# Replicating Mining Studies with *SOFAS*

Giacomo Ghezzi and Harald C. Gall
s.e.a.l. – software evolution and architecture lab
Department of Informatics
University of Zurich, Switzerland
{ghezzi, gall}@ifi.uzh.ch

***Abstract*—The replication of studies in mining software repositories (MSR) is essential to compare different mining techniques or assess their findings across many projects. However, it has been shown that very few of these studies can be easily replicated. Their replication is just as fundamental as the studies themselves and is one of the main threats to validity that they suffer from. In this paper, we show how we can alleviate this problem with our *SOFAS* framework. *SOFAS* is a platform that enables a systematic and repeatable analysis of software projects by providing extensible and composable analysis workflows. These workflows can be applied on a multitude of software projects, facilitating the replication and scaling of mining studies. In this paper, we show how and to which degree replication can be achieved. We investigated the mining studies of MSR from 2004 to 2011 and found that from 88 studies published in the MSR proceedings so far, we can fully replicate 25 empirical studies. Additionally, we can replicate 27 additional studies to a large extent. These studies account for 30% and 32%, respectively, of the mining studies published. To support our claim we describe in detail one large study that we replicated and discuss how replication with *SOFAS* works for the other studies investigated. To discuss the potential of our platform we also characterise how studies can be easily enriched to deliver even more comprehensive answers by extending the analysis workflows provided by the platform.**

## I. Introduction

Mining software repositories (MSR) has a strong foundation in empirical studies,however, a systematic approach to replicability is still missing. As reported by Robles [31], very few published studies can be easily replicated. The problem lies in both the tools and the data used by such tools. Tools are available for approximately 20% of the studies and another 20% are only partially available [31]. Moreover, even when publicly available, they are difficult to set up and use. As a matter of fact, they are mostly prototypes (or just a collection of scripts) working under specific operating systems and settings, and are usually offered "as-is" without user manuals or technical documentation. Data can be divided into "raw" data and processed data. "Raw data" can be directly retrieved from publicly available sources such as version control systems, issue trackers, plain source code, mailing lists, etc. Processed data, which is what is actually used by researchers to perform their analyses, is the result of the retrieval and processing of such raw data. While "raw" data is usually widely available (at least in the case of OSS projects), processed data is not.

Different approaches have already been proposed to address this problem. But these efforts are mainly aimed at creating large, internet accessible, evolution analysis data repositories, similarly to PROMISE [8]. Some of them offer a query-able, static collection of data for specific projects fetched from single [26] or multiple sources [2], [27]; others allow the user to interactively run specific analyses on her own projects of interest [15], [16] through online web applications. Large, static software data repositories give third party applications and analyses a sizeable, common body of knowledge to build upon. They can also be useful to provide benchmark data to test and compare similar tools/analysis that use such data. However, they do not target the *replication* of the actual analyses and are based on a fixed set of software projects. The existing interactive approaches that address this issue limit the user to a few predefined analyses. Replicability is thus still limited to very specific cases. A broader and more systematic approach to replicability is missing. We claim that a solution such as our *SOFAS* (SOFtware Analysis Services) platform [12] can cover this need.

*SOFAS* is a RESTful software analysis architecture and platform that was originally created to support interoperability of distributed software analyses in flexible and lightweight way. It is made up of three main constituents: Software Analysis Web Services, Software Analysis Ontologies and a Software Analysis Broker. The Software Analysis Web Services offer different software evolution analyses as standard RESTful web service interfaces. They use specific Ontologies to define and represent the data they consume and produce. The Software Analysis Broker is the single entry point to all services, acting as an interface between the services and the users. It contains a *Services Catalog* of all the registered analysis services with respect to a specific software analysis taxonomy. These analyses can be easily accessed and invoked by passing them the URLs of the source control repository, the issue tracking system, or release notes, etc. Analysis services can be combined into analysis workflows that perform specific, composite analysis tasks. A custom composition language derived from WS-BPEL called *SCoLa* was employed to specify these workflows [11] and make them reusable across projects and analysis runs.

In this paper, we show how and to which degree replication can be achieved with *SOFAS*. We investigated the mining studies of MSR from 2004 to 2011 and found that out of 88 studies published in the MSR proceedings so far, we can fully replicate 25 empirical studies; additionally, we can replicate an additional 27 studies to a large extent. These studies account respectively for 30% and 32% of the mining studies published.

To support our claim we describe in detail one large study that we replicated and discuss how replication with SOFAS works for the other studies investigated. To discuss the potential of our platform we also characterize how studies can be easily enriched to deliver even more comprehensive answers by extending the analysis workflows provided by the platform.

The main contribution of this paper is a showcase of the use of *SOFAS* as a viable way to replicate mining studies. In particular, we argue for a platform versus ad-hoc scripting, we advertise repeatable processes versus manual configurations, we provide for flexible and scalable analysis and focus on a technology independent framework.

The remainder of the paper is structured as follows: in Section II we give a brief overview of *SOFAS* (for a detailed description we refer to [12]). In Section III, we present the method used in this paper to assess the replicability offered by *SOFAS* and show the results. To further substantiate these results and better show the potential of *SOFAS* in such a replication context, Section IV presents in detail the replication and extension of one of the mining studies. Section VI gives an overview of the related work. We then conclude with a discussion of the strengths and weaknesses of the framework and an outlook onto the next steps.

## II. *SOFAS*

*SOFAS* is a RESTful platform devised to provide software analyses as RESTful web services and to combine them into analysis workflows in an easy and effective way. It is made up by three main components: Software Analysis Web Services, a Software Analysis Broker, and Software Analysis Ontologies. Software Analysis Web Services expose the functionality and data of software (evolution) analyses through a standard RESTful web service interface. The Software Analysis Broker, acts as the services manager and the interface between the services and the users. It contains a catalog of all the registered analysis services. Moreover, it also offers interfaces for both humans and computers to compose such services into workflows and execute them. Ontologies define and represent the data consumed and produced by the different services. In the following we briefly describe each of these three components.

### *A. Software Analysis Web Services*

From a user's perspective, software analyses are inherently linear and uniform in the way they work. Given some information about a software project (be it the code, its source code repository, some data already calculated by an analysis, etc.) and possible analysis calibration settings, they extract and/or calculate their specific data. Once that is completed, the results can be fetched. Such linearity and uniformity make REST a perfect fit for our needs, as some of its inherent principles are also the main requirements and characteristics of our services.

