**Empirical Software Engineering manuscript No.**
(will be inserted by the editor)

# Addressing Problems with Replicability and Validity of Repository Mining Studies Through a Smart Data Platform

**Fabian Trautsch · Steffen Herbold · Philip Makedonski · Jens Grabowski**

*The final publication is available at Springer via https://doi.org/10.1007/s10664-017-9537-x*

Received: date / Accepted: date

**Abstract** The usage of empirical methods has grown common in software engineering. This trend spawned hundreds of publications, whose results are helping to understand and improve the software development process. Due to the data-driven nature of this venue of investigation, we identified several problems within the current state-of-the-art that pose a threat to the replicability and validity of approaches. The heavy re-use of data sets in many studies may invalidate the results in case problems with the data itself are identified. Moreover, for many studies data and/or the implementations are not available, which hinders a replication of the results and, thereby, decreases the comparability between studies. Furthermore, many studies use small data sets, which comprise of less than 10 projects. This poses a threat especially to the external validity of these studies. Even if all information about the studies is available, the diversity of the used tooling can make their replication even then very hard. Within this paper, we discuss a potential solution to these problems through a cloud-based platform that integrates data collection and analytics. We created SmartSHARK, which implements our approach. Using SmartSHARK, we collected data from several projects and created different

Fabian Trautsch
Institute of Computer Science, Georg-August-Universtität Göttingen, Germany
E-mail: trautsch@cs.uni-goettingen.de

Steffen Herbold
Institute of Computer Science, Georg-August-Universtität Göttingen, Germany
E-mail: herbold@cs.uni-goettingen.de

Philip Makedonski
Institute of Computer Science, Georg-August-Universtität Göttingen, Germany
E-mail: makedonski@cs.uni-goettingen.de

Jens Grabowski
Institute of Computer Science, Georg-August-Universtität Göttingen, Germany
E-mail: grabowski@cs.uni-goettingen.de

analytic examples. Within this article, we present SmartSHARK and discuss our experiences regarding the use of it and the mentioned problems. Additionally, we show how we have addressed the issues that we have identified during our work with SmartSHARK.

**Keywords** Software Mining · Software Analytics · Smart Data Platform · Replicability · Validity

## 1 Introduction

The usage of empirical methods, (e.g., controlled experiments, case studies), has grown common in software engineering. They were used by only 2% of papers published at major journals and conferences between 1993 and 2002 (Sjøberg et al 2005) and the usage raised to 94% at the three major venues and conferences (International Conference on Software Engineering (ICSE) (2012, 2013), European Software Engineering Conference/Symposium on the Foundations of Software Engineering (2011 to 2013), and Empirical Software Engineering (2011 to 2013)) (Siegmund et al 2015a). This new trend spawned hundreds of publications, which focus on, e.g., data collection from software repositories (Draisbach and Naumann 2010; Dyer et al 2015), data analysis of the collected data (Di Sorbo et al 2015; Giger et al 2010), or the development of new methods that could help in supporting the software development process (He et al 2015; Jorgensen and Shepperd 2007).

A recent study by Siegmund et al (2015a) highlights two different problems that the current empirical software engineering research has: replication studies are important, but there are very few of them; the validity of studies can often not be assessed or checked with new data. There are two different kinds of replication: exact, where the original procedures of the experiment are followed as closely as possible and conceptual, where the same research question is evaluated by using a different experimental procedure or different data (Shull et al 2008). To enable researchers to perform either of the replication types the following needs to be available: data from the original study that should be replicated, used methods, and used algorithms. Very few papers provide all the above mentioned information, as González-Barahona and Robles (2012) point out in their study. The problem is also present in the field of Mining Software Repositories (MSR) (Robles 2010).

The problem of insufficient replications and the validity problems are tightly connected. As root cause for both of them, 5 different problems can be identified:

1. **Heavy re-use of data sets.** While the re-use of the same data is important for the comparability of results, too heavy re-use without also considering new data poses a threat to the external validity of the results. One example for this is the heavy use of the NASA defect data for software defect prediction, which is used in at least 58 different studies on

software defect prediction (Hall et al 2012). This allows good comparability between results, but poses a threat to the validity, as, e.g., Shepperd et al (2013) point out problems with the quality of the data which could, thereby, threaten the validity of 58 studies at once.

2. **Non-availability of data sets.** The opposite of the above mentioned problem is that the data used for a study is not available for other researchers. In this case, a replication of the study is not possible, which makes it hard to compare results between studies with different data sets.
3. **Non-availability of implementations.** The implementations of approaches are not always publicly available as open source. Furthermore, visualizations that summarize results, give an overview of the data, or are used to study certain aspects of a software system (e.g., (Van Rysselberghe and Demeyer 2004)) are even less often available for researchers. But these visualizations are important, as they make the results and the data better comprehensible for humans (Thomas and Cook 2006).
   For a complex research proposal, a re-implementation of approaches and/or visualizations can require a high amount of resources. Even for a simple approach, the re-implementations may differ from the initial implementation due to different interpretations of the paper contents where the original implementation was described. Moreover, there are instances in which the description of an approach does not provide all necessary details for a one-to-one re-implementation.
4. **Small data sets.** Due to a lack of readily preprocessed public data, some studies use rather small amounts of data, often of less than ten software projects. Moreover, only very few studies use larger data sets with more than 100 software projects. This is a threat especially to the external validity of these studies.
5. **Diverse tooling.** For most studies, developers create their own tool environment based on their preferences. These environments are often based on existing solutions for data analytics, e.g., R[1] for data mining or WEKA (Hall et al 2009) for machine learning. These solutions range from prototypes that can only be applied to the data used in the case study but nothing else, to close-to-industrial level solutions that can be applied in a broad range of settings. However, these solutions are usually incompatible to each other (e.g., solutions for Linux or for Windows). Hence, even if all data, implementations, etc. required for a replication are available, the diversity of the required tooling puts a heavy burden on the replication process, especially if multiple results need to be replicated.

Within this paper, we want to investigate how we can address these problems. Our solution to the above mentioned problems, which hinder the replication and therefore the assessment of the validity of an approach is the creation of a smart data platform, which combines data collection and data analysis. The integration of the data analysis into the platform gives researchers a common ground, on which they can build their implementations. Furthermore,

[1] `https://www.r-project.org/`

we want to enable researchers to directly share their used data as well as the used implementations. This would remove the burden (especially for novice researchers) to set up a complete environment to replicate studies or perform conceptual replications. Furthermore, such a platform would help researchers, who do not have a background in MSR, to analyze the data. This is especially important, as more and more researchers from other disciplines use data from, e.g., software repositories to perform their studies.

That such a platform is possible in principle was demonstrated by Microsoft, which developed a tool for internal use called CODEMINE by Czerwonka et al (2013). However, within the current state of the art we found no publicly available platform, which is similar to CODEMINE in terms of its capabilities. Hence, we wanted to create a platform for researchers who work on publicly available data, which is based on publicly available technologies. To this aim, we created SmartSHARK. With SmartSHARK we want to evaluate the feasibility of such a smart data platform in terms of the following factors: 1) capability to perform different analytic tasks; 2) potential to address the problems discussed above; and 3) provide lessons learned for researchers interested in such a platform. To this aim, we conducted an experience study to evaluate the platform to the above mentioned criteria. Furthermore, we considered the usability of such a platform, because this is a central aspect for the acceptance by other researchers.

In summary, the contributions of our paper are the following.

- We analyzed the current state of practice with respect to the five problems mentioned above, based on the proceedings of the ICSE 2015, the International Conference on Mining Software Repositories (MSR) 2015, the International Symposium on Empirical Software Engineering and Measurement (ESEM) 2015, and the Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE) 2015.
- We implemented the SmartSHARK platform that combines automated data collection from different sources with a web fronted from which Apache Spark jobs can be submitted to the platform to perform software analytics on the collected data.
- We analyzed, based on an experience report, if and how SmartSHARK can be used to resolve problems mentioned above.
- We evaluated the lessons learned from CODEMINE (Czerwonka et al 2013) from a research perspective.

This work is based on our previous work (Trautsch et al 2016) which was published in the proceedings of the MSR 2016. Based on the feedback from the community we received during the conference and our continuing work on the project, this article provides the following extensions in comparison to the conference paper.

- We added the analysis of the current state of practice.
- We extended our focus include to replication studies for MSR research.

- We improved the design and implementation of the platform to address issues we raised ourselves during the initial presentation of our work, as well as community feedback.
- We added a new visualization, which is a replication of a visualization from the paper by Bird et al (2006).
- We added a table that shows the need of data storage for our newly developed plugins if they are executed on the analyzed projects.

