and for 2 actor groups (Testers and Analysts):

$$\text{RAP(C1, Testers)} = (0.34\%/1,\ 0.71\%/2,\ 1.0\%/3,\ 55.7\%/9)$$
$$\text{RAP(C1, Analysts)} = (8.1\%/4,\ 9.6\%/4,\ 11.4\%/5,\ 6.9\%/5)$$

These vectors confirmed the well-defined roles of the actor groups considered, which indicated testers dominating in reporting bugs and analysts—providing a relatively unbalanced contribution to all issue types.

Similarly, we could derive commenting actor activity profiles (CAP) related to Testers, Developers, and Analysts. Here, the dominant commentators were Testers and Developers, which could be considered a typical situation.

$$\text{CAP(C1)} = (44\%/6, 47\%/12, 8.5\%/2)$$

The contributions of individual testers ranged from 3% to 17%; for developers, it was 2–9%.

5.2. Issue Commenting Activities

A useful issue feature that was related to comments could be considered a metric of issue resolving effort as well as a complimentary characteristic of actors' activities. Here, we considered four statistics: (i) the distribution of the number of registered comments within the reported issues, (ii) the distribution of the number of commenting actors of the issue, (iii) the distribution of the comment sequence duration within the issues (CS), (iv) the distribution of the size of reported comments (e.g., comment length in words). This is illustrated in Table 10 (median, third quartile—Q3, maximum, average values). In the projects considered, the median values for distribution types (i), (ii), and (iii) were in the ranges of 2–4 comments, 2–3 commentators, and 2–47 days, respectively. However, the maximum values were quite significant: 126–506 comments, 13–107 commentators, and 2275–7488 days. The size of comments was in the ranges of 10–39 (median), 32–86 (quartile Q3), and 4834–107,000 words (maximum). This confirms the need for deeper comment analysis based on advanced text mining. The statistical values were lower when compared with issue descriptions (median 35–92, Q3 72–198), except for the maximum of 5684–79,479.

Table 10. Statistics of comments: the number of comments/the number of commenting reporters in the issue (rows with suffix—a), comment sequence time CS in days (rows with suffix—b).

Param	Cassandra	Spark	Flink	Ignite	Mozilla	Red Hat
Med—a	4/3	2/2	3/2	2/2	4/3	2/2
Med—b	5.3	1.9	4.2	2.3	16.2	47.5
Q3—a	8/3	4/3	7/3	4/3	7/4	4/3
Q3—b	48.8	24.0	63.7	21.7	300.0	201.1
Max—a	295/29	126/43	370/22	144/13	506/107	161/42
Max—b	3619.6	3195.6	2803.4	2275.7	7488.4	6152.7
Av—a	6.9/2.9	3.3/2.3	6.0/22.0	3.2/2.4	6.9/3.2	3.2/2.4
Av—b	100.9	73.3	123.6	55.0	360.6	159.3

The minimum values of the number of comments/commentators for issues (parameters with suffix—a) were 0/1, and Q1 values were in the ranges (0–2)/(1–2). The minimum and Q1 values of the comment sequence time (parameters with suffix—b) were 0 and 0.0–0.23 days, respectively. These statistics show the need for deeper analysis. We could trace the delay of the first comment in the issue (TC1) and time between comments in the relevant sequence (CC). We could also check the stability (fluctuation) of these parameters in time (e.g., subsequent quartiles, years of project development). We illustrated this for the MongoDB project by giving Q2 (median) and Q3 quartiles for long-time observation {Q2,Q3} and three subsequent quarters of the year <(Q2, Q3), (Q2, Q3), (Q2, Q3)>:

$$\begin{aligned}&\text{TC1(Mongo DB)} - \{2, 9\}, <(2.9, 9.9); (1.7, 5.9), (2.8, 14)>,\\ &\text{CS(Mongo DB)} - \{0.04, 7\}, <(0, 8), (0.05, 2{,}6), (0.2, 15.6)>,\\ &\text{CC(Mongo DB)} - \{0.9, 6.7\}, <(0.9, 7.1), (0.7, 3.2), (1, 9.7)>\end{aligned}$$

the maximum values were in the range of 178–190 days within the long period and 93–114, 34–40, 178–190 days for the first, the second, and the fourth quarter of 2021 (minimum values were close to 0; Q1 was in the range of a few hours).

For the C1 commercial project, we provided these parameters (Q2 and Q3) covering 3 subsequent sprints {} and values for these sprints separately within brackets <. . .>):

$$\begin{gathered} \text{TC1(C1)} - \{3.1, 14.2\}, <(3, 10); (3, 7), (1, 6)>, \\ \text{CS(C1)} - \{4.7, 19.7\}, <(3, 13), (4, 22), (4, 15)>, \\ \text{CC(C1)} - \{0.95, 4.8\}, <(0.9, 7.1), (1, 5.5), (0.9, 4)> \end{gathered}$$

The maximum values were as follows: 91–165, 62–164, and 56–142 days for TC1, CS, and CC parameters, respectively (with minimum and Q1 values close to 0). The median and Q3 values were below the period of a single sprint, which confirmed the relatively good consistency of handling issues with the sprint plan. Nevertheless, the maximum values exceeded the sprint duration (4 weeks), which required moving unresolved issues to a following sprint.

Usually, the list of commentators and reporters differed. In the case of MongoDB projects, there was no overlap for the 20 most active issue reporters and commentators. Moreover, *Githook user* shell scripts were a significant comment contributor, providing 600–800 comments per quarter, while the most active personal commentators provided 5–70 comments. *Githook* triggered actions in response to specific events to help automatize development workflow (during the lifecycle and while implementing continuous integration). They contributed over 50% of all comments. The average number of comments per issue was about 2 (1–3, depending on issue priority), and there were maximally 17 comments. In the case of the Scrum C1 commercial project, the commentators constituted three groups, which, to some extent, overlapped with these issue reporters: Testers (6), Developers (10 out 15), Business Analysts (4), and the Product owner (1). They generated 547 (34–306), 689 (39–129), 234 (41–102), and 64 comments for three subsequent sprints, respectively. We provided the ranges of comments per commentator in the brackets. These parameters fluctuated for individual sprints; however, general relations were similar. It was observed that the average number of comments per issue was over 6, and there were about 3 comments for urgent and other priority issues. The maximum number of comments within an issue ranged from 7 to 27 (7 for cosmetic priority). Issues with the maximum number of comments were worth tracing with the aim of finding their reasons and possibly delaying resolution. The number of comments and time features could be correlated with the issue-handling processing presented in our previous paper [15]. In [45], we presented text mining schemes for classifying comment categories and analyzing their sequences. This also facilitated the process of detecting discussion deficiencies among project actors.