A RESTful web service provides a uniform interface to the clients, no matter what the service actually does. It is a collection of resources all identified by URIs, which can be accessed and manipulated with HTTP request methods (e.g., POST, GET, PUT or DELETE). Furthermore, every message exchanged is self-descriptive as it always contains the Internet media type of the content, which is enough to describe how to process it.

These analyses can be divided into three categories: **Data gatherers**, **basic** and **composite software evolution analyses**.

**Data gatherers** work on raw data, extracting it from different software repositories, such as version control, issue tracking, mailing lists, plain source code, etc., and making it available to other SOFAS services analyses. This includes:

- Version history importers for CVS, SVN, GIT and Mercurial extract the version control information from a given repository.
- Issue tracking history importer for Bugzilla, Google Code, Trac, and SourceForge extract the issue tracking history from a given issue tracker.

**Basic software evolution analyses** exploit the data imported by one of these data gatherers to calculate all sort of software evolution information. For example:

- Version history and issue tracking metrics calculators, calculating several statistics from version and issue histories extracted by one of the aforementioned services.
- Code ownership calculators, detecting which developers "own" every file of a software project, based on its extracted version control history and using different heuristics.
- Change distribution calculator, calculating the distribution of changes between the developers in a given version control history using the Gini coefficient as proposed by Giger et al. [14].

**Composite software evolution analyses** aggregate data produced by other analyses to calculate more complex and domain spanning evolution information. This includes:

- Issue-revision linkers, reconstructing the links between issues of an issue tracking history and the revisions of a version control history that fixed them.
- Code Disharmonies detector, detecting the code disharmonies [22] existing in a software project using the code metrics extracted by the aforementioned calculators.
- Defect predictors, predicting defect prone entities using different heuristics, e.g. using code metrics or the bug cache algorithm that uses issue tracking and version histories of a project [21].

### *B. Software Analysis Ontologies*

To describe the data produced by our analyses, we have developed our own family of ontologies, called *SEON* (Software Engineering ONtologies).[1] An ontology is a formal description of the important concepts identified in the domain of discourse and their relationship to one another [17]. It provides a common vocabulary for a specific domain, which can be used to express the meta-data needed to capture the knowledge of the shared and reused data. Our ontologies, defined in OWL, are organized in a pyramidal structure. The top layer comprises ontologies describing general concepts,

[1]http://www.se-on.org

the attributes to describe them, and the relations between the concepts. The second-highest layer defines domain-spanning concepts. These concepts describe knowledge that spans a limited number of subdomains. The third layer is made up of ontologies describing different domains corresponding to important aspects of software evolution, e.g. issue and version management. At the bottom of the pyramid sit ontologies describing system-specific or language-dependent concepts (e.g. Java-specific constructs, SVN concepts, etc.). We refer to [36] and *SEON*'s official web page for more details on the approach and for a description of some of these ontologies.

### C. Software Analysis Broker

The Software Analysis Broker offers a single entry point to *SOFAS*' services. Through it, they are kept track of, classified in a registry, queried, monitored and coordinated. In this way, users do not have to interact directly with the plain services. The Broker is in turn made up of four main sub-components: the *Services Catalog*, a user interface, the *Services Composer*, and a series of management tools.

The *Services Catalog* stores and classifies all the registered analysis services so that a user can automatically discover services, invoke them, and fetch the results. We developed a software analysis taxonomy to systematically classify services. This taxonomy divides the possible analyses into three main categories: development process, underlying models, and source code. For more details we refer to the *SOFAS* website.[2]

The User Interface is the access point to the Software Analysis Broker. It consists of a web UI meant for human users and a series of RESTful service endpoints to be semi-automatically used by applications. Through the UI the user can easily browse through the *Services Catalog* to check for analyses offered, invoke them individually or, if necessary, combine them into workflows. The user interface offers an intuitive, high level way to do that, allowing the user to combine the services in a "pipe and filter" fashion. The system then translates this high-level workflow into an executable form using *SCoLa*, our WS-BPEL-based custom composition language. Furthermore, users can also pick from some already predefined combinations of analysis services provided as high level analyses workflows. On the other hand, applications directly submit analysis workflows written in *SCoLa* to the Broker using its REST API.

The *Services Composer* takes care of translating the workflows defined through the UI into actual, executable ones and execute them. Having the composition definition and the actual composition language decoupled, allows the user to focus on the creation of meaningful composite analyses without having to deal with the complexity and technicalities of the actual composition and orchestration. Moreover, calls to additional management services can be automatically weaved into a user defined workflow.

Management services are used to support a correct execution of analysis workflows. In fact, such workflows are usually made up of long running, asynchronous web services. Thus, in order to effectively handle them, every single service needs to be logged and monitored to check if is up and running, if it is in an erroneous state, if it completed a required operation, etc.

### D. Supporting Replication with SOFAS

A successful study replication can occur only if the following three aspects are fulfilled:

- **Availability of ground data.** The data on which the study is based should be easily and readily accessible in some form, preferably over the Internet.
- **Availability of the analysis itself.** The tools or scripts used to perform the study–which handle and analyze the ground data to produce the final results–should be publicly available and usable. If not, detailed instructions on how to perform the analysis, or the algorithms, should be provided.
- **Availability and traceability of results.** The results produced in the study should be available in the same way as the ground data. While not essential for the actual replication, it facilitates the verification of the results and claims of the original study, and the comparison with the results of the replicas.

Wuersch et al. [37] already made a case for using the Semantic Web to obviate the issue of availability and traceability of the results in mining software repositories analyses. The experience we gained in implementing different software analysis approaches shows that the combination of semantic web and REST, which is the core foundation of *SOFAS*, is a perfect fit in fulfilling the three aforementioned aspects and thus facilitating and supporting replicability.

Having analyses as RESTful web services with a uniform interface improves their accessibility. Users do not have to install or configure any tool, but just need to supply the analysis service with the necessary data. Moreover, services can also be easily integrated into custom user application and scripts. In fact, being RESTful, they can be called with simple HTTP methods, without the need of custom libraries or frameworks. For example, this could come down to a single command issued to a simple command-line tool like *cURL*[3]. With this solution, also the results are available online to all the other *SOFAS* users as they can be retrieved straight from the analysis itself. For example, given one of the services extracting the complete history of a version control repository reachable at http://habanero.ifi.uzh.ch/gitImporter, the results of specific analyses are available at http://habanero.ifi.uzh.ch/gitImporter/analyses/⟨analysis_name⟩. Depending on the user's needs, these results can be browsed online, downloaded in their RDF form or queried online using SPARQL. In this last case, the user has to encode the query in the URL, e.g., http://habanero.ifi.uzh.ch/gitImporter/analyses/⟨analysis_name⟩?query=⟨sparql_query⟩, or use the service web UI that has a form through which queries can be composed and ran.