### 1.1 Paper Organization

The remainder of this paper is structured as follows. In Section 2 we give an overview of related work. Then, we present the results of our analysis of current papers in Section 3. Afterwards, in Section 4, we describe the SmartSHARK platform, which was the basis for our experience report presented in Section 5. In Section 6 we discuss how SmartSHARK can contribute to resolve the five problems mentioned above. Then, we describe how we have addressed the problems described in the experience report in an updated version of SmartSHARK in Section 7. Afterwards, we present threats to validity of this paper in Section 8. Finally, we conclude our paper in Section 9.

## 2 Related Work

The research on software data collection, as well as software analytics, cover many different directions and aspects. Most related work only considers either the data collection (e.g., Bevan et al 2005; Čubranić et al 2005; German 2004), potentially including the provision of data sets or queryable databases (e.g., Howison et al 2005; Gousios and Spinellis 2012; Gousios et al 2014; Di Ruscio et al 2015, the GitHub Archive[2]) or the analytical aspect (e.g., defect prediction (Catal and Diri 2009), effort prediction (Jorgensen and Shepperd 2007), developer social networks (Jermakovics et al 2011)). Only very few studies deal with both aspects. Furthermore, there is no approach which focus on tackling the problems regarding the replication and validity of MSR case studies.

A proprietary and not publicly available approach for building a software analytics platform with integrated data collection is the Microsoft internal CODEMINE by Czerwonka et al (2013). With our work, we tried to build a platform with features similar to CODEMINE with publicly available and cost-free tools. CODEMINE is designed in a way, that more than one CODEMINE instance can run at the same time by different product teams. These instances all have a common core. In CODEMINE, the data is collected by different data loaders and stored in a data store. The stored data is then exposed via different Application Programming Interfaces (APIs) on which analytics and tools can be defined. Due to its proprietary nature, no details on the implementation of the platform are available. However, Czerwonka et al. provide lessons learned

[2] https://www.githubarchive.org/

for building tools similar to CODEMINE. Our SmartSHARK platform uses the same general structure as CODEMINE and these lessons, provided by Czerwonka et al., were vital for the development of SmartSHARK. Details on how the lessons learned influenced SmartSHARK are discussed in Section 6.2. Due to the focus of CODEMINE on supporting the product teams at Microsoft only Microsoft internal data is collected. Therefore, considerations on the collection and analysis of publicly available data are out of scope of their work. Conversely, our paper focuses on the development of a platform that can be used by researchers who work with publicly available data and the impact such a platform can have on the replication and validity of studies.

Dyer et al (2015, 2013) developed *Boa*, a domain-specific language and infrastructure for analyzing ultra-large-scale software repositories. Boa is a query system, where complicated queries can be executed to extract information from previously cached repositories using a distributed MapReduce query language. Boa programs compile down to different MapReduce jobs, which are then executed on an Apache Hadoop[3] cluster. The key difference between Boa and SmartSHARK is the type of analytics that is supported. While Boa provides Abstract Syntax Trees (ASTs) of the projects, it does not directly enable deep analytics. Data like, software metrics, social metrics, etc. would have to be calculated manually for each project by the researchers, which is very time consuming and, thereby, probably lead to performance problems of the analytic approaches. Additionally, previously developed programs could not be re-used. Furthermore, Boa heavily uses MapReduce for its queries, whereas SmartSHARK uses Apache Spark (Zaharia et al 2010, 2012). It is reported that *Hadoop* MapReduce is inefficient for interactive analytics (Zaharia et al 2010) as they are intended to be performed with Boa. Finally, we evaluated different aspects of such a platform. The developers of Boa focused on how to enable large-scale analytics, while we additionally consider how factors related to the validity and replication of approaches can be improved and how our platform contributes to the five mentioned problems. However, Boa can also contribute in solving the problems mentioned in the introduction, e.g., through collecting data from repositories (see: problem 1, 2, and 4). But, due to the different focus of Boa, it currently does not provide the sharing of implementations, although it should be possible to extend Boa in such a way. Furthermore, regarding the diversity of tooling, Boa solves only part of the problem. The Boa domain-specific language harmonizes the queries and the analysis that are directly performed on the Boa database. However, in case the results of the Boa queries are further processed, e.g., with an additional machine learning or visualization tool, users may still use arbitrary technologies on the Boa output.

Gousios and Spinellis (2009) developed the *Alitheia Core* platform to perform "large-scale software quality evaluation studies" (Gousios and Spinellis 2009). The architecture of Alitheia Core is divided into three different layers: (1) result presentation, (2) system core, and (3) data mirroring, storage, and

[3] `https://hadoop.apache.org/`

retrieval. The first layer is implemented via a web front-end. The second layer includes a job scheduler and cluster service as well as other services, which are connected via an Open Services Gateway Initiative (OSGi) interface Alliance (2007). Basically, the OSGi specification describes a hardware-independent platform that makes it easier for developers to modularize and maintain applications and services. The third layer is responsible for the storage and retrieval of the collected data. This platform provides a metrics plug-in system, which enables researchers to implement their own plug-ins to calculate metrics out of the collected data. From a structural perspective and general idea, Alitheia Core is similar to our approach: we also have a data collection part, an analytic core, and a web front-end. However, our platform is designed around big data technologies to allow scalable analytics. Moreover, our platform is deployed in a cloud, which allows elastic scaling of resources. Finally, our analytic core, using Apache Spark, is more powerful in terms of computational capabilities due to the usage of an Apache Hadoop cluster for job execution and provides powerful algorithms for data analytics through Apache Spark's Mllib[4] and GraphX[5] libraries. Moreover, same as Boa, the authors did not consider how the platform can help in tackling the five problems mentioned in Section 1.

There are also commercial approaches that try to combine software data collection with software analytics. Bitergia[6] offers several packages, which differ in the level of analytic capabilities. However, the analytics provided by Bitergia are on the level of Business Intelligence (BI), i.e., reporting the history of the repositories. For example, they provide dashboards that display the number of performed commits or how many authors are active in the analyzed project. Similar to Bitergia is the OpenHub project[7]. OpenHub is an open platform, where every user can add or edit projects. The platform calculates different statistics for added projects, which are similar to the statistics calculated by Bitergia and also on the BI level. In comparison to SmartSHARK, Bitergia and OpenHub do not support deep analytics or predictions for the future of projects. Additionally, users of Bitergia and OpenHub are not allowed to create their own analytics.

## 3 Current State of Practice

To show that the five problems mentioned in Section 1 are still topical, we looked at the current state of practice by analyzing papers from the ICSE 2015, MSR 2015, the ESEM 2015 and the ESEC/FSE 2015 in respect to them.

We only included papers, which have done a case study in the field of MSR, were published on the main research track, and where the case study does not only includes a study with human participation (e.g., surveys). We

[4] http://spark.apache.org/docs/latest/mllib-guide.html
[5] http://spark.apache.org/graphx/
[6] http://bitergia.com/
[7] https://www.openhub.net/

determined, if a paper has one of the problems mentioned in Section 1 in the following way:

- Heavy re-use of data sets: the paper used a previously published data set (e.g., from the PROMISE Menzies et al (2015) repository).
- Non-availability of data sets: the paper does not provide all data (results, raw, and processed data) or the data is not publicly available.
- Non-availability of implementations: the paper does not provide all implementations (used tools, scripts) or made them publicly available.
- Small data sets: the paper included less than 10 projects in their data set. However, the problem of small data sets heavily depends on the research question and the used data. Nevertheless, this evaluation is useful to have a first look at the generalization of approaches, as it is more concerned with the assessment of the external validity of them.
- Diverse tooling: the paper does not provide or describe details of the execution environment. We checked this only for papers, where the implementation is publicly available, as otherwise, we would not be able to execute the implementation anyhow.

Tables 1 and 2 report the results of our inspection. Table 1 shows the number of papers for the four conferences, which were included in our inspection, and the number of papers that had the problems mentioned in Section 1. For the first problem (heavy re-use), we found that papers concerned with defect prediction often re-use data sets, e.g., the problematic reuse of the pre-processed NASA defect data (Shepperd et al 2013). Six papers used an established data set and produced their own additionally to it.

We found that over 54% did not (or only partly) publish their data sets (problem 2). Furthermore, we found that sometime the links, where the data sets should have been published, are dead links, even though the conferences have taken place only a year ago. For six papers, the data sets are only partly available (e.g., only the results as CSV-file).