6. Issue Reporting Dependencies (Correlations)

The statistical distribution of repository attribute contents considered in this study could be derived independently or could be conditioned by selected features from other issue report fields or other repositories. Typically, we could use filters in relevance to issue types, priority, severity, or reporter category. This last feature was rarely available directly (e.g., in the project management documentation); however, it could be derived indirectly by exploring activity features and other reporting traces in textual fields. Diverse correlations of semantic issue features are presented in Section 6.1. On the other hand, generated statistics were based on repository data at some point in time (specified number of registered issues). These statistics could change during the project lifecycle, as was shown in Table 7. Hence, time dependencies of issue features need investigation; we illustrated this in Section 6.2.

6.1. Issue Data Correlations

Most of the studied features of the projects corresponded directly to data comprised in relevant issue report fields, as was illustrated in the previous sections. They could be

observed independently or could be correlated. We considered mutual attribute correlations and attribute correlations with other software repository features, e.g., the scope of performed code corrections. When confronting the first problem, we introduced correlation profiles related to two attributes: A (with n category values) and B (with m category values), defined as:

$$A \mid B: \{<\mathbf{a1} \mid b1>, <\mathbf{a2} \mid b2>. . .<\mathbf{am} \mid bm\}$$

where **ai** ($1 \leq i \leq m$) were vectors specifying the number of issues within subsequent categories of attribute A, which simultaneously assumed value *bi* of the attribute B. To illustrate that, we provided correlation profiles for four pairs of attributes in the Cassandra project: Ch | P, Ch | T, Co | P, and Co | T, which corresponded to attributes *change category* (Ch—<semantic, code clarity, quality, performance, operability>), *priority* (P—{low, normal, high; urgent}, *issue type* (T—{Impr, New Feat, task, bug}), and *complexity* (Co—<low, normal, challenging, Byzantine>). Vector values of the first attribute are given in <>, while subsequent vectors in {. . .} correspond to the values of the second attribute:

Ch | P: {<11, 14, 31, 9, 36>; <182, 79, 200, 64, 310>; <0, 3, 7, 1, 13>; <8, 0, 1, 2, 0>}
Ch | T: {<63, 37, 126, 58, 229>; <10, 2, 8, 2, 55>, <123; 56, 103, 13, 70>; <2, 0, 1, 2, 4>}
Co | P: {<152, 40, 5, 0>; <762, 1202, 65, 4>, <9; 15, 5, 0>; <8, 22, 23, 0>}
Co | T: {<324, 249, 20, 0>; <19, 57, 7, 0>, <137; 238, 9, 0>; <0, 2, 2, 1>}

We could notice that Ch attribute is filled mostly for *Improvement*, *New Feature,* and *task* issues, which require some changes (as opposed to *bugs*). The complexity attribute was filled mostly for normal and low priority issues; for higher priority issues, this information was not considered useful due to the need to provide a fast solution independent of the complexity of the problem. Here, we should note that the filling ratio of compared attributes can differ. In the profiles considered, the attributes *priority* and *type* were filled in at 100%, but *change category* and *complexity* were filled in at 5.2% and 13.1%, respectively. The change category filling ratio distribution in relation to priority was ChF | P = [0.10, 0.87, 0.02, 0.01] and in relation to Type ChF | T = [0.53, 0.08, 0.37, 0.01, 0.01]. The filling ratios for complexity category were as follows: CoF | P = [0.08, 0.88, 0.01, 0.02] and CoF | T = [0.26, 0.04, 0.16, 0.54, 0.00]. Correlation profiles in the cases of attributes with a 100% filling ratio are easier to interpret. In particular, the correlation of reporters with other attributes demonstrated their responsibility in the area of issue documentation. It was useful in projects involving many (usually fluctuating) reporters. In the cases of low filling ratios for compared attributes, correlation profiles might not be satisfactory. Deeper analysis might involve conditional profiles including an additional attribute, e.g., such that improved the relative filling ratios of the compared attributes.

We discussed correlations of project actors with issue types in Section 4.2. Depending on the projects, we could observe various practices. Correlating actors with other attributes might reveal their negligence or lack of competence, e.g., in case of selecting only default or neutral attribute values. An additional dimension in correlation was time (Section 6.2), which allowed us to derive trends and behaviors in relevance to a project cycle phase or a version. Yet another problem consisted in considering the impact of actor fluctuations. We could extend the correlation analysis over diverse repositories, e.g., issue tracking and version control. A metric of bug handling effectiveness was the correlation of reported bugs with code corrections (commits). For this purpose, we used bug correction vector BCV(P):

$$BCV(P) = <Br; Bc; Bc1; \ldots Bck>$$

where Br—was the number of reported bugs, Bc—was the number of bugs requiring corrections, and Bci—was the ratio of bugs requiring i corrections; parameter i could be

specified explicitly or as a range of integer values. To illustrate that, we provided the BCV vectors for three subsequent quarters of the year in the MongoDB project:

$$\begin{aligned} BCV(MongoDB)_1 &= <477; 344; 48.55\%; 47.09\%; 4.36\%>, \\ BCV(MongoDB)_2 &= <693; 477; 64.36\%; 32.29\%; 3.35\%>, \\ BCV(MongoDB)_3 &= <522; 312; 51.28\%; 46.15\%; 2.56\%;> \end{aligned}$$

Bug ratios (presented in percentages) were related to a single correction, 2–4 corrections, and more than 4 corrections. The maximum number of corrections for a single bug issue was 9 (and it was related to 0.3% of issues). A high percentage of Bci for i > 1 could show problems with issue handling, e.g., caused via imprecise bug description, problems with diagnostics, etc. Here, we could investigate correlations with bug description features (see our previous paper [45]), bug reporters, etc. In the case of the C1 commercial project, the BCV vectors derived for subsequent sprints showed better efficiency; typically, single corrections constituted 85–95% (depending on the Scrum sprint), and the maximum number of corrections was 5 (0.3%). We could also evaluate the cost of corrections by providing the distribution of added or modified code files or lines. This problem was analyzed in our previous paper [3], with illustrative results for commercial projects.