The use of public semantic web ontologies to describe these analysis results facilitates their interpretation. In fact,

[2]https://seal.ifi.uzh.ch/sofas

[3]http://curl.haxx.se/

not only they describe in a clear way the domain of discourse, both semantically and syntactically, but they also come with a powerful, standard query language, SPARQL [29]. This means that researchers can also make the queries they used to extract and aggregate the final results public, to be re-used or verified straight away. Moreover, the semantic web concept of statements represented by triples of URIs enables us to link and query data that is stored on different services and build an internet-scale graph of analysis information. All *SOFAS* services, in addition to a standard SPARQL endpoint to query the analysis data, also expose a URI for every piece of information they extracted so that it can be dereferenced over the Internet. For example, let us consider one of *SOFAS*' issue-revision linker services which, given the issue tracking and version histories of a specific software project (extracted by other services), reconstruct the links between issues and the revisions that fixed them. Such service would produce triples like this:

```
bugzillaImporter/analyses/project_x/issues/124
se-on.org/ontologies/integration-history-issues/isFixedIn
gitImporter/analyses/project_x/changesets/1239
```

This triple states that the issue number 124 (as imported by service *bugzillaImporter*) of *project_x* was fixed during its revision 1239 (as imported by service *gitImporter*). From triples like this, client applications, as well as humans, can easily follow the links to access the actual resources involved or use them as an input for further SPARQL queries. From a very small initial piece of information, users can incrementally navigate the information space and expand their knowledge about the system being used.

*SOFAS* includes an entire family of analysis services called **data gatherers**, which we already introduced earlier and which sole purpose is to import raw, ground data in *SOFAS* to be used by other services, as well as human users. In this way, the ground data can be easily extracted, accessed and used as we just described for the other analysis services.

Finally, the ability to compose services into workflows allows users to execute more complex analyses building on the existing offering. This not only broadens the amount of existing studies and analyses that can be replicated, but can also be exploited in creating and executing novel ones.

## III. Replicability Evaluation

To show the applicability of our framework in replicating mining studies, we performed a complete review of all the studies published in the proceedings of the MSR conferences from 2004 to 2011 (a total of 8 editions). We then filtered out the non-experimental studies for which replication cannot be achieved. These papers, which actually account for the 51% (88 papers) of the total, typically propose new methods, analysis tools and frameworks, visualizations, case studies, etc. The remaining papers were then classified into 6 broad categories. These categories were subjectively constructed and were kept rather generic on purpose. In fact our goal was to identify the main "macro" empirical research areas and check whether our approach was particularly successful on any of those. We did not intend to perform a thorough and detailed review of the field.

9% of the 84 studies found, deal with the plain *mining of version history* data from version control systems for different purposes; to shed light on the development process [24], to better understand how developers work [38] and their dynamics [35]. The same number of studies addresses the issue of *bug prediction*, i.e., finding code entities that are most likely to be fixed or to be buggy based on different historical information. The work by Giger et al. [13] or by Sliwerski et al. [33] are two prime examples of this category. The two largest categories, both accounting for 22% of the total, are *defect analysis* and *social networks and team analysis*. The first one deals with the analysis of all sort of defects and flaws, e.g. clones [30], reported bugs [1], etc. and the exploitation of such data. The second one deals with the extraction and analysis of the social networks associated to a software project [5] and the team/development dynamics of it [25]. 20% of the studies belong to a more generic *historical mining* category, which encompasses studies exploiting all sorts of software development-related historical data for a wide range of applications. For example, deducing a developer's expertise based on its source code contributions to ease bug assignment [23] or finding license violations [18]. The remaining 15% deal with *change analysis*. That is, the analysis of the changes performed on the source code to uncover or extract more information about how the software changed. For example, studying how identifiers are renamed [6] or extracting and analyzing commonly occurring change patterns [20].

Every study was then inspected in detail to assess if and how it could be replicated using *SOFAS*. As a result, the studies were then qualified as **replicable**, **partially replicable** and **not replicable**. A study was considered *replicable if the same results can be replicated or if all the necessary data can be computed with the exception of their final aggregation or interpretation*. On the other hand, a study was considered *partially replicable if the results cannot be replicated out of the box, but the ground data from which they are derived can be computed*. Finally, a study was deemed *not replicable* if no, or very little, ground data could be computed.

Replication is usually divided in two main categories: exact and conceptual. Exact replication is when the procedures of the experiment are followed as closely as possible. Conceptual replication is when the experimental procedure is not followed strictly, but the same research questions or hypotheses are evaluated, e.g. different tools/algorithms are used or some of the variables are changed [32]. In this paper, we did not distinguish between exact and conceptual replication. A study was considered replicable whenever it could be replicated, either conceptually or exactly, using services currently available in *SOFAS*.

The results of our replicability assessment are reported in Table I. The level of replicability is spread evenly across the different research categories and there is no category for which *SOFAS* was significantly more or less successful. The only exceptional category is *History Mining*, as no full replicability

| Research category | Number of studies (in %) | Replicable studies | Partially replicable studies | Not replicable studies |
|---|---|---|---|---|
| Version History Mining | 8 (9%) | 4 | 0 | 4 |
| History Mining | 17 (20%) | 0 | 8 | 9 |
| Change Analysis | 13 (15%) | 5 | 6 | 2 |
| Social Networks and People | 19 (22%) | 6 | 5 | 8 |
| Defect Analysis | 19 (22%) | 8 | 6 | 5 |
| Bug Prediction | 8 (9%) | 2 | 2 | 4 |
|  | 88 (100%) | 25 (30%) | 27 (32%) | 32 (38%) |

TABLE I
REPLICABILITY OF MSR STUDIES FROM 2004–2011

could be achieved for it. This is most likely due to the fact that it is a quite broad category encompassing rather diverse studies each requiring their own specific analyses. If such analyses are not available in *SOFAS*, the studies cannot be yet replicated. However, simply adding services implementing these analyses would fix this shortfall. Nevertheless, 49% of such studies can already be partially replicated. This is because a lot of them are based on data extracted from software repositories such as version control, issue tracking, mailing lists or plain source code, which are well covered by *SOFAS*. The other categories are more specific and deal with common issues in software evolution analysis. That means that analyses dealing with such issues are likely to already exist in *SOFAS*. Moreover, studies in these categories are often similar to each other and can thus be conceptually replicated using the same analyses.