In over 65% of all analyzed papers the authors did not publish their used implementations (problem 3). The authors did not mention which tools they used or they did not make their own tools publicly available. Moreover, we had the same problem as mentioned above: links that should redirect to the tools or direct download links are no longer working.

In our inspection we saw that nearly 50% of the papers that we included in our analysis use only a small data set. Two papers were concerned with user profiles or user sessions and one with Topcoder tasks[8] and not projects. These three papers were excluded from the analysis of the last problem, as we were not able to see from how many different projects these data origins.

Table 2 highlights the problem of diverse tooling. We included all papers, where the implementation is freely available and the environment is described in the paper or in supplementary material (e.g., in the replication package). Furthermore, we added the description of the environment as we found it in

[8] https://www.topcoder.com/

**Table 1** Current state of practice. The numbers in the table corresponds to the problems mentioned in Section 1.

| **Conference** | **#Papers** | **(1)** | **(2)** | **(3)** | **(4)** |
|---|---|---|---|---|---|
| ICSE 2015 | 15 | 3 | 9 | 10 (1 partly) | 7 |
| MSR 2015 | 23 | 5 (3 partly) | 13 (2 partly) | 15 (1 partly) | 11 |
| ESEM 2015 | 12 | 3 | 7 | 9 | 5 |
| ESEC/FSE 2015 | 47 | 21 (3 partly) | 18 (4 partly) | 28 | 23 |
| **Overall** | 97 | 32 (6 partly) | 47 (6 partly) | 62 (2 partly) | 46 |

the material. Hence, some descriptions are more elaborate (e.g., which version of Java or Python is used), whereas others are more generic. Overall this table supports our hypothesis that even if the implementations are available, the diversity of the required tooling puts a heavy burden on the replication process.

## 4 The SmartSHARK Platform

In order to get insights of how a CODEMINE-like tool can be made available for all researchers, as well as analyze the features that such a platform should offer from a research perspective, we created the platform SmartSHARK[9]. Figure 1 gives a logical overview of the platform, which is independent of the underlying infrastructure. SmartSHARK is designed as a cloud-based data platform. This means that data is shared between all users of the platform. The process of analyzing a software project can be divided into two steps: 1) Extract, Transform, Load (ETL) of the project data and 2) writing and running the analytic program. The ETL is implemented as an automated process that loads extracted project data in a MongoDB. Hence, researchers can focus on writing their analytic programs that utilizes the previously stored data from the MongoDB.

In this section, we present our first prototype of SmartSHARK. This information is needed, as the reader needs to know the version on which our experience report in Section 5 and the subsequent discussion in Section 6 is based on. Changes that we made to cope with the limitations and problems are discussed in Section 7.

### 4.1 Data ETL Process

Projects, from which the data should be extracted, transformed and loaded, are added via the web front-end of SmartSHARK. SmartSHARK only requires (1) the URL of the Version Control System (VCS), (2) the programming language, and (if applicable) (3) the URL of the mailing list of the project. After this

[9] The complete source code as well as deployment scripts are available in our public SVN: `http://trex.informatik.uni-goettingen.de/svn/smartshark/`. A running instance is located at the following URL: `http://smartshark.informatik.uni-goettingen.de`.

**Table 2** Papers and their environment that must be built before executing their approach (as described in the paper or supplementary material).

| Reference | Environment, as described |
|---|---|
| (Yang et al 2015) | APKTools, Python 2, Java SDK 1.7, Android SDK tools, Android SDK platform-tools |
| (Zhu et al 2015) | WEKA, C#, Roslyn, Visual Studio 2012+ |
| (Avdiienko et al 2015) | R, Python, Flowdroid |
| (Joblin et al 2015) | Python, R, MySQL, Perl, CPPStats, Nginx, Nodejs |
| (Nanz and Furia 2015) | Python, R, Cloc |
| (Lin and Whitehead 2015) | CVSAnalY, Python, ChangeDistiller, SimPy, NetworkX, ply, Java |
| (Linares-Vásquez et al 2015b) | Python, Apktool, Dex2jar, Procyon, srcML, MITLM |
| (Coelho et al 2015) | MongoDB, Java |
| (Tao and Kim 2015) | IBM Watson Libraries (WALA), Python difflib |
| (Claes et al 2015) | Ocaml |
| (Le et al 2015) | Java, Python, git, svn |
| (Kouroshfar et al 2015) | Class Dependency Analyzer, Java, Bunch, ACDC, ArchDRH |
| (Gupta et al 2015) | Jflex, Java |
| (Gallaba et al 2015) | Node.js |
| (Cavalcanti et al 2015) | Groovy, Neo4j, gitminer, Java |
| (Bang et al 2015) | Java SDK 7, Mathematica |
| (Safi et al 2015) | Java, Scala |
| (Linares-Vásquez et al 2015a) | Android Debug Bridge, HierarchyViewer, UIAutomator, jMetal |
| (Arcuri et al 2015) | Java SDK 8, Maven |
| (Long and Rinard 2015) | Amazon Machine Image (AMI) Image and Virtual Machine (VM) Image |
| (Samak and Ramanathan 2015) | VM Images |
| (Shi et al 2015) | PIT, Ekstazi |
| (Gong et al 2015) | Node.js, Java SDK 1.6+, git, libgmp, Chrome, Python 2.7, Visual Studio 2010, Windows 7 64-bit SDK |
| (Nguyen et al 2015a) | Eclipse, PHP Development Tools for Eclipse |
| (Siegmund et al 2015b) | git, xamarin, accord, accord.math, aforge, aforge.math, ilnumerics (only for MacOSX described) |
| (Xu et al 2015) | Ant, Java |
| (Eichberg et al 2015) | JDK8+ |
| (Smith et al 2015) | GenProg, TrpAutoRepair |
| (Dhar et al 2015) | Java, Soot |
| (Nguyen et al 2015b) | Java SDK 6, Eclipse |
| (Beyer et al 2015) | VM Image or Linux, OpenJDK 1.7, CPAchecker, UltimateAutomizer |
| (Park et al 2015) | VM Image |
| (Hermann et al 2015) | Scala, R |
| (Foucault et al 2015) | VM image |

information is supplied, the ETL process for this project can be started and is performed automatically. This process generally also works for closed source software, as we just need to be able to clone a project via git. Nevertheless, this feature is not implemented in this version of SmartSHARK, as we focus on open-source projects first, because they are widely used in research.

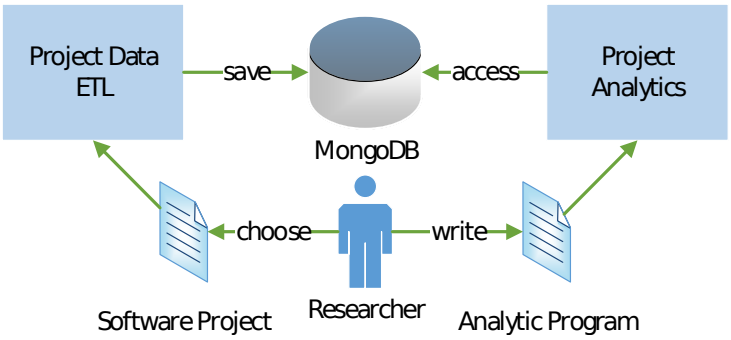


**Fig. 1** Design of SmartSHARK

Currently, SmartSHARK supports GIT as VCS. The programming language is required by InFamix[10] for the calculation of software metrics. During the ETL process, the progress can be observed in the web front-end to give the user feedback. Furthermore, each project has its own configuration files, which can optionally be edited via the web front-end to customize the ETL process. In this prototypical version of SmartSHARK we only support data collection from VCSs and mailing lists. Nevertheless, we plan to include the collection of data from several Issue Tracking Systems (ITSs) (e.g., bugzilla, jira) in our improved platform, which is explained in Section 7.

#### *4.1.1 Model-based Fact Extraction and Transformation Framework*

Many tools and methods, which are used for data extraction and application, are context-specific. These tools are often have a tight coupling between the extraction process and the application, which can be problematic.

The output from the extraction process for each tool is usually in a format suitable for the intended application, which can make the integration of the output from different tools challenging. Additionally, each application may have certain requirements towards its input, requiring further adaptation of the integrated output from the extraction process. Performing the integration and adaptation at input and output level may incur considerable development overhead and some redundancies, while still remaining context-specific. Figure 2 illustrates an abstract representation of this scenario where inputs and outputs are integrated in an ad-hoc manner at a lower level of abstraction. The programs in this figure are available data collection programs (e.g., CVSAnalY[11]), that generate different outputs (e.g., CVS or XML files). These outputs are then integrated by different integration scripts in an ad-hoc manner, depending on what the application need as input. For example, if a researcher wants to analyze data using the application WEKA (Hall et al 2009), she needs

[10] The developing company Intooitus does not exist anymore and the tool is also not available anymore.