6.2. Time Dependancies

The dynamics of the statistical parameters could be visualized in relevant time plots. Here, we distinguished two classes of these plots, namely incremental and cumulative ones. Incremental plots showed the statistics derived from data corresponding to issues registered within the specified time unit (e.g., a week, a month) on a time axis. The time dependencies of issues could be studied in two perspectives:

(1) Graphical plots of the number of issues reported, resolved, or waiting for resolution in time.
(2) Numerical statistics related to issue handling time distributions.

These considerations could be treated globally or differentiated via attribute values (e.g., issue types, priority, resolution scheme). They were based on tracing issue timing attributes, including historical specifications of issue state changes.

Cumulative plots provided statistics corresponding to all issues registered up to the time point on the *x*-axis. Some illustrative plots are provided in Figures 2 and 3. They could also be useful in various predictions (Section 2). In most projects, the distribution (as a percent) of issues by type, priority, and resolution type was relatively stable in time. However, in Mozilla, issues with priority P5 were low (slightly exceeding P4) for most of the time but significantly increased in the last 4 years (even exceeding P1). In Cassandra, low priority issues dominated in the first 7 years and dropped down in comparison with the dominant normal priority of the last 6 years. This could be correlated with growing experience, better adaptation of developers to the project, etc. However, high staff fluctuation might show more inconsistent behavior (see Section 4.2).

Illustrative plots for the Ignite project are given in Figures 2 and 3, where the *x*-axis shows subsequent report dates (year-month-day), and the *y*-axis specifies the number of visualized issues. Figure 2 shows the number of registered issues in four categories (task, bug, improvement, and new feature). Bugs and tasks dominated here. Figure 2 presents statistics in time related to the number of closed issues considering final decisions: qualified as duplicated, resolved according to 5 methods (see Section 4.1), specified as not requiring fixing (Won't fix—7 subcategories), postponed for later resolution, and others (dominate resolved).

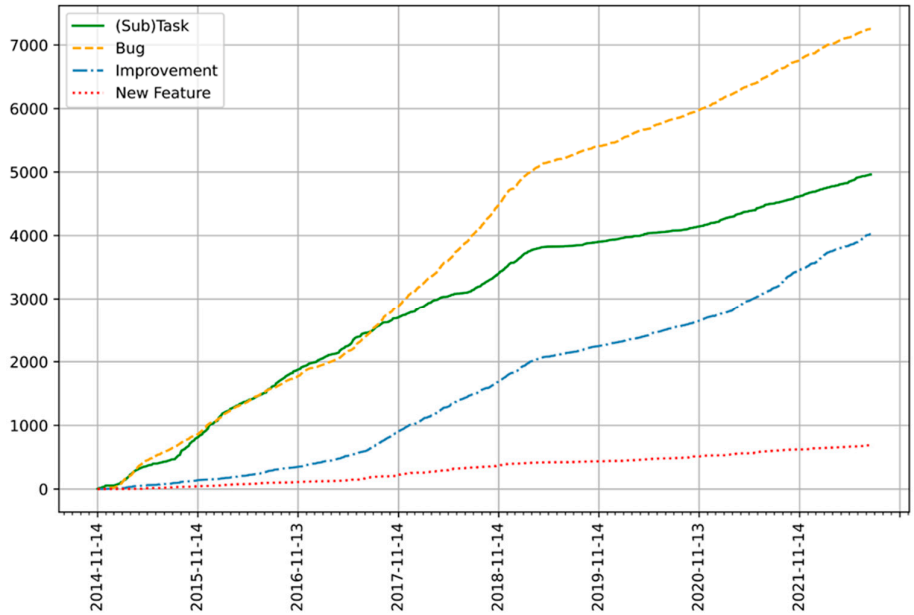

Figure 2. Cumulative plots of the number of reported issue categories for the Ignite project (9-year perspective).

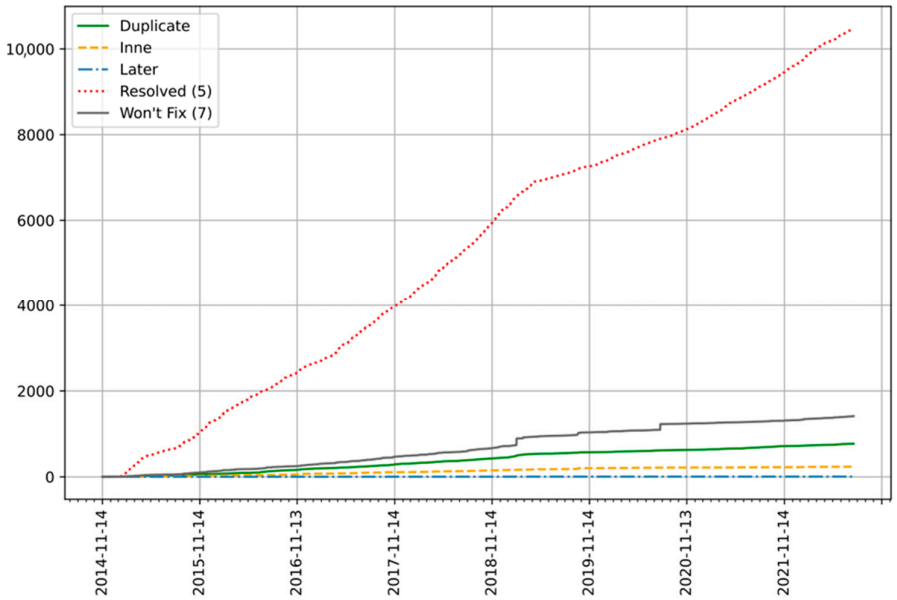

Figure 3. Cumulative plots of the number of issues in relevance to resolution methods for the Ignite project (9-year perspective).