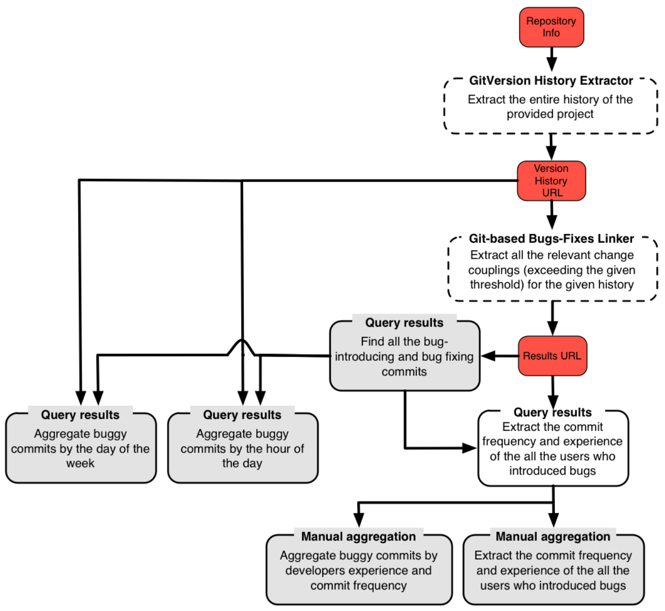
In the following section, we replicate one of the studies to better show and prove how *SOFAS* supports such replication.

## IV. Replication Study

Next, we present in detail the replication of the study by Eyolfson et al. [7] entitled: *"Do time of day and developer experience affect commit bugginess?"* published in MSR 2011. In particular, we (1) introduce the original study, (2) describe what we replicate, (3) show how it is done and extended with *SOFAS*, and (4) present and discuss the results of the (extended) replication.

The original study, performed by Eyolfson et al. [7], investigates the correlation between the bugginess of a commit and a series of factors: the time of day of the commit, the day of week of the commit, the experience and commit frequency of the committer. Such analysis is based solely on the history of a project extracted from its version control system. The authors consider as bug-introducing commit any commit for which there exists another commit explicitly fixing it later in time. To find them, they first detect all the bug fixing commits using a standard heuristic used in the field. That is, finding the ones that have specific keywords (e.g. “fix”, “fixed”, etc.) in their commit message. Buggy commits are then the ones that changed files that were involved in such fixes.

In their investigation, the authors studied the version control histories of the Linux kernel and PostgreSQL and uncovered four main findings. First, about a quarter of the commits in a project history introduce bugs. Second, the time of the day influences the introduction of bugs, as late night commits (submitted between midnight and 4 AM) are significantly more bug-prone and commits between 7 AM and noon are less buggy. Third, regularly committing developers (daily-committers) and more experienced committers introduce fewer bugs. Last, the influence of the day of the week on the commits bugginess is project dependent.



Repository Info
GitVersion History Extractor
Extract the entire history of the provided project
Version History URL
Git-based Bugs-Fixes Linker
Extract all the relevant change couplings (exceeding the given threshold) for the given history
Query results
Find all the bug-introducing and bug fixing commits
Results URL
Query results
Aggregate buggy commits by the day of the week
Query results
Aggregate buggy commits by the hour of the day
Query results
Extract the commit frequency and experience of the all the users who introduced bugs
Manual aggregation
Aggregate buggy commits by developers experience and commit frequency
Manual aggregation
Extract the commit frequency and experience of the all the users who introduced bugs

Fig. 1. Principle workflow used to replicate the targeted study

For our paper, we verify these four findings by fully replicating the original study. We also *extend the study* by testing if such findings hold for three other popular OSS projects: Apache HTTP, Subversion and VLC. The results of this replication are available online from the services used in the study.

### A. Replication Set-up

To replicate this study, we create and execute an analysis workflow that consists of the following main steps:

1) **Extract the complete project version control history.** This is accomplished using one of the version control history extractors currently registered in *SOFAS*. As of now, six of such services targeting different version control systems exist. They handle Git (two of them), CVS, Mercurial and SVN (two of them) repositories. In

this case, we used one of our Git services. The results are available at *habanero.ifi.uzh.ch/newGitImporter/<project name>* with project name being either *httpd*, *postgresql*, *linux*, *subversion* or *vlc*.[4]

2) **Find all the bug-introducing and bug-fixing commits (a.k.a. revisions) from such history.** This is also accomplished by one of the currently registered bug-revision linkers. These services extract such information from a project history extracted by one of the aforementioned version control services. Currently, five of them exist. Three of them are version control system independent but only find the bug-fixing commits. This is because this is the only information that can be recovered directly from version control history alone. Finding bug-introducing commits is, on the other hand, more complex to recover and the process to do so is version control system-dependent. The two linkers currently supporting this feature target Git and Mercurial-based repositories. These services are based on algorithms that are very similar to the ones used in the original study. The main difference lies in the heuristics used to detect bug-fixing commits. In fact, we use a larger set of terms to classify a commit as a bug fix based on its commit message.
3) **Extract the commit frequency and experience of all users who introduced bugs (calculated from the bug introduction date).** This is achieved by querying the data extracted in the first step with specific SPARQL queries.
4) **Aggregate the buggy commits by time of the day, day of the week, developer experience and commit frequency.** This is also achieved with SPARQL queries.
5) **Final results interpretation.** *SOFAS* simply supports the extraction and combination of analyses and data. The conclusions still have to be manually drawn by the initiators of such analyses, depending on their specific needs. The links found are available at *habanero.ifi.uzh.ch/bugFixesLinker/<project name>* (the project names are the same as in Step 1.

Figure 1 depicts the principle view of the workflow we used for the replication.

## B. Results

Next, we describe the results of this replication grouped into the original four main findings. All the projects were analyzed between July 1st and July 10th, 2012.