[11] `http://github.com/MetricsGrimoire/CVSAnalY`

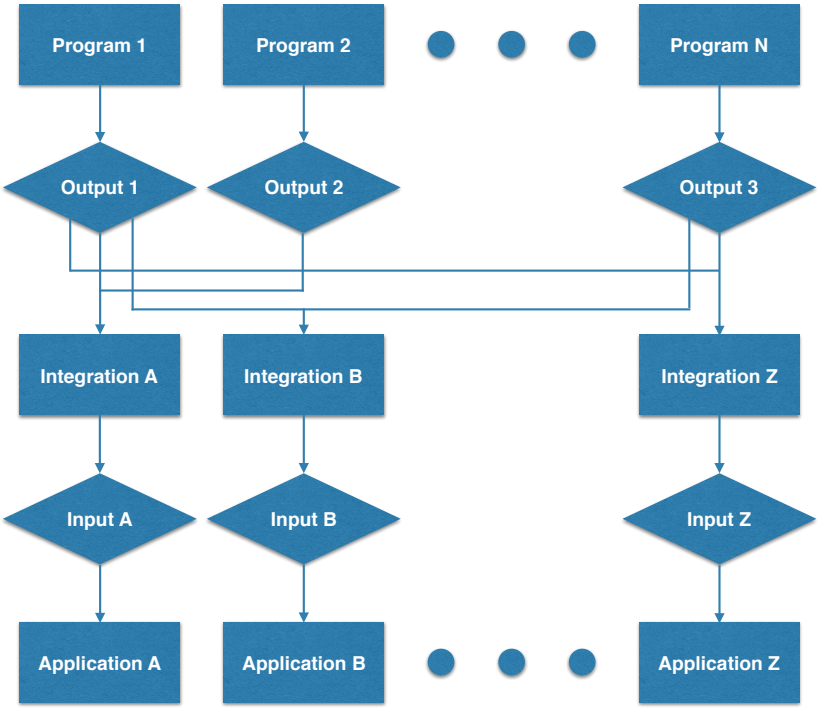


**Fig. 2** Ad-hoc integration at input-output level.

to make sure that all the output from the data collection programs are integrated and transformed into a format that is readable by WEKA (Hall et al 2009). If the researcher now wants to use another application for the analysis besides WEKA (Hall et al 2009), she needs to create another integration to change the input accordingly, which results in a development overhead.

Hence, Makedonski et al (2015) proposed a model-based software mining infrastructure, called DECENT, to circumvent these problems. The infrastructure relies on domain-specific models of facts extracted from the outputs of the different tools, making the facts available in a homogeneous high-level representation. These domain-specific facts models are then integrated and transformed into models that are related to a certain assessment task. The integrated assessment task models are queried and exported into the input formats required by the various applications.

The DECENT infrastructure is built on top of the Eclipse Modeling Framework (EMF). The EMF and related technologies provide interfaces for obtaining models from various lower-level representations, such as structured text, relational databases, XML files, and CSV files, by means of corresponding mappings. This makes the integration of various input and output formats

very convenient and reusable, where resulting models can be used in a variety of contexts with little development overhead.

To extract facts from raw assets, DECENT uses various tools, which are widely used in research. These include CVSAnalY for extracting information out of source code repository logs (e.g., used by Fernandez-Ramil et al (2009) and Herraiz et al (2007)), InFamix for calculating software quality metrics (e.g., used by Alexandru and Gall (2015)), and DuDe (Draisbach and Naumann 2010) for duplication detection.

A high-level overview of the DECENT infrasturcture is shown in Figure 3. Existing programs (such as CVSAnalY or InFamix) can be used to collect data from software projects . The resulting outputs in different formats (e.g., MySQL databases, XML files) are translated into facts models by means of resource mappings and then combined into integrated assessment models by means of model-to-model transformations resulting in a DECENT model instance that corresponds to the project. This model instance now holds all the project data relevant for the assessment task that was collected by the different tools. Thanks to EMF, the model instance can be stored in different output formats, such as a relational database, XML file, or a MongoDB with little or no additional development overhead. The model instance can then be querried and refined further to suit the input requirements for the different applications.

#### *4.1.2 VCS-ETL*

The VCS-data, obtained during the ETL process, is change-based. For each change (i.e. commit), we store all changed software artifacts and their location in the project. Therefore, it is possible to reconstruct the whole project structure at any point in time. The software artifacts include source code files, classes, methods, functions as well as documentation files, such as readme files. Five different types of data are stored for each artifact: 1) software metrics; 2) source code changes; 3) project structure; 4) change history; and 5) bug-related labels. This data is stored for each commit of the project.

The software metrics include static source code metrics, change metrics, and social metrics (e.g. developer cooperation factors). A list of all the metrics can be found in the SmartSHARK documentation online[12]. Additionally, we save delta values for each metric, which indicate how this metric has changed since the artifact was last touched. In addition to the metrics, we also save the concrete textual change that was made, i.e., the diff [13] of the commit. The diffs can be particularly important for the development of new analytic approaches, as shown by Walden et al (2014). The project structure includes information about the location of each artifact at each commit, including logical artifacts such as methods and classes. The change history includes information regarding the time and purpose for each change as well as the person responsible for

[12] `http://smartshark.informatik.uni-goettingen.de/index.php?r=site%2Fmongodesign`

[13] `http://www.gnu.org/software/diffutils/`

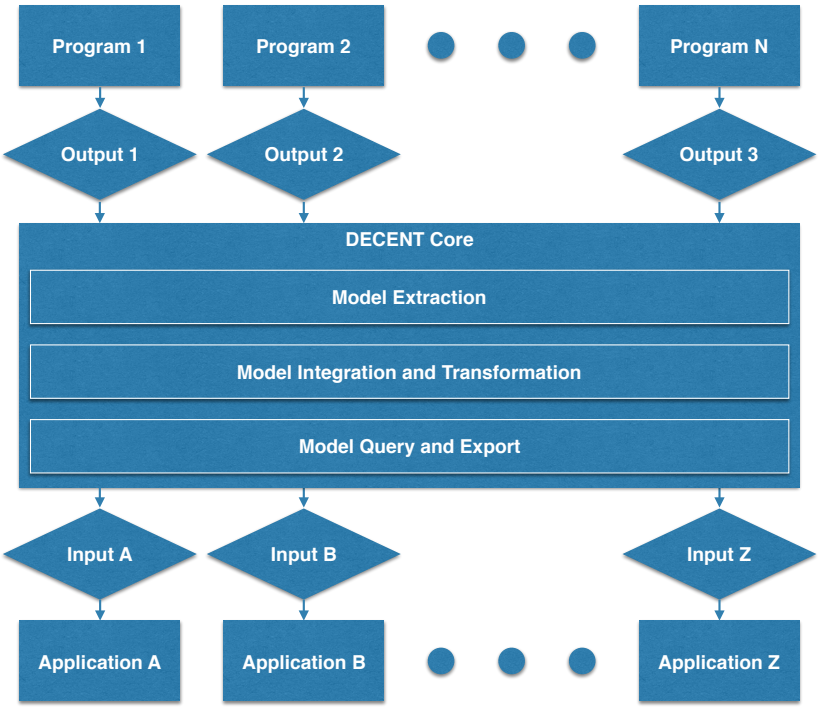


**Fig. 3** High-level model-based integration with DECENT.

it. The bug-related labels include information on whether a change is considered to be a fix and/or a cause for a fix, i.e. a bug.