The efficiency of handling issues could be assessed via the distribution of issue handling time. We could measure this time for each issue by taking the timestamp of the issue registration and the time of its resolution. Issue resolution needs some comment; theoretically, it should relate to the closed state; however, quite often this state was skipped, so in the case of the last *resolved_x* state not succeeded by the *closed* one, we took its timestamp as also the final handling time. Yet another problem was related to issues for which the handling process did not terminate in the considered repository time space. Hence, when deriving timing distributions, we provided two numbers in Table 11: those related only to terminated issues (suffix—a) and considering the remaining issues as terminated at the last timestamp of the considered repository (suffix—b). Time plots showing the number of all registered events at a specified moment (starting from the beginning of the repository period) and the number of the closed ones provided another view on issue handling in the

project. Derivative plots could show the ratio (as a percent) of closed issues or the number of unresolved issues. To illustrate that, in Tables 12 and 13, we provided relevant statistics for four projects. Issue handling processes involved several phases, specified in the repository by processing states (with entry and exit times). By extracting timestamps relevant to these states, we could trace issue handling paths and relevant timing features. They were useful in reliability assessment and in the identification of development anomalies. We presented an original analysis of this problem in [15].

Table 11. Issue handling time distribution (days).

Param	Cassandra	Spark	Flink	Ignite	Mozilla	MongoDB
Med—a	19.4	10.6	21.9	43.0	40.0	232.2
Med—b	11.1	7.8	11.8	15.8	15.1	28.4
Q3—a	193.9	152.8	191.0	471.6	538.8	326.1
Q3—b	71.6	76.5	60.3	67.8	276.0	52.3
Max—a	4496.7	3206.0	2974.3	2811.1	8296.8	499.6
Max—b	3359.3	3206.0	2564.7	2362.0	7033.6	358.2
Av—a	324.5	188.0	219.7	400.9	726.3	242.5
Av—b	110.5	153.9	111.2	108.3	304.7	45.2

Table 12. Ratio of not resolved issues in subsequent 10 years [%].

Year	1	2	3	4	5	6	7	8	9	10
Cas	0.15	0.14	0.07	0.09	0.10	0.12	0.13	0.13	0.14	0.14
Spark	0.38	0.27	0.23	0.14	0.13	0.13	0.14	0.06	0.08	0.07
Flink	0.22	0.19	0.18	0.21	0.20	0.21	0.19	0.15	0.17	0.16
Moz	0.25	0.26	0.18	0.17	0.16	0.16	0.14	0.13	0.13	0.12
MonDB	0.38	0.06	0.05	0.07	0.07	0.06	0.07	0.10	0.10	0.18

Table 13. The number of issues waiting for resolution in the subsequent 10 years.

Year	1	2	3	4	5	6	7	8	9	10
Cas	450	490	700	1150	1520	1750	2000	2250	2300	2320
Spark	150	400	1500	2100	2600	3350	3800	1850	1900	2650
Flink	200	450	820	1370	1860	2500	3460	3350	4600	5200
Moz	1100	2600	3500	5150	6070	6230	6350	6750	7000	7250
MonDB	123	330	543	795	1021	1271	1557	1992	2458	3263

The ratio of unresolved issues was usually higher in the first years of the project, which showed better problem knowledge and adaptation to handling issues (Table 13). However, the absolute number of unresolved issues increased in time, which could have triggered the employment of additional staff (Table 12). Here, an important issue was the speed of this increase, e.g., checked by tracing the number of additional unresolved issues in subsequent time periods (e.g., months—incremental plots). In long-time projects, the number of reported issues per month might have increased in time due to wider usage, performed updates, added new functions, etc. In the C1 commercial project developed according to the Scrum scheme, the ratio of unresolved issues in the subsequent 6 years was relatively low: 0.17, 0.06, 0.07, 0.07, 0.06, and 0.12. In general, a decreasing ratio of unresolved issues was required. This could be achieved by adapting the number of

issue handling staff and improving the efficiency of this process (e.g., improved/automatic diagnosis). It was also important to ensure a small percentage of unresolved issues of higher priority. For the 5 projects considered in Table 12, we obtained fewer than 10, 50, 80, 100, and 60 of unresolved issues of the two highest priorities, which constituted less than 1%. Moreover, the handling times of these issues were lower than in the other cases. This feature was also confirmed in the C1 commercial project.

In many projects, we could observe a postponing of the resolution of low-level priority (e.g., cosmetic) issues. This may have caused the so-called effect of bug debt [46], which leads to overlooking important problems due to the false qualification of issue priority (class) and the combined impact of many postponed issues, which could also be triggered in correlation with code upgrades or system configuration. When confronting this problem, we introduced an algorithm that detected a significant increase in such postponed issues and initiated their investigation with the aim of searching for suspicious ones and with a view of checking their criticality. This analysis was performed with the use of issue description text mining based on developed machine learning algorithms [45]. The output of this analysis was the set of suspicious issues that required deeper analysis. Some experiments with several projects showed that the algorithm filtered out only several percent of the postponed issues as suspected; deeper analysis was restricted only to these issues, and it confirmed that 80–90% of them were critical.

Resolving the issues triggered activities of diverse project contributors (e.g., testers, analyst, developers). It was reasonable to control their workload and efficiency. This could be traced in correlation with the issue handling model introduced in our previous work [15]. Nevertheless, we could gain a general insight into this problem by deriving the distribution of performed code changes, which had a significant impact on testers' and developers' workflow. To illustrate that, we provided statistics in the a/b% form, where a denoted the number of performed code changes, and b% denoted the percentage of relevant issues:

MongoDB: 1/48.5%, 2/22.1%, 3/17.7%, 4/7.3%, 5/2.6%, 6//0.3%, 7/0.3%, 8/2.3%, 9/0.3%
C1: 1/84%, 2/11.3%, 3/4.1%,4/0%, 5/0.3%

In [3], we generated statistics of the number of changed files and code lines in relation to the number of performed commits. Higher numbers of changes (or their scope) might have revealed the need for better issue partitioning. We should also note that issue resolution might not need code changes, e.g., an unconfirmed problem and duplicated or negligible issues. Moreover, changing the environment configuration was sometimes sufficient. Nevertheless, such resolutions of issues required activities of appropriate project actors (e.g., analysts, testers). In the C1 Scrum project, only 49.8% of reported issues needed code changes/extensions, and many issues needed configuration refinement. We could also trace the number of code or file changes in time; quite often they decreased as the project stabilized or became mature. In the C1 project, the average number of changed files per commit decreased from 3 to 1 within two years.