*1) Percentage of buggy commits:* Our replication confirms the results of the original study for both Linux and PostgreSQL. The slightly different values can be explained by the different heuristic used to detect bug fixes and the different analysis date (the projects were analyzed a year later than the original study to also provide some added value). Moreover, all the other analyzed projects exhibit similar values (22-28%), as shown in Table II. These results seem to indicate a trend worth investigating more in detail, with a larger body of projects.

[4]This data can be accessed in restricted read mode with username: REPLGUEST, password: REPLGUEST2013.

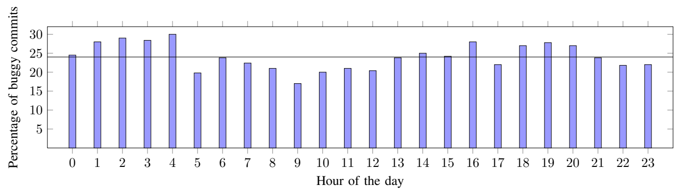


Fig. 2. Percentage of buggy commits versus time-of-day for PostgreSQL

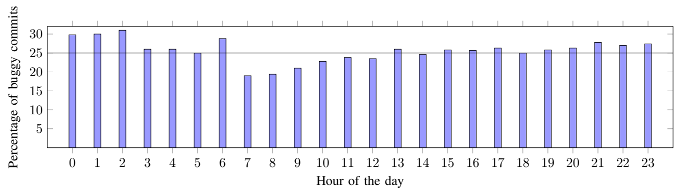


Fig. 3. Percentage of buggy commits versus time-of-day for Linux

*2) Influence of time of the day on commit bugginess:* Figures 2 to 6 show the correlation of the time of the day of a commit with its bugginess for the analyzed projects. The graph compares the time of the day of each commit on a 24-hour clock (in the committers local time) to the percentage of bug-introducing commits. The horizontal line in the graphs indicates the overall percentage of buggy commits for each project. Our replication confirmed the results of the original study for both original projects. Moreover, the analysis of the additional projects corroborates the finding that the amount of commits introducing a bug is particularly high between midnight and 4 AM and that then it tends to drop below average in the morning and/or early afternoon. However, these 'windows' of low bugginess greatly vary between projects. Furthermore, the projects' commit bugginess follows very different patterns, which do not allow any further generalization on the influence of the time of the day on the commit bugginess.

*3) Influence of developer on commit bugginess:* Figures 7 to 11 correlate author experience at time of commit (elapsed time between the author's first ever commit and the commit being examined) to the bugginess of the commit. Our replication confirms the original results that bugginess decreases

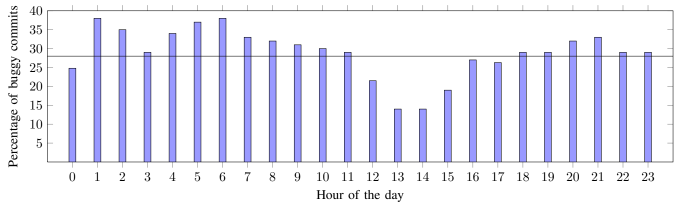


Fig. 4. Percentage of buggy commits versus time-of-day for Apache HTTP Server

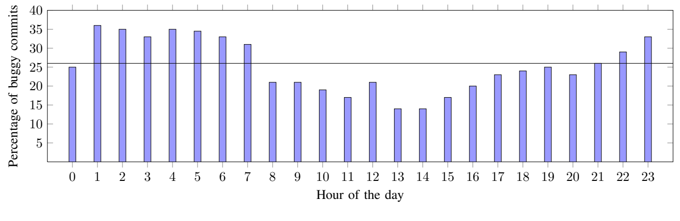


Fig. 5. Percentage of buggy commits versus time-of-day for Subversion

| | | # commits | # bug-introducing commits | # bug-fixing commits |
|---|---|---|---|---|
| Original Study | **Linux** | 268820 | 68010 (25%) | 68450 |
| | **PostgreSQL** | 38978 | 9354 (24%) | 8410 |
| Extended Study | **Apache Http Server** | 30701 | 8596 (28%) | 7802 |
| | **Subversion** | 47724 | 12408 (26%) | 10605 |
| | **VLC** | 47355 | 10418 (22%) | 10608 |

TABLE II
SMALL SUMMARY OF THE CHARACTERISTICS OF THE ANALYZED PROJECTS

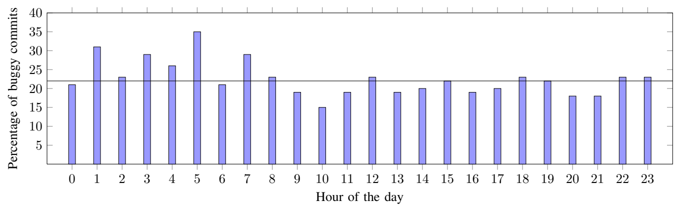


Fig. 6. Percentage of buggy commits versus time-of-day for VLC

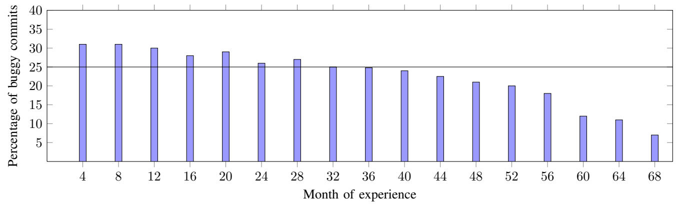


Fig. 7. Percentage of buggy commits versus author experience for Linux

with increased author experience for all the projects analyzed. In all projects, a drop in commit bugginess is evident with the increasing amount of time a developer has spent on a project. In four of the projects such drops happen between 32 and 40 months of experience, while in the case of PostgreSQL such a drop takes place much later, at 104 months of experience.