Whether a change is considered to be a fix can be determined based on a regular expression evaluated against the commit message or other means. Whether a change is considered to be the cause for a fix is computed by means of a weighted multi-factor multi-layer approach for identification of potential causes for events of interest based on an origin analysis (Makedonski and Grabowski 2016). The approach establishes cause-fix relationships between changes to artifacts at different layers of granularity (project, file, class, method) in a cause-fix graph. Based on weights assigned to a fix according to a given factor, such as a regular expression, it computes the contribution of each cause to the fix depending on the number of causes. Then, for each cause it computes the average contribution to fixes according to the given factor. The approach supports determining the causes for fixes across different factors, such as refactorings and bug fixes, based on regular expressions or other means for determining the fixes. It is implemented within the DECENT infrastructure and by default it uses regular expressions inherited from CVS-

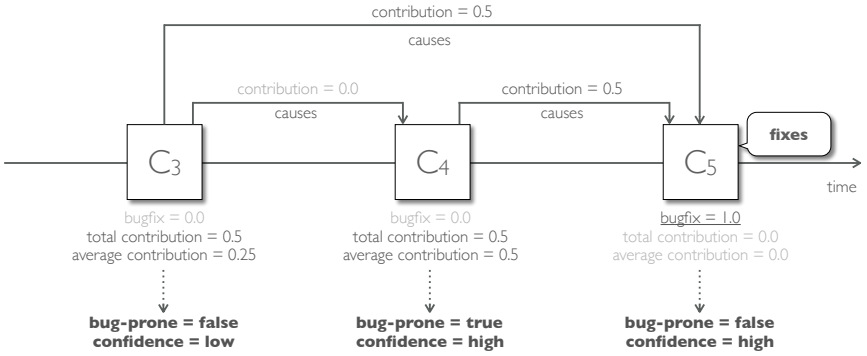


**Fig. 4** Identifying causes and fixes.

AnalY [14] for determining whether a change is considered a fix for a bug. The regular expressions can also be customized further to suit a specific deployment scenario.

A simplified overview of the approach for finding causes and fixes is outlined in Figure 4 containing an excerpt from a cause-fix graph including three changes $C_3$, $C_4$, and $C_5$. $C_5$ is identified as a fix based on the commit message, thus it has a weight for the "bugfix" factor equal to 1.0. $C_3$ and $C_4$ are identified as potential causes, as, e.g., they both changed lines in a file that were also changed in the fix in $C_5$. For both, $C_3$ and $C_4$ it is assumed that they contribute equally [15] (contribution = 0.5) to the fix. In this example, the total contribution to fixes for both $C_3$ and $C_4$ is the same, however, since $C_3$ also contributes to changes in $C_4$, where $C_4$ is not identified as a fix under the "bugfix" factor, the average contribution of $C_3$ is lower.

Based on the computed average contributions to causing a fix, changes are labeled as causes for fixes. In addition to the label (true or false), a confidence indicator (high, low) is added to indicate whether the resulting label can be trusted. The label is based on a threshold derived from the distribution of the average contributions. The threshold for this study is calculated as 50% of the mean average contribution for the corresponding level of granularity. If the average contribution of a change is below the threshold, it is less likely that the change caused a fix as the actual cause for the fix is either distributed among several potential causes or the change caused a number of other changes that were not necessarily considered fixes. If the average contribution of a change is above the threshold for the corresponding level of granularity, it is more likely

[14] The default set of regular expressions includes:
`"defect(s)?", "patch(ing|es|ed)?", "bug(s|fix(es)?)?", "(re)?fix(es|ed|ing|age|\s?up(s)?)?", "debug(ged)?", "\#\d+", "back\s?out", "revert(ing|ed)?"`

[15] The default assumption may be overridden by applying different strategies based on the size of the changes or other information.

that the change caused a fix (i.e. was bug-prone). The confidence indicator is based on the proximity to the threshold. If the average contribution of a change is close to the threshold then the label is considered less reliable, hence its confidence is "low". In this study, we considered a window of 1% around the threshold. Considering the example scenario illustrated in Figure 4, if we assume for illustrative purposes that the mean average contribution for the file level (including all changes, not only the shown ones) is 0.5, the threshold will be 0.25 (50% of the mean average contribution). In this case, $C_4$ is labeled as bug-prone, while both $C_3$ and $C_5$ are labeled as not bug-prone, however, $C_3$ will have a low confidence as it is close to the threshold, while $C_4$ and $C_5$ will have a high confidence as they are farther away from the threshold.

By applying this approach, a DECENT model instance is enriched with the computed contributions, labels, and confidence indicators for each change at each level of granularity.

#### *4.1.3 Mailing List-ETL*

SmartSHARK extracts, transforms, and loads mailing lists of projects in the MongoDB. The data includes information about the sender, receiver, time, subject, actual message, and if the mail was sent in response to another mail.

This data is linked via the email address of the contributor to the VCS data, as proposed by Herraiz et al (2006). Therefore, it is possible to build complex developer cooperation networks on basis of this data. This is especially interesting for the development of new models regarding the cooperation and communication of developers or to perform intention mining, as shown by Di Sorbo et al (2015).

### 4.2 Software Analytics

We define software analytics as data analytics applied in the domain of software development to gain insights into the software development process and support the associated decision making process. This includes defect prediction (Catal and Diri 2009), effort prediction (Jorgensen and Shepperd 2007), dependency analysis (Honsel et al 2015), developer social networks (Jermakovics et al 2011), and other topics related to the field of software evolution.

With the collected data, various kinds of aspects regarding the software projects and their evolution can be analyzed, e.g., the number of file changes in a given timeframe, changes within the file ownership, and patterns in mailing list activities. Furthermore, different applications of machine learning, e.g., for defect prediction are possible.

To facilitate such a diversity of analytic problems, SmartSHARK uses the Apache Spark framework as analytic back-end. Apache Spark is especially designed for big data analytics. In our deployment, Apache Spark uses an Apache Hadoop cluster to distribute data and computations across nodes. For the definition of analytic jobs, Spark supports multiple languages. With

SmartSHARK, we currently support Java and Python. To perform analytics on SmartSHARK, the researcher first writes their analytic program. The analytic programs can use the whole functionality of the programming language as well as Spark specific features. The standard language features are computed on a single node of the Hadoop cluster used by Spark. The Spark specific features allow the distributed computation of results within the Hadoop cluster and, therefore, analytics that are very complex or data intensive. Researchers can upload and execute the compiled Java application or a Python file to the platform using the web front-end. The front-end allows to add Spark arguments as well as program arguments. This allows for parametrizeable analytic jobs, e.g., to define the name of the project to be analyzed or a certain threshold important for the analytics. For the execution, the Spark jobs are submitted to the Hadoop cluster. Results can either be saved in the MongoDB to enrich the database or as a file that users can download later.

### 4.3 Web Front-End

From an end-user perspective, the web front-end is the central part of our platform and it is based on the Yii2 framework[16]. It is the only place where users directly interact with SmartSHARK and the starting point for both the data collection and the software analytics. The front-end allows to add projects as well as view information about the already processed projects, e.g., general information about the number of commits, project ages, and changed artifacts in each commit. Moreover, it provides a job submission interface, through which users of SmartSHARK can run their analytic programs. Each user owns a results folder, which can be accessed through the web front-end. SmartSHARK implements a simple access rights system. We defined different roles: the administrator has all rights on the web interface and complete access to the MongoDB, whereas the advanced spark user role is restricted in using the web interface and can only modify the underlying MongoDB through analytic jobs. The normal user role has very limited access rights in the web interface and can only read data from the MongoDB.

The front-end is connected via a REST API to the Apache Hadoop cluster manager. But in the current state of the platform several shell scripts are executed for the data ETL process and the sending of the analytic jobs to the Spark instance. Nevertheless, in the near future we want to change this by providing a REST API for submitting Spark jobs.

Moreover, the web front-end allows users to create a backup of the MongoDB for download. This way, users can replicate the data of the platform and, e.g., use it in their local environment or archive it as part of a replication set for a published study.

[16] `http://www.yiiframework.com/`

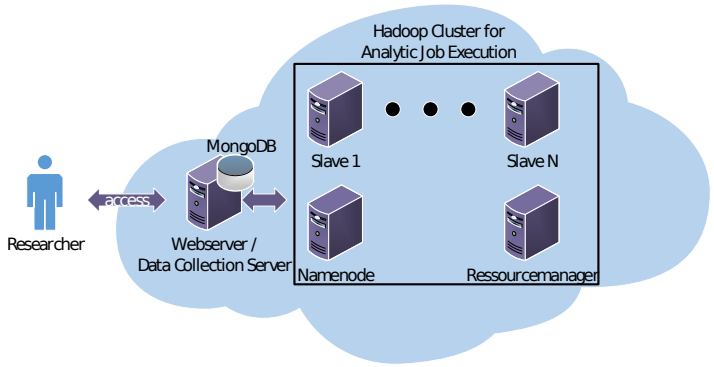


**Fig. 5** Infrastructure of SmartSHARK.

### 4.4 Cloud Deployment

The infrastructure of our SmartSHARK deployment is depicted in Figure 5. The web-server also serves as database server and data collection server. Furthermore, there is an Apache Hadoop cluster to execute analytic jobs. The platform can be automatically deployed via our deployment scripts using the DevOps technologies Vagrant[17] and Ansible[18]. Details regarding the used software versions and more information regarding the deployment are given on the homepage of SmartSHARK[19].