7. Discussion

The methodology of assessing issue reports presented in this study confirmed the availability of valuable information comprised in a multitude of attributes, which was neglected in the literature (see Sections 2 and 3). Our investigation shows that our approach and results fill a gap in the literature on issue tracking in the following areas: (i) the number of attributes considered (typically a few in the literature versus a few dozen in the paper), (ii) the assessment of attribute values and the importance of information (neglected in the literature and explored in our approach based on the profiles introduced), (iii) the identification of repository imperfections/anomalies requiring improvements (which was possible due to points (i) and (ii)). The analysis conducted revealed certain deficiencies in attribute specifications, primary due to actors' negligence in filling them, inattention in ensuring that relevant values were fixed (for instance due to diverse reporter habits or reporter fluctuations), and changes observed over time. This might have been due to a

misunderstanding of the attributes by issue reporters, a lack of awareness of the attribute significance, or the high fluctuations of actors in the project lifecycle.

When monitoring the introduced issue/attribute profiles, we could identify diverse anomalies in issue reporting and attribute specifications; project actors' activities or contributions (including comments); and issue-handling schemes (inflow, resolution profiles, and timing dependencies). They were illustrated for real projects in Section 4, Section 5, and Section 6, respectively. The results of such an analysis could be disclosed to project stakeholders and discussed systematically with the project manager to introduce these relevant improvements: (i) global ones—redefining issue reporting policy and attribute values, adding new attributes, deleting redundant or irrelevant ones, presenting description and documentation deficiencies, e.g., related to excessive number of comments to questions, etc.; and (ii) individual ones—addressed at actors showing differing operating profiles compared with typical ones, e.g., the excessive number of rejected issues reported by the users denoting their poor knowledge of documentation, the descriptions of issues reported by testers with a low percentage of technical objects being considered too superficial (difficult to diagnose, requiring additional clarifications in the comments, which results in longer issue resolution time), etc.

The analysis of open-source projects revealed the following observations: (i) the uneven activity of project actors (typically, 10 actors contributed 20–30% of issues; 20–30% reporters were active only on a single day in the project lifecycle) and (ii) non-uniform attribute specification, the appearance of new values in time, the careless selection of values (e.g., dominating default values in a Thunderbird project), lacking or ambiguous attribute definitions (organizational mess). Moreover, the number of non-closed issues (including the postponed ones) in some projects is too high, which indicates a certain shortage of developers and testers. On the other hand, there were issues closed in a very short time (up to 10 minutes) without any additional information (probably duplicates). These aspects were neglected in classical issue predictions; hence, a more accurate analysis should take these observations into consideration.

The detailed analysis of developed commercial projects showed fluctuations in the time of attribute features, e.g., filling ratio and value distribution. They resulted from project stakeholders' fluctuations (new people needed some time to understand the project) or changes introduced by project managers, e.g., new attributes resulted in the usage of other attributes, which could be deleted. Some changes (e.g., project actors) could be imposed by introducing new functionalities or restructuring the project team (including the project manager). Hence, it was important to revise the set of attributes and their values periodically, especially in projects with a long lifetime. This recommendation allowed us to reduce the number of attributes by 50% in projects considered in [3] as well as reduce the number of issue processing states. Recently, many software companies employed external jobholders for specific tasks, which resulted in the diversity of their competence and personal fluctuation in time. Hence, it was reasonable to monitor their impact on reported issue features. For example, in the C1 project, introducing 8 new testers caused a significant change in the issue description attribute by extending it by 60%; moreover, the relevant issues showed a higher number of involved corrections per issue. The reported issues were too complex; hence, the recommendation was to split them into simpler problems. In C1, the project manager presented assessments of issue features regularly after each sprint, and deeper discussions with the team were held several times per year.

The researchers and practitioners who utilize ITS systems to deal with software development problems should understand issue reporting practices assumed in the projects. Unfortunately, attribute category values are quite often not clearly specified; this stems from using symbolic names known only to specific actor groups, data availability limitations in commercial projects, and the lack of information on project stakeholder categories and skills. This is especially crucial in influential issue attributes. Such problems could be traced with the proposed issue, actor activity, and correlation profiles. An important aspect was the practitioners' perception regarding the integration of ITS features into software

analytics tools supporting development processes. Upon comparing repository profile metrics between diverse projects, we could also incorporate good practices from other projects or identify suspicious symptoms for further exploration (lessons learned from retrospective analysis of good and bad practices). This was useful in attribute aggregation, standardization, splitting, deletion, adaptation to project environment, etc.

Deep issue evaluation was helpful for the project manager to control relevant processing and team activities. Nevertheless, identified issue reporting features could be discussed in interviews with project stakeholders and experts. Moreover, they could be correlated with Version Control Systems (VCS) [47] to identify fuzzy feature descriptions or insufficient traceability and introduce improvement decisions in future practices. Exchanging experiences from diverse projects and teams, based on the comparative statistical metrics proposed in the paper, could provide synergistic effects.

Many authors confronted the problem of software monitoring for different purposes, such as detecting abnormal behavior or finding performance issues (see the survey paper [48]) by considering log data. We showed the richness of issue repositories and their usefulness in project development assessment. The data derived could also be correlated with relevant logs generated in relevance to specified issues. Similarly, these studies could be combined with the results of software bug predictions based on software complexity metrics [49] (the identification of components prone to defects). This could extend the space for further investigations. By correlating issue resolution times with issue types, priorities, and project actors, we could study their impact on the fixing time [2]. Upon checking the number of times a given issue is reopened or duplicated, we could evaluate its complexity or deficiencies of reporting in the ITS. The developed PHG model from our previous publication [15] was useful in this process, since it refers to historical values of the issue state attribute.

The models of diverse software feature predictions or recommendations performed better with higher quality issue reports. In general, this related to report accuracy and completeness. These features could be assessed with the deep exploration of attributes used, their filling ratio, value distribution, and diverse information profiles or dependencies proposed in the paper. They were considered in relevance to issue types, attribute categories, project actors, and observation perspectives. Our findings suggested that issue reporters are often unaware of attribute values, and other project actors may not correctly understand the context of the data being produced in the repository. Hence, the presented issue evaluation methodology and metrics could improve ITS quality and project stakeholders' responsibility in the area of producing and exploring repository data.