*4) Influence of day of the week on commit bugginess:* Figures 12 to 16 show the correlation between the day of the week with the commit bugginess on that day. As before, the solid horizontal line represents the overall commit bugginess of the project. Our results confirm the results of the original study. However, the additional projects' bugginess present

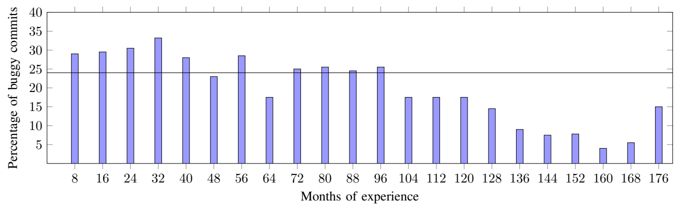


Fig. 8. Percentage of buggy commits versus author experience for PostgreSQL

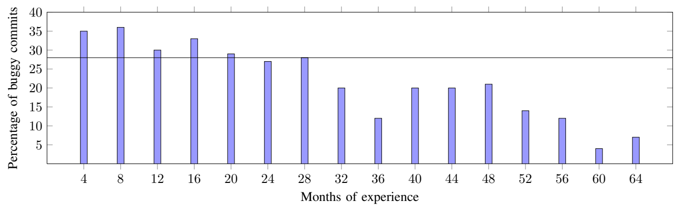


Fig. 9. Percentage of buggy commits versus author experience for Apache HTTP server

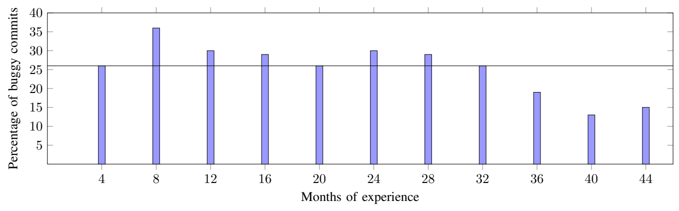


Fig. 10. Percentage of buggy commits versus author experience for Subversion

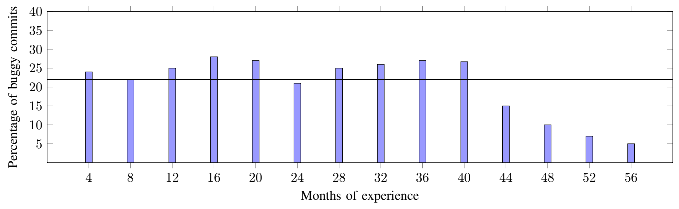


Fig. 11. Percentage of buggy commits versus author experience for VLC

different patterns. Apache HTTP server and Subversion tend to have two commit bugginess "phases": a higher than average one from Tuesday to Friday and a lower than average one from Saturday to Monday. On the other hand, the bug introduction in VLC is almost the opposite, as it is lower in the middle of the week (Wednesday to Friday). The analysis of these additional projects shows that the finding of the original project, that commits on different days of week have about the same bugginess, is not generalizable. Moreover it also shows that the results of a previous similar study [33] that showed the Friday was the day with the most buggy commits (based on the analysis of Mozilla and Eclipse) also cannot be generalized.

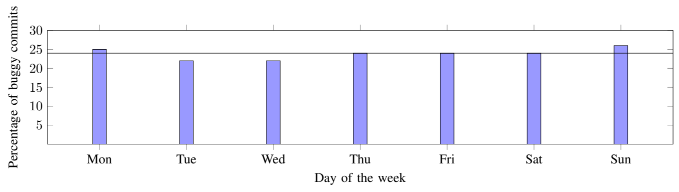


Fig. 12. Percentage of buggy commits versus day-of-week for PostgreSQL

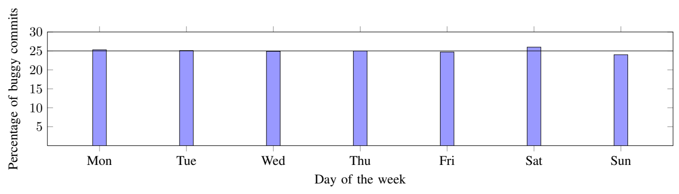


Fig. 13. Percentage of buggy commits versus day-of-week for Linux

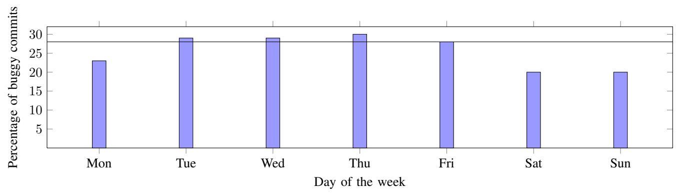


Fig. 14. Percentage of buggy commits versus day-of-week for all Apache HTTP server

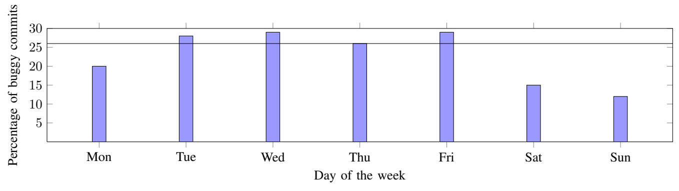


Fig. 15. Percentage of buggy commits versus day-of-week for Subversion

## V. Interpretation of the Results

In the interpretation of our results we consider the aspects of feasibility of replication, the scalability using more than the original projects for the study, and the extensibility with additional services to enrich the results. We discuss them in turn.

### *Feasibility*

We performed a feasibility assessment of using *SOFAS* as a platform to replicate mining studies. By exploiting *SOFAS* ability to compose a wide range of software evolution analyses into structured workflows we can replicate 30% of the analyzed studies and compute the ground data needed for another 32% of them. The studies we can replicate do not predominantly belong to any of the macro categories presented in Table I. However, they all use historical data extracted from different repositories (mainly issue trackers and version control systems) as their basis. This is because *SOFAS* original goal was the support of software evolution analysis, which cuts across the categories we identified.

### *Scalability*

To better show how replication with *SOFAS* works, we replicated a selected study using five software projects. However, the approach can scale up to hundreds of projects. Once the workflow introduced in Section IV-A is defined, it can then be automatically run with different project repositories as input. The only limitation for such a substantial corpus of projects would be the total execution time. In fact the analysis of a single long standing project such as the Apache HTTP server takes around 8 hours. However, the fact that the analysis services are deployed on industry-grade servers allows us to run several workflows in parallel without a noticeable performance degradation to alleviate this problem. Thus, even though it we did not specifically target performance improvement of software evolution analysis, the architectural features of our approach present serious advantages over similar approaches.