## 5 SmartSHARK Experience Report

In order to evaluate if SmartSHARK can address the problems of heavy data re-use, non-availability of data sets, non-availability of implementations, small data sets, and diverse tooling, we made experiments with the platform. We collected data from multiple projects to see if such a high degree of automation for such complex data is feasible. Then, we proceeded with the definition of three kinds of software analytics: 1) visual analytics of evolutionary trends during the development and within the mailing list usage; 2) a complex machine learning based defect prediction approach; and 3) a statistical evaluation of data in order to support effort estimation. The three analytical examples are selected in a way to cover a broad area of research topics. Furthermore, each of these topics is a very prominent research area. Additionally, we wanted to evaluate if the machine learning support of Apache Spark is good enough for complex research tasks like defect prediction. The last part of our experience report is related to an often neglected, but important attribute of such platforms: the usability.

Note that the experience report is based on our first prototype version of SmartSHARK, which is presented in Section 4. In Section 7 we describe how

[17] `https://www.vagrantup.com/`
[18] `http://www.ansible.com/`
[19] `http://smartshark.informatik.uni-goettingen.de/`

we changed our platform to cope with the limitations that we identified during our work with SmartSHARK.

### 5.1 Data Collection

To evaluate the data collection, we randomly selected 23 projects from GitHub[20]. We did not follow any specific methodology for the selection of projects, instead we used the *explore* function of GitHub to browse available projects. The only requirement for the inclusion of a project was that it is programmed in Java, C, or C++. Table 3 lists the chosen projects, including number of commits, number of files in the repository, size of the repository, programming language, number of stored mails, and a very brief project description. We started the ETL process via the SmartSHARK web interface and noted the following problems during the data collection:

- CVSAnalY stopped working for the oryx project for a while, but then simply resumed. Upon further investigation we determined that this was most likely a race condition in the "Content" extension of CVSAnalY. The "Content" extension of CVSAnalY is used to store the textual content of files at each revision.
- InFamix did not work for all revisions of the Mahout[21] project. We were not able to track down the problem because InFamix is a closed source software.
- The collection of source code metrics was restricted to one programming language per project, because InFamix does not support multiple languages within a project.
- We could collect the data for only one project at a time due to the limited resources and the high resource consumption in terms of both memory and computing power required for the data collection.

Besides these problems, there were no other failures in our data collection process for the projects we have tested.

Nevertheless, in our current version of SmartSHARK we have substituted InFamix and CVSAnalY with two tools, that we developed on our own. More information regarding these tools can be found in Section 7.

[20] `http://github.com/`

[21] `https://github.com/apache/mahout`

**Table 3** Information about the examined projects.

| Project | Lang. | #Commits | Size | #Files | #ML Messages | Type |
|---|---|---|---|---|---|---|
| https://github.com/mtytel/cursynth | C++ | 219 | 3 MB | 185 | n/a | Audio |
| https://github.com/dmlc/cxxnet | C++ | 852 | 4 MB | 173 | n/a | Machine Learning |
| https://github.com/elastic/elasticsearch-hadoop | Java | 1243 | 9 MB | 576 | n/a | Search Engine |
| https://github.com/facebook/fatal | C++ | 401 | 3 MB | 141 | n/a | Programming |
| https://github.com/google/guice | Java | 1442 | 89 MB | 713 | n/a | Programming |
| https://github.com/lawloretienne/HackerNews | Java | 12 | 1 MB | 78 | n/a | News |
| https://github.com/KDE/k3b | C++ | 5508 | 47 MB | 1069 | 587 | Media |
| https://github.com/KDE/ksudoku | C++ | 668 | 7 MB | 233 | 12544[22] | Game |
| https://github.com/01org/libxcam | C++ | 250 | 3 MB | 242 | n/a | Camera |
| https://github.com/01org/libyami | C | 487 | 4 MB | 248 | 108 | Media |
| https://github.com/apache/log4j | Java | 3266 | 18 MB | 643 | 54546 | Programming |
| https://github.com/lawloretienne/Minesweeper[23] | Java | 65 | 7 MB | 207 | n/a | Game |
| https://github.com/dmlc/mxnet | C++ | 236 | 1 MB | 124 | n/a | Machine Learning |
| https://github.com/oclint/oclint | C++ | 733 | 3 MB | 349 | n/a | Programming |
| https://github.com/google/ohmu | C++ | 226 | 3 MB | 185 | n/a | Programming |
| https://github.com/SFTtech/openage | C++ | 1761 | 8 MB | 560 | n/a | Game |
| https://github.com/cloudera/oryx | Java | 372 | 48 MB | 537 | n/a | Machine Learning |
| https://github.com/facebook/osquery | C++ | 2208 | 9 MB | 555 | n/a | Instrumentation |
| https://github.com/gamelinux/passivedns | C | 220 | 1 MB | 50 | n/a | Network |
| https://github.com/ushahidi/SMSSync | Java | 1395 | 40 MB | 557 | n/a | Messaging |
| https://github.com/facebook/swift | Java | 496 | 8 MB | 427 | n/a | Programming |
| https://github.com/01org/wds | C++ | 238 | 3 MB | 193 | 27 | Media |
| https://github.com/dmlc/xgboost | C++ | 1847 | 8 MB | 373 | n/a | Mathematics |

[22] There is no ksudoku specific list. Instead we collected the whole kde-games-devel mailing: https://mail.kde.org/pipermail/kde-games-devel

[23] The project is not available anymore.

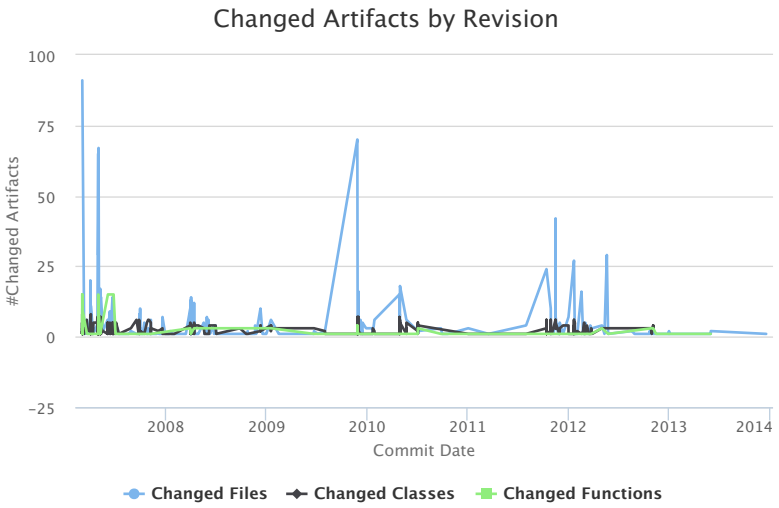


**Fig. 6** Example of the change history visualization for the project ksudoku.

## 5.2 Software Analytics

The second part of our experience report should evaluate how well the collected data in combination with the architecture of the SmartSHARK platform is suited to perform tasks from software analytics. The source code for all analytics is available in the SmartSHARK SVN.

### *5.2.1 Visualization of Evolution Trends*

We created six different visualizations for the projects we added to SmartSHARK.

**Change History Visualization.** This visualization shows how many software artifacts (i.e., files, classes, functions) were changed at which point in time (i.e., commit). Furthermore, the commit message can be retrieved for each change in this visualization. An example for this visualization is shown in Figure 6 for the project ksudoku.

**File-level Bug Overview Visualization.** This visualization depicts the number of changed files over the project lifespan together with the number of defects. An example for this visualization is shown in Figure 7 for the project k3b. This figure only shows the graph for the number of defects over the time, while the other graph is deactivated. The confidence cut indicates till which revision we can trust our defect label data. We can not trust all the data, as we do not know directly when a new version of the program is committed, which files in the new revision are defect prone. The detection of bugs within revision takes some time (some bugs are detected in week, some after three years or even later). Therefore, we have chosen such a confidence cut as suggested by Tan et al (2015) to draw a line between trustful and non-trustful data.