Issue attribute data had a significant impact on project evaluation, especially on performed predictions. For example, fluctuations of actor activities might impede software reliability predictions. Here, it was also important to distinguish bugs reported by diverse project contributors (e.g., testers, users) as well as bug priorities, which are handled with different involvement. The statistical profiles and metrics presented were useful to assess the quality of issue reporting. The number of non-NL objects in issue/comment descriptions (Table 7) provided some overview of technical contents (important for bug localization). Here, we could also include statistics of diverse links. In [50], issue relationship links indicating technical or workflow dependencies were studied. Issues with many links or multiple link types could also indicate that an issue was too big and needed to be broken down into multiple parts. Correlation analysis with other repositories, e.g., SVC repositories, also provided another perspective of assessment. For example, the distribution of performed commits in relevance to issues could show deficiencies in task partitioning (bug correction vectors (BCV) specified in Section 6.1).

The presented considerations were illustrated on the basis of selected repositories relevant to open-source and a few commercial projects. Hence, some comment is needed on the internal and external validity of our approach. Internal validity was assured by dealing with complete data extracted from project repositories and using well defined statistical profiles/metrics supported with analysis scripts and the manual tracing of suspicious cases.

The limited number of the projects that were considered hindered external validity, and thus the extent to which our results could be generalized. However, due to the diversity of project repository profiles (attribute features, reporter activities, and timing schemes) we can state that the overall approach is transferable. Moreover, some results were related to commercial projects in which we were involved. Hence, this was quite a representative project selection with diverse processing schemes compatible with the Jira platform. The number and diversity of analyzed attributes, as well as their analysis scope (covering various perspectives and proposed statistical profiles), significantly exceeded that reported in the literature. The assessment metrics and profiles presented here can be adapted to the specificity of other projects, e.g., diverse issue types, textual objects, or analysis perspectives.

There are some limitations to the interpretations of the analysis results due to the lack of feedback from stakeholders of open-source projects. In commercial projects, some specific reporting schemes and attributes can be encountered; moreover, there is a problem of data confidence mostly imposed by product owners (the ordering party). We can only negotiate the scope of using these data or the anonymization range. Deeper interpretation of results requires some feedback from project stakeholders, e.g., obtaining responses in appropriate questionnaires [51]. They can be focused on listing drawbacks in issue trackers used, proposed attributes to be added or deleted, comments on issue report deficiencies, and suggestions of relevant improvements, etc. When working out such questionnaires, we can address problems identified in the issue analysis performed with our approach.

In this paper, we presented illustrative examples related to Jira repositories. These repositories are quite popular among practitioners and researchers [41,51,52]. Most research studies in Section 2 refer to Jira and Bugzilla ITS systems. Bugzilla is limited to only tracing bugs, while Jira also includes other issues (requests of new functions, improvements, and other tasks). Moreover, its interface is more flexible and is adapted to new development technologies (e.g., Agile, Scrum). Nevertheless, this study could also handle other repositories due to the similarity of report contents. This would only require the adaptation of REST API to the repositories considered and the performance of relevant data parsing; however, the general ideas are comparable. The analysis methodology developed here, including introduced metrics and profiles, is universal; moreover, it can be further extended. This can also be enhanced with other repositories, e.g., GitHub and testing reports. However, here we might face the problem of communication between repositories [53]. These aspects can be addressed in subsequent studies, which are beyond the scope of our investigation presented in this paper.

8. Conclusions

The advanced studies of software repositories performed in this paper revealed rich data contents, which documented development processes involving diverse tasks, their specificities, importance, critical aspects, emerging problems, activities of project actors, etc. The investigation of these data required the development of special software tools to extract characteristic features from raw data. These involved well-defined aspects consistent with the issue taxonomy introduced. It included the deep analysis of category and textual attributes of the issues as well as involvement of project actors and data interrelationships. We summarized the scope of use, benefits, and limitations of our approach in Section 7.

The approach proposed involves original evaluation profiles and metrics related to issue reporting aspects, the activity of project actors, etc. Illustrative results were derived both for open source and commercial projects. They confirmed the practical significance of detailed repository content studies to enhance the assessment of project effectiveness and identify repository deficiencies (e.g., inconsistent attribute values in the project). Diverse observation perspectives and multidimensional assessment profiles revealed many aspects neglected in classical software engineering studies. The diversity of issue features (scope and assumed values), activities of project stakeholders (their involvement, fluctuations), and issue-reporting data or timing dependencies have a significant impact on assessing

project progress, quality, and performed predictions. The results presented show a need for multidimensional predictions, including specific features of issue types, actor activities, etc.

Further research will be targeted at enhancing classical software reliability and process control studies with additional issue and actor characteristics. Deeper correlations of the issue repository with software version control reports require further studies. Here, machine learning techniques can be enhanced by considering specific issue-derived features, as was undertaken in our previous paper [45] on text mining. We also plan to combine the issue feature analysis presented here with the previously introduced PHG graph model [15] to trace bug-handling processes (paths). We plan to extend it to other issue types and provide fine-grained tracing, considering resolution efficiency in relation to diverse attributes (types, priority, project actor classes, etc.).