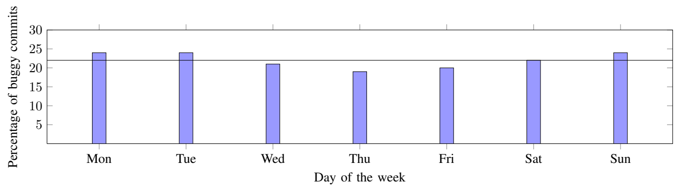


Fig. 16. Percentage of buggy commits versus day-of-week for VLC

### *Extensibility*

We only focused on the replication of existing studies. However, with *SOFAS*, experiments can also be extended and enriched with data produced by additional analyses, which were unaccounted for in the original study. Once the historical data has been imported into *SOFAS* and analyzed with the chosen analyses, the results–along with the ground data–can be fed to other services. They can also be used by third-party analyses and tools outside *SOFAS* with data from repositories such as PROMISE [8] or FLOSSmole [19]. For example, we could easily expand the study we replicated in Section IV by also taking into account code ownership measured using the Gini coefficient, as proposed by Giger et al. [14]. In that study, the authors calculated the Gini coefficient for every source code file in their dataset (the version control history of a selected group of Eclipse core plugins) based on the distribution of the changes to that particular file by all developers. They showed that, in the case of the selected projects, the more changes to a file are done by a few developers (high Gini coefficient) the less likely it will have bugs. The study we replicate would thus be extended to *"Do time of day, developer experience* ***and file ownership*** *affect commit bugginess?"*.

Since there is already a Gini coefficient calculator registered in *SOFAS*, this study extension would come down to feeding the previously extracted version control histories to it. This service computes the distribution of changes to source code files between the developers in a given version control history using the Gini coefficient. To actually find out whether file ownership computed with this method has an influence on commit bugginess, the user would have to manually query the bug-revision linker used and the Gini coefficient calculator to check whether the files in the previously found buggy commits exhibit high Gini coefficients. As of now, SPARQL does not support federated queries (i.e., writing and executing a query on different, distributed endpoints). This means that the user would have to query the two services separately and manually aggregate the results. However, a W3C recommendation about federated SPARQL queries is being currently worked on [28] such that in the future also this last step could be automated.

## VI. Related Work

The replication of studies is an essential task to expand and mature the current body of knowledge of any branch of science or technology. In particular, it is vital in gaining a deeper understanding on which results or observations hold under which conditions. However, such task is also intrinsically difficult, primarily because it is hard to reproduce a setting that is the same–or extremely similar–as the original study. This has led to the introduction of lab packages [3] (also known as replication packages) and data repositories. A lab package is a detailed description of an experiment, containing all the material needed to replicate an experiment or at least simplify the replication. Data repositories, host publicly available software engineering data sets containing a wide range of data: error data, failure data, software metrics, analysis tools, etc. Such repositories are mostly used as commonly accepted benchmarks to validate models and studies. One of the most prominent example is PROMISE [8].

Such systematic approaches are still lacking in software evolution analysis and mining software repositories, as reported by Robles [31]. The main reason behind this lack of replication being the unavailability of the tools, scripts and instructions necessary to run the study (80% of the cases) and the actual results data set.

Similarly to the PROMISE repository, researchers have established online software evolution data repositories. Flossmole [19] contains metadata about more than 500,000 open source projects extracted from the major code source forges (sourceforge, github, freshmeat, etc.). The extracted data focuses on more high-level development information and dynamics and is offered "as is" (no actual analysis is performed). The Ultimate Debian Database [27] follows a similar approach but only for the Debian Linux distribution and all its binary packages. However, it focuses on extracting and presenting more system specific information (package popularity, history of packages upload, etc.). Mockus [26], on the other hand, collected and indexed the version control history and the actual source code of a large sample of software projects from the most notable forges. With all these repositories we share the concept of having diverse, automatically retrieved, software history data easily available on-line. However, with *SOFAS* is possible to proactively fetch such data for new projects, while these repositories handle only a fixed, pre-defined set of projects.

The lack of proactiveness of data repositories can be overcome by tools and platforms combining a wide range of software analyses. Systems such as Kenyon [4] or Evolizer [9] automatically extract the version control history, bug history of a software and run additional analyses based on that. Moreover, they have means to use and interpret such data (e.g. visualizations, querying interfaces, etc.). These tools allow to easily replicate studies that are based exactly on the analyses provided. No other replication is possible as the supported analyses–and their combinations–are created beforehand and hardcoded into the tools. Furthermore, none of them makes the results easily available from outside of the tool, a crucial requirement for successful replication.

Gasser et al. [10] point out the need for a sharable research infrastructure and collections of data under common access points and frameworks. *SOFAS*, along with FOSSology [15] and Alitheia Core [16] are systems devised to address that need. FOSSology is a framework to analyze source code with different, custom analyses (called agents) that can be created by users to fulfill their specific needs. The framework itself is just in charge of extracting the source code from a given repository which will then be analyzed by these analysis agents supplied by the framework's users. To the best of our knowledge, for now only an agent that detects the code license exists. Alitheia Core is the approach closest to *SOFAS*. The major differences lie in the implementation (plug-in vs. service based architecture), the analyses currently offered and how they can be combined. In contrast with *SOFAS*, Alitheia does not allow to freely combine analyses. Data plugins retrieve and process data from version control systems, issue trackers and mailing lists, which can then used by metrics plugins to calculate additional knowledge. However, these plugins cannot be further combined to provide more complex metrics nor their data be used by other plugins.

The only analogous study existing in the literature is by Tappolet et al. [34]. In this study, the authors showed that if the data used by the MSR empirical studies were available in their software evolution ontology (EvoOnt), 75% of them could be reproduced with at most two SPARQL queries. However, no concrete study replication was performed. In fact, their goal was to demonstrate the potential of the inherent capabilities of semantic web ontologies in supporting software evolution research and overcome some of its most significant obstacles. We share with them the concept of representing software evolution data with ontologies and the opinion that they are extremely beneficial in the representation, sharing and combination of such data. However, we focus on the replication of actual MSR studies and in proving how that can be addressed by a platform like *SOFAS*.

## VII. Conclusions

In this paper, we demonstrated how our *SOFAS* platform can be used to effectively replicate mining studies previously presented at the MSR conference. To identify these studies, we performed a comprehensive literature review of the papers published in the proceedings of any of the MSR conferences from 2004 to 2011. All the respective studies found were then classified into 6 different categories. For each of them, we manually assessed whether it can be replicated with a combination of the software evolution analysis services currently available in *SOFAS*. As a result, we found that 30% of such studies could be replicated fully, while and additional 32% could be partially replicated. A study was considered partially replicable, if we could not replicate its results straight out of the box, but we could calculate all the ground data from which they were derived. To support our claim of replicability with *SOFAS*, we presented in detail the replication of one of the published studies.