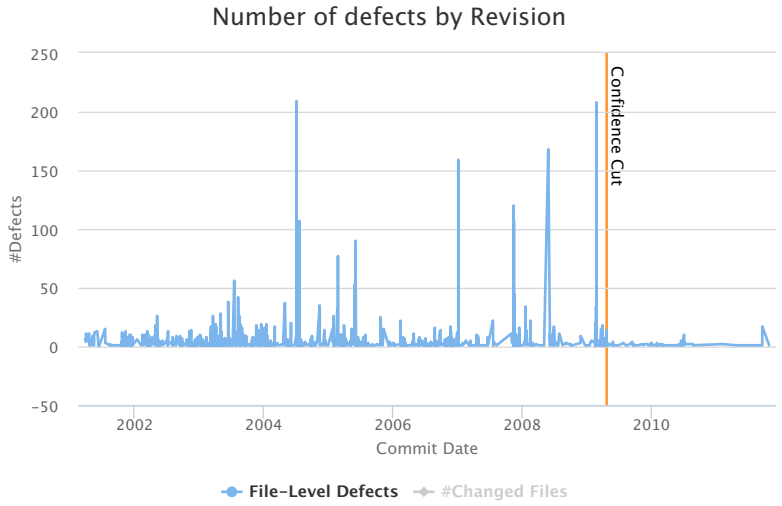


**Fig. 7** Example of the file-level bug overview visualization for the project k3b.

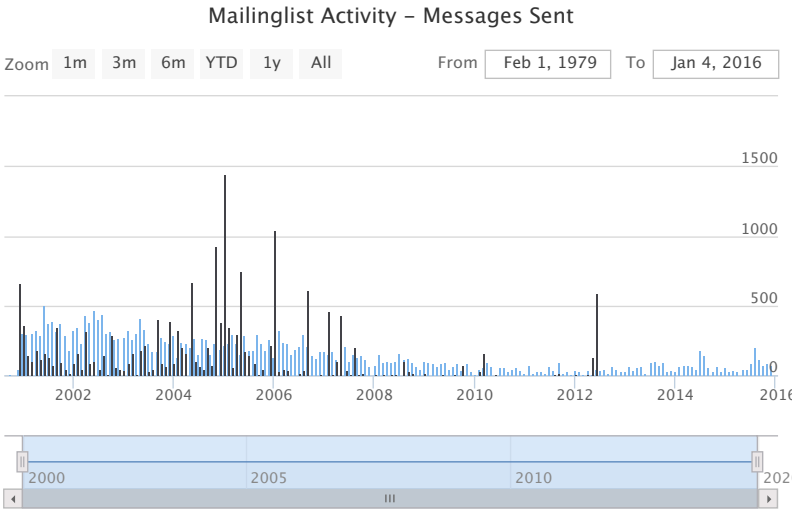


**Fig. 8** Example of the mailing list activity (sent messages) visualization for the project log4j. The blue bars depict the number of sent messages, whereas the black bars shows the number of changed files at a certain point in time.

**Mailing List Activity - Number of sent Messages.** For projects that have a mailing list, this visualization depicts how many messages were sent at which point in time. The mailing list activity is shown together with the number of changed files for the project in order to allow researchers to look for correlations. Figure 8 depicts an example for this kind of visualization. This figure shows the number of sent messages (blue bars) and changed files (black bars) for the project log4j.

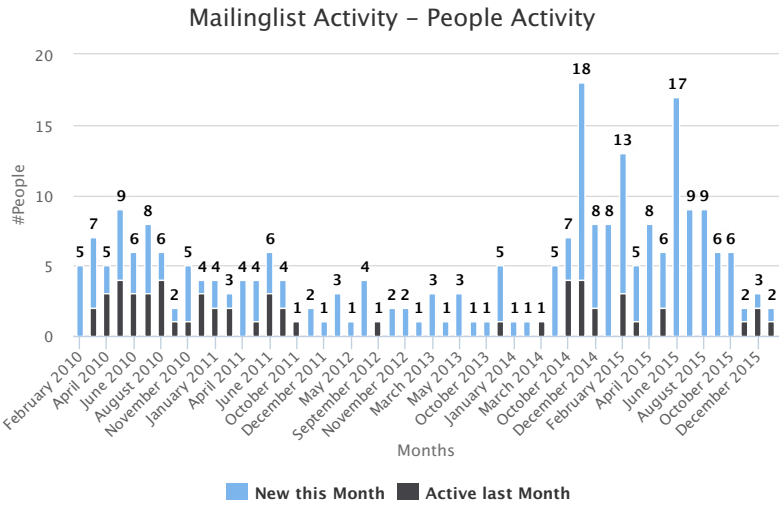


**Fig. 9** Example of the mailing list activity (active people) visualization for the project k3b.

**Mailing List Activity - Activity of contributing People.** For projects that have a mailing list, this visualization shows how many people were active on the mailing list on a monthly basis. The visualization allows to differentiate between new users on the mailing list and these also active in the previous month. An example for this visualization is depicted in Figure 9 for the project k3b.

**Mining Email Social Networks.** For projects that have a mailing list, this visualization shows the social network of developers. This visualization is a replication of a figure from the paper by Bird et al (2006). We analyzed the mailing list by the number of messages sent by developers, and the number of messages sent in reply to developers. There exist an edge between developer A and developer B if A has sent a message to B[24]. Here, we distinguish between big and small mailing lists: If overall more than 3000 messages were sent on the mailing list, there exist an edge between A and B only if A has sent B at least 150 messages, which is conform to the approach by Bird et al (2006). For mailing lists smaller than 3000 messages, we did not apply this rule, as the goal of this threshold is to reduce the size of the figure, which is not necessary for smaller mailing lists. Nevertheless, we experienced, that for this kind of visualization the data must be thoroughly cleaned and prepared. For some projects, we were only able to reconstruct that the developers have sent messages to the mailing list, but not if they were replies to other messages. An example for this visualization is depicted in Figure 10 for the project libyami.

**Defect Prediction Results Visualization.** In order to show if the platform can also be used to give feedback on analytics performed, we created a visualization of the defect prediction results (see Section 5.2.2). We show which

[24] Note, that all messages were always additionally sent to the mailing list.

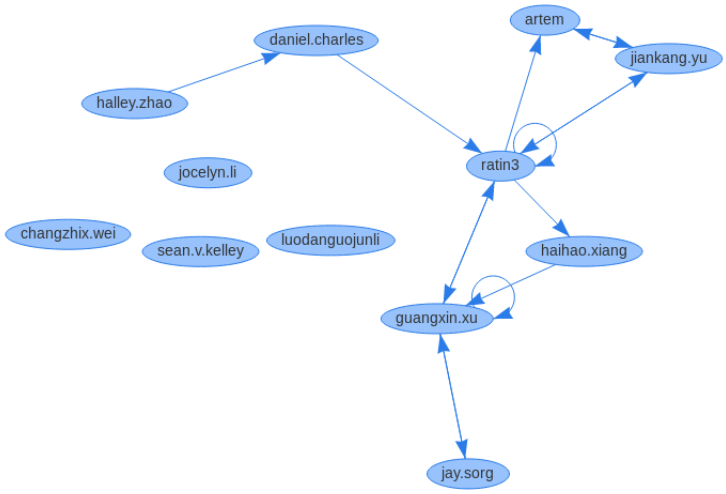


**Fig. 10** Example of the email social network of the project libyami.

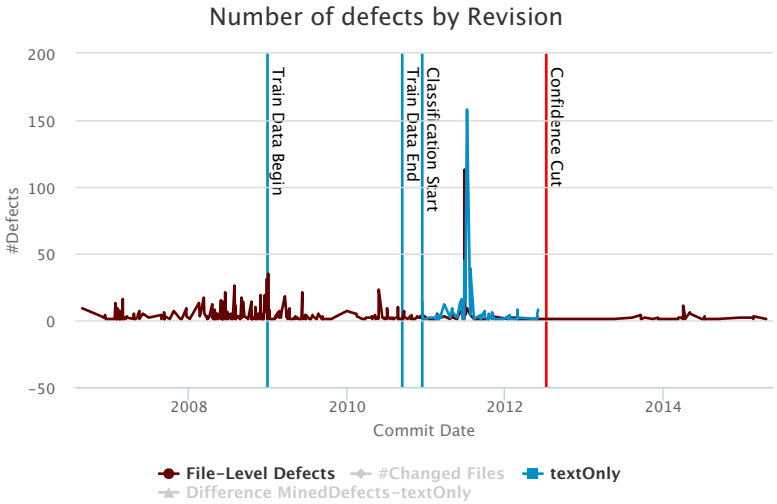


**Fig. 11** Example of the defect prediction results visualization for the project guice.

files were predicted as defective by our created model in comparison to which files were marked as defective by the data collection process. This visualization shows, that visualizations for results computed by complex analytics are also possible with the platform. An example for such an visualization is depicted in Figure 11 for the project guice. The figure shows for a classifier which data was used for the training and when the classification started. It also shows the results (blue line). The reasoning behind the confidence cut is explained above.