Author Contributions: Conceptualization, J.S.; methodology and investigation, J.S., B.D. and Ł.R.; software and experiments, Ł.R. and B.D.; validation, J.S., writing—original draft preparation and editing, J.S.; visualization, Ł.R. and B.D.; supervision, J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are unavailable due to privacy and other restrictions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Aljedaani, W.; Javed, Y. Bug reports evolution in open source systems. In *5th International Symposium on Data Mining Applications*; Advances in Intelligent Systems and Computing; Springer: Berlin/Heidelberg, Germany, 2018; Volume 753. [CrossRef]
2. Nayrolles, M.; Hamou-Lhadj, A. Towards a classification of bugs to facilitate software maintainability tasks. In Proceedings of the ACM/IEEE 1st International Workshop on Software Qualities and Their Dependencies, ACM, Gothenburg, Sweden, 28 May 2018. [CrossRef]
3. Polaczek, J.; Sosnowski, J. Exploring the software repositories of embedded systems: An industrial experience. *Inf. Softw. Technol.* **2021**, *131*, 106489. [CrossRef]
4. Izadi, M.; Akbari, K.; Heydarnoori, A. Predicting the objective and priority of issue reports in software repositories. *Empir. Softw. Eng.* **2022**, *50*, 27. [CrossRef]
5. Jahanshahi, H.; Cevik, M.; Başar, A. Predicting the Number of Reported Bugs in a Software Repository. In *Advances in Artificial Intelligence*; Goutte, C., Zhu, X., Eds.; Canadian AI. Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12109, pp. 309–320. [CrossRef]
6. Banerjee, S.; Syed, Z.; Helmick, J.; Culp, M.; Ryan, K.; Cukic, B. Automated triaging of very large bug repositories. *Inf. Softw. Technol.* **2017**, *89*, 1–13. [CrossRef]
7. Ebrahimi, N.; Trabelsi, A.; Islam, S.; Hamou-Lhadj, A.; Khanmohammadi, K. An HMM-based approach for automatic detection and classification of duplicate bug reports. *Inf. Softw. Technol.* **2019**, *113*, 98–109. [CrossRef]
8. Nadeem, A.; Sarwar, M.U.; Malik, M.Z. Automatic issue classifier: A transfer learning framework for classifying issue reports. In Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Wuhan, China, 25–28 October 2021; pp. 421–426. [CrossRef]
9. Aljedaani, W.; Javed, Y.; Alenezi, M. Open source systems bug reports: Meta-Analysis. In Proceedings of the 3rd International Conference on Big Data and Education (ICBDE'20:), ACM, London, UK, 1–3 April 2020. [CrossRef]
10. Sanei, A.; Cheng, J.; Adams, B. The impacts of sentiments and tones in community-generated issue discussions. In Proceedings of the IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), Madrid, Spain, 20–21 May 2021; pp. 1–10.
11. Hanagal, D.; Bhalerao, N. *Software Reliability Growth Models*; Springer: Berlin/Heidelberg, Germany, 2021; ISBN 978-981-16-0025-8.
12. Elmishali, A.; Kalech, M. Issues-Driven features for software fault prediction. *Inf. Softw. Technol.* **2023**, *155*, 107102. [CrossRef]
13. Rana, R.; Staron, M. When do software issues and bugs get reported in large open source software projects? In Proceedings of the International Conference on Software Measurement, IWSM-Mensura, Kraków, Poland, 5–7 October 2015.
14. Edwards, N.; Jongsuebchoke, D.; Storer, T. Sciit: Aligning Source Control Management and Issue Tracking Architectures. In Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 29 September–4 October 2019; pp. 402–405. [CrossRef]
15. Sosnowski, J.; Dobrzyński, B.; Janczarek, P. Analysing problem handling schemes in software projects. *Inf. Softw. Technol.* **2017**, *91*, 56–71. [CrossRef]
16. Rakha, M.S.; Bezemer, C.-P.; Hassan, A.E. Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval. *Empir. Softw. Eng.* **2018**, *23*, 2597–2621. [CrossRef]

17. Lunesu, M.I.; Tonelli, R.; Marchesi, L.; Marchesi, M. Assessing the Risk of Software Development in Agile Methodologies Using Simulation. *IEEE Access* **2021**, *9*, 134240–134258. [CrossRef]
18. Yadav, A.; Singh, S.K.; Suri, J.S. Ranking of software developers based on expertise score for bug triaging. *Inf. Softw. Technol.* **2019**, *112*, 1–17. [CrossRef]
19. Hussain, M.; Khan, H.U.; Khan, A.W.; Khan, S.U. Prioritizing the Issues extracted for Getting Right People on Right Project in Software Project Management from Vendors' Perspective. *IEEE Access* **2021**, *9*, 8718–8732. [CrossRef]
20. Santos, F. Supporting the Task-driven Skill Identification in Open Source Project Issue Tracking Systems. *ACM SIGSOFT Softw. Eng. Notes* **2023**, *48*, 54–58. [CrossRef]
21. Goyal, A.; Sardana, N. Feature ranking and aggregation for bug triaging in open-source issue tracking systems. In Proceedings of the 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 28–29 January 2021; pp. 871–876. [CrossRef]
22. Sarkar, A.; Rigby, P.C.; Bartalos, B. Improving bug triaging with high confidence predictions at Ericsson. In Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 29 September–4 October 2019; pp. 81–91. [CrossRef]
23. Kim, J.; Lee, S. An Empirical Study on Using Multi-Labels for Issues in GitHub. *IEEE Access* **2021**, *9*, 134984–134997. [CrossRef]
24. Herbold, S.; Trautsch, A.; Trautsch, F. On the feasibility of automated prediction of bug and non-bug issues. *Empir. Softw. Eng.* **2020**, *25*, 5333–5369. [CrossRef]
25. Jiang, Y.; Lu, P.; Su, X.; Wang, T. LTRWES: A new framework for security bug report detection. *Inf. Softw. Technol.* **2020**, *124*, 106314. [CrossRef]
26. Peters, F.; Tun, T.T.; Yu, Y.; Nuseibeh, B. Text Filtering and Ranking for Security Bug Report Prediction. *IEEE Trans. Softw. Eng.* **2019**, *45*, 615–631. [CrossRef]
27. Panichella, S.; Canfora, G.; Di Sorbo, A. "Won't We Fix this Issue?" Qualitative characterization and automated identification of wontfix issues on GitHub. *Inf. Softw. Technol.* **2021**, *139*, 106665. [CrossRef]
28. Wu, X.; Zheng, W.; Pu, M.; Chen, J.; Mu, D. Invalid bug reports complicate the software aging situation. *Softw. Qual. J.* **2020**, *28*, 195–220. [CrossRef]
29. Umer, Q.; Liu, H.; Sultan, Y. Sentiment based approval prediction for enhancement reports. *J. Syst. Softw.* **2019**, *155*, 57–69. [CrossRef]
30. Merten, T.; Falis, M.; Hubner, P.; Quirchmayr, T.; Bursner, S.; Paech, B. Software feature request detection in issue tracking systems. In Proceedings of the IEEE 24th International Requirements Engineering Conference (RE), Beijing, China, 12–16 September 2016; pp. 166–175. [CrossRef]
31. Alonso-Abad, J.M.; López-Nozal, C.; Maudes-Raedo, J.M.; Marticorena-Sánchez, R. Label prediction on issue tracking systems using text mining. *Prog. Artif. Intell.* **2019**, *8*, 325–342. [CrossRef]
32. Ramírez-Mora, S.L.; Oktaba, H.; Gómez-Adorno, H.; Sierra, G. Exploring the communication functions of comments during bug fixing in Open Source Software projects. *Inf. Softw. Technol.* **2021**, *136*, 106584. [CrossRef]
33. Arya, D.; Wang, W.; Guo, J.L.; Cheng, J. Analysis and detection of information types of open source software issue discussions. In Proceedings of the IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 25–31 May 2019; pp. 454–464. [CrossRef]
34. Huang, Y.; da Costa, D.A.; Zhang, F.; Zou, Y. An empirical study on the issue reports with questions raised during the issue resolving process. *Empir. Softw. Eng.* **2019**, *24*, 718–750. [CrossRef]
35. Choetkiertikul, M.; Dam, H.K.; Tran, T.; Pham, T.; Ragkhitwetsagul, C.; Ghose, A. Automatically recommending components for issue reports using deep learning. *Empir. Softw. Eng.* **2021**, *26*, 14. [CrossRef]
36. Rath, M.; Mäder, P. Structured information in bug report descriptions—Influence on IR-based bug localization and developers. *Softw. Qual. J.* **2019**, *27*, 1315–1337. [CrossRef]
37. Li, Z.; Jiang, Z.; Chen, X.; Cao, K.; Gu, Q. Laprob: A Label propagation-Based software bug localization method. *Inf. Softw. Technol.* **2020**, *130*, 106410. [CrossRef]
38. Gomes, L.A.F.; Torres, R.d.S.; Côrtes, M.L. Bug report severity level prediction in open source software: A survey and research opportunities. *Inf. Softw. Technol.* **2019**, *115*, 58–78. [CrossRef]
39. Qamar, K.A.; Sülün, E.; Tüzün, E. Taxonomy of bug tracking process smells: Perceptions of practitioners and an empirical analysis. *Inf. Softw. Technol.* **2022**, *150*, 106972. [CrossRef]
40. Tu, F.; Zhu, J.; Zheng, Q.; Zhou, M. Be careful of when: An empirical study on time-related misuse of issue tracking data. In Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista, FL, USA, 4–8 November 2018; pp. 307–318. [CrossRef]
41. Montgomery, L.; Lüders, C.; Maalej, W. An alternative issue tracking dataset of public jira repositories. In Proceedings of the 19th International Conference on Mining Software Repositories, New York, NY, USA, 23–24 May 2022; pp. 73–77. [CrossRef]
42. Martinez-Fernandez, S.; Vollmer, A.M.; Jedlitschka, A.; Franch, X.; Lopez, L.; Ram, P.; Rodriguez, P.; Aaramaa, S.; Bagnato, A.; Choras, M.; et al. Continuously Assessing and Improving Software Quality with Software Analytics Tools: A Case Study. *IEEE Access* **2019**, *7*, 68219–68239. [CrossRef]