The amount of studies that can be fully replicated amounts to about one third. However, combined with the partially replicated ones, we can simplify the replication of up to 62% of the currently existing studies. We are convinced that this is a promising results, as our main goal was to prove the applicability of our platform in tackling an essential issue of study replication in the software evolution community, namely uniform availability of analysis and results.

The main limitation of our approach is the breadth of replication support by the analyses currently offered by *SOFAS*. Any further improvement would require the addition of new services. In the future, we plan to add further services and analyze a wide corpus of OSS projects with *SOFAS* to create an online data repository freely available to researchers to use as benchmark and for their analyses. We hope that the availability of such data and the replication framework will spark interest to tackle replicability in a more systematic and standardized way.

## References

[1] J. Anvik and G. C. Murphy. Determining Implementation Expertise from Bug Reports. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007.

[2] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: a Search Engine for Open Source Code Supporting Structure-Based Search. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA '06)*, pages 681–682, 2006.

[3] V. Basili, F. Shull, and F. Lanubile. Building Knowledge Through Families of Experiments. *IEEE Transactions on Software Engineering*, 25(4):456 –473, jul/aug 1999.

[4] J. Bevan, E. J. Whitehead, Jr., S. Kim, and M. Godfrey. Facilitating Software Evolution Research with Kenyon. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 177–186, 2005.

[5] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining Email Social Networks. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 137–143, 2006.

[6] L. M. Eshkevari, V. Arnaoudova, M. Di Penta, R. Oliveto, Y.-G. Guéhéneuc, and G. Antoniol. An Exploratory Study of Identifier Renamings. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 33–42, 2011.

[7] J. Eyolfson, L. Tan, and P. Lam. Do Time of Day and Developer Experience Affect Commit Bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162, 2011.

[8] T. M. G. Boetticher and T. Ostrand. PROMISE Repository of empirical software engineering data. West Virginia University, Department of Computer Science, http://promisedata.org/repository 2007.

[9] H. C. Gall, B. Fluri, and M. Pinzger. Change Analysis with Evolizer and ChangeDistiller. *IEEE Software*, 26(1):26–33, January/February 2009.

[10] L. Gasser, G. Ripoche, and R. J. Sandusky. Research Infrastructure for Empirical Science of F/OSS. In *Proceedings of the International Workshop on Mining Software Repositories*, 2004.

[11] G. Ghezzi and H. Gall. A Framework for Semi-Automated Software Evolution Analysis Composition. *Automated Software Engineering (to be published)*, 2013.

[12] G. Ghezzi and H. C. Gall. SOFAS: A Lightweight Architecture for Software Analysis as a Service. In *Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture*, 2011.

[13] E. Giger, M. Pinzger, and H. C. Gall. Comparing fine-grained source code changes and code churn for bug prediction. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 83–92, 2011.

[14] E. Giger, M. Pinzger, and H. C. Gall. Using the gini coefficient for bug prediction in eclipse. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, pages 51–55, 2011.

[15] R. Gobeille. The FOSSology project. In *Proceedings of the 5th IEEE International Working Conference on Mining Software Repositories*, pages 47–50, 2008.

[16] G. Gousios and D. Spinellis. A Platform for Software Engineering Research. In *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories*, pages 31 –40, 2009.

[17] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199 – 220, 1993.

[18] A. Hemel, K. T. Kalleberg, R. Vermaas, and E. Dolstra. Finding software license violations through binary code clone detection. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 63–72, 2011.

[19] J. Howison, M. Conklin, and K. Crowston. FLOSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. *International Journal of Information Technology and Web Engineering*, pages 17–26, 2006.

[20] S. Kim, E. J. Whitehead, and J. Bevan. Analysis of signature change patterns. In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, pages 1–5, 2005.

[21] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller. Predicting faults from cached history. In *Proceedings of the 29th international conference on Software Engineering*, pages 489–498, 2007.

[22] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc., 2005.

[23] D. Matter, A. Kuhn, and O. Nierstrasz. Assigning bug reports using a vocabulary-based expertise model of developers. In *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories*, pages 131–140, 2009.

[24] S. McIntosh, B. Adams, and A. Hassan. The evolution of ANT build systems. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, pages 42 –51, 2010.

[25] S. Minto and G. C. Murphy. Recommending Emergent Teams. In *Proceedings of the 4th International Workshop on Mining Software Repositories*, 2007.

[26] A. Mockus. Amassing and indexing a large sample of version control systems: Towards the census of public source code history. In *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories*, pages 11–20, 2009.

[27] L. Nussbaum and S. Zacchiroli. The Ultimate Debian Database: Consolidating bazaar metadata for Quality Assurance and data mining. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, pages 52 –61, 2010.

[28] E. Prud'hommeaux and C. Buil-Aranda. SPARQL 1.1 Federated Query. W3C Proposed Recommendation, 8 November 2012. http://www.w3.org/TR/sparql11-federated-query/.

[29] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 15 January 2008. http://www.w3.org/TR/rdf-sparql-query/.

[30] F. Rahman, C. Bird, and P. Devanbu. Clones: What is that smell? In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, pages 72 –81, 2010.

[31] G. Robles. Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, pages 171–180, 2010.

[32] F. Shull, J. C. Carver, S. Vegas, and N. J. Juzgado. The role of replications in Empirical Software Engineering. *Empirical Software Engineering*, 13(2):211–218, 2008.

[33] J. Sliwerski, T. Zimmermann, and A. Zeller. When Do Changes Induce Fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, 2005.

[34] J. Tappolet, C. Kiefer, and A. Bernstein. Semantic web enabled software analysis. *Web Semantics*, 8(2-3):225–240, 2010.

[35] C. C. Williams and J. W. Spacco. Branching and merging in the repository. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 19–22, 2008.

[36] M. Würsch, G. Ghezzi, M. Hert, G. Reif, and H. Gall. SEON: A Pyramid of Ontologies for Software Evolution and its Applications. *Computing*, pages 1–31, 2012.

[37] D. S. Würsch M., Reif G. and G. H. C. Fostering synergies - how semantic web technology could influence software repositories. In *2nd International Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, 2010.

[38] K. W. Ying Liu, Eleni Stroulia and D. German. Using CVS Historical Information to Understand How Students Develop Software. In *Proceedings of the 2004 International Workshop on Mining Software Repositories*, 2004.