43. Yang, Z.; Wang, C.; Shi, J.; Hoang, T.; Kochhar, P.; Lu, Q.; Xing, Z.; Lo, D. What Do Users Ask in Open-Source AI Repositories? An Empirical Study of GitHub Issues. In Proceedings of the 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), Melbourne, Australia, 17 March 2023; pp. 79–91. [CrossRef]
44. Lasynskyi, M.; Sosnowski, J. Extending the Space of Software Test Monitoring: Practical Experience. *IEEE Access* **2021**, *9*, 166166–166183. [CrossRef]
45. Dobrzyński, B.; Sosnowski, J. Text mining studies of software repository contents. In Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering/Kaindl Hermann, Mannion Mike, Maciaszek Leszek A. (red.), Prague, Czech Republic, 24–25 April 2023; SciTePress: Setubal, Portugal, 2023; pp. 562–569. [CrossRef]
46. Li, Y.; Soliman, M.; Avgeriou, P. Identifying self-admitted technical debt in issue tracking systems using machine learning. *Empir. Softw. Eng.* **2022**, *27*, 131. [CrossRef]
47. Seiler, M.; Paech, B. Using tags to support feature management across issue tracking systems and version control Systems. In *Requirements Engineering: Foundation for Software Quality. REFSQ 2017*; Grünbacher, P., Perini, A., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2017; Volume 10153. [CrossRef]
48. Cândido, J.; Aniche, M.; van Deursen, A. Log-based software monitoring: A systematic mapping study. *PeerJ Comput. Sci.* **2021**, *7*, e489. [CrossRef] [PubMed]
49. Hernández-Molinos, M.J.; Sánchez-García, A.J.; Barrientos-Martínez, R.E.; Pérez-Arriaga, J.C.; Ocharán-Hernández, J.O. Software Defect Prediction with Bayesian Approaches. *Mathematics* **2023**, *11*, 2524. [CrossRef]
50. Lüders, C.M.; Bouraffa, A.; Maalej, W. Beyond duplicates: Towards understanding and predicting link types in issue tracking systems. In Proceedings of the 19th International Conference on Mining Software Repositories, New York, NY, USA, 23–24 May 2022; pp. 48–60. [CrossRef]
51. Raatikainen, M.; Motger, Q.; Lüders, C.M.; Franch, X.; Myllyaho, L.; Kettunen, E.; Marco, J.; Tiihonen, J.; Halonen, M.; Männistö, T. Improved Management of Issue Dependencies in Issue Trackers of Large Collaborative Projects. *IEEE Trans. Softw. Eng.* **2022**, *49*, 2128–2148. [CrossRef]
52. Diamantopoulos, T.; Nastos, D.-N.; Symeonidis, A. Semantically-enriched Jira issue tracking data. In Proceedings of the IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), Melbourne, Australia, 15–16 May 2023; pp. 218–222. [CrossRef]
53. Urrea-Contreras, S.J.; Flores-Rios, B.L.; González-Navarro, F.F.; Astorga-Vargas, M.A.; Ibarra-Esquer, J.E.; Pacheco, I.A.G.; Agüero, C.L.P. Process mining model integrated with control Flow, case, organizational and time perspectives in a software development project. In Proceedings of the 10th International Conference in Software Engineering Research and Innovation (CONISOFT), Ciudad Modelo, San José, Chiapa, Mexico, 24–28 October 2022; pp. 92–101. [CrossRef]