The visualizations give a first impression about the project and provide researchers insights regarding several evolutionary questions, e.g., patterns on the mailing list activity, the evolution of the size of a project or when bugs were created. Furthermore, we were able to replicate a visualization from the paper by Bird et al (2006).

In general, this visualizations use different javascript visualizations libraries like HighCharts/HighStock[25] or visJS[26]. These libraries can be easily integrated for use in a view in the web front-end of SmartSHARK. Therefore, if we want to create a new visualization, we can choose a library that supports this visualization style and add it to the web-frontend to access the library in our created view. Other visualizations are accessible via the SmartSHARK website.

### *5.2.2 Defect Prediction*

The second analytic example implemented on SmartSHARK is a defect prediction model. For this, we performed a conceptual replication of a recent publication by Tan et al (2015). The authors suggest to perform just-in-time defect prediction, i.e., for each commit, based on change metrics (e.g., added lines) as well as textual data from the commits and source code. Tan et al. suggest to use the first part of a project as training data, then leave a gap and predict the remainder of the project using a prediction model trained on the first part of the data. We selected to replicate this approach, as we are not aware of any public defect prediction data set that contains both the metrics and textual features.

The data required for the defect prediction was readily available in the MongoDB back-end of SmartSHARK, i.e., the defect labels, which were determined by DECENT and the software metrics, which are calculated by InFamix via statically analyzing the source code. As explained in Section 4.1.2, this data (labels and metrics) is collected for each commit of a project for each changed artifact.

Using the functionality provided by Spark, creating the data splits as required for the approach by Tan et al (2015) as well as the training of the defect prediction model was straight-forward. After fetching the data from the MongoDB, a Map job was used to prepare the data. Then, the defect prediction model was trained using Apache Spark's Mllib. We evaluated our classifier with the project data available in the MongoDB. Furthermore, a confusion matrix is created to allow the calculation of performance metrics like *recall* and *precision*. The prediction results, as well as the confusion matrix, are then stored in the MongoDB for later use by the visualization.

To evaluate the versatility of the platform to exploit different types of data, as well as create different types of models, we created four different defect prediction models as described above, all using different data and classifiers:

[25] See: http://www.highcharts.com/
[26] See: http://visjs.org/

1) a logistic regression model based on static source code metrics and change metrics; 2) a random forest based on static source code metrics, change metrics, and social metrics; 3) a naïve bayes for text classification based on textual diffs of revisions; 4) and a majority voting scheme with three algorithms: a random forest and a logistic regression model trained on the static source code metrics, change metrics, and social metrics as well as a naïve bayes model trained on the textual diffs. All of these models could be trained and defined without any problems.

Thus, our example demonstrates that SmartSHARK can be used for regression of probabilities as shown with logistic regression, classification problems as shown with the random forest, as well as text mining. Moreover, the example demonstrates that the Spark-based approach allows for the arbitrary combination of techniques, which we demonstrate with the voting scheme we implemented.

#### *5.2.3 Effort Estimation*

The third analytic example was created with the aim to gain insights into how much effort must be put into the creation of a simple effort prediction model. The example was inspired by work on effort estimation in open source projects by Gousios et al (2008) and Robles et al (2014). Gousios et al. estimate the contributions of developers as the LOC they contributed, plus additional contribution factors. Robles et al. try to determine the person months invested in an open source project using the number of commits and the number of active days within a period.

With our example, we create a mixture of both. As input, we use a list of projects. To create the effort model, the means for the lines added, lines removed, the growth of the absolute file size of the repository, and the LOC in the repository *per commit* were calculated for each project. The required information is all part of the collected data. For the aggregation of the information with Apache Spark, two Map jobs and two reduce jobs were defined, first to create a map between the file names and the associated metric data, and then to reduce the maps to the required means. Moreover, the mean values not only for each project by themselves, but for all listed projects are calculated. This information gives an overview on the average contribution of developers per commit, similar to the work of Gousios et al (2008).

The output of the analytic job is then a list of the means for the metrics for each project in the list, as well as the overall mean values. This information can be used to estimate and predict the effort for new projects or project phases based on the number of source code revisions that are expected. We denote the mean LOC as $L$ in the following.

The number of expected code revision can be determined the same way, as the mean values of change for each revision: by determining the mean number of commits per developer *per month*, denoted as $C$ in the following. If $\#dev$ developers are working on the project, it follows that the mean output will be $\#dev \cdot C \cdot L$ per month. Thus, if the estimated size of a project release is

$x$, the expected time required for the project is $t = \frac{x}{\#dev \cdot C \cdot L}$ months. This estimation of developer activity for effort estimation mimics the person month estimation by Robles et al (2014), except that we estimate the LOC produced and not the person months.

This straight forward approach shows how SmartSHARK can facilitate insights into the software development using simple analysis that can be used to interpolate how projects evolve in terms of effort measured in time and LOC based on previous research. Please note that we are aware that this is not a perfect state-of-the-art effort prediction model, but just an example we developed to evaluate the potential of SmartSHARK. The same concepts can be applied to other questions, e.g., how the complexity evolves over time simply by replacing LOC with a complexity metric.

## 5.3 Usability

Usability is defined through its major aspects: a given task must be executed with effectiveness, efficiency and satisfaction (ISO/IEC 1998). If users are able to accomplish their tasks, the product has a good effectiveness. Efficiency includes, that users need a minimum amount of resources (e.g. psychological strain, time, material) to achieve their task or goal. If the users expectations are fulfilled or even excelled they feel satisfied with the product. To assess these attributes we conducted different experiments with SmartSHARK. In this section, the usability of the four most important tasks is evaluated.

### *5.3.1 Definition of Analytic Jobs*

SmartSHARK allows the usage of any existing library for the definition of analytic jobs. In case the full power of Apache Spark is required, e.g., to achieve scalability for complex analytics, features for the distributed processing must be used. This includes regular map/reduce jobs and libraries, which are directly developed for Apache Spark. We demonstrated the use of the Spark machine learning library for the definition of our defect prediction model.

The definition of analytic jobs fulfilled all of the different major usability aspects, as defined in the ISO 9241-11 (ISO/IEC 1998) standard, because we were able to successfully define different analytic jobs without much effort. Furthermore, we were able to use libraries to which we are used to, which fulfilled our expectations.

### *5.3.2 Debugging*

Currently, SmartSHARK offers the possibility to gather debugging information via log files of the Hadoop cluster. Although, this solution is feasible, most developers are used to directly debug within their Integrated Development Environment (IDE). Furthermore, the debugging of applications which are executed on multiple nodes, is difficult to perform, e.g., debugging Hadoop

Map Reduce programs[27]. Nevertheless, Apache Spark offers debugging possibilities, which can be included in later versions of SmartSHARK to offer a direct interface for developers to debug their programs.

We were able to successfully debug our programs via the log file access. But the satisfaction for this task is currently lacking, because the debugging was a repetitive task as we needed to gather information from the log file to pinpoint the bug.

#### *5.3.3 Evaluation and Presentation of Results*

The evaluation of the results with Apache Spark is well supported by libraries. The feedback loop, of how the results are displayed and returned to the user, is passive. Currently, SmartSHARK offers three options: 1) storage of the results directly in the MongoDB, as we demonstrate with the defect prediction example; 2) storage on the file system for later download by the user as we do it in the effort estimation example; and 3) directly output the desired results via the language specific print commands.

One way to make the analysis of the results more comfortable was shown by incorporating the defect prediction results directly in the SmartSHARK website. By providing such a visualization as shown in Figure 11 together with metrics like recall, precision, G-Measure, F-Measure, and MCC, the evaluation of, e.g. defect prediction, approaches gets more comfortable. The researcher only needs to execute the corresponding Spark job for a project and the website will then show the results directly.

Therefore, the task of evaluating and presenting the results fulfilled all of the different usability aspects, at least for our defect prediction example. Our visualization directly gives feedback to the developer, without the need of her to actively retrieve the results. This shows, that SmartSHARK is capable of switching the feedback loop from a passive one (i.e., the user needs to get active to get the results) to an active one. The only requirement for this is the provision of Yii2-Widgets[28], which are plug-ins that implement the logic to gather and display the data. Furthermore, these plug-ins can easily be shared.

#### *5.3.4 Addition of New Data Sources*

We developed SmartSHARK in two increments: the first increment contained only the ETL of VCS data, while in the second increment the ETL of mailing list data was added to the platform. Therefore, at first we were only able to make use of the first increment (collecting VCS data), but as we added the second one we were able to collect the mailing list data without problems.

Hence, our design of the ETL process and the schema of the MongoDB ensured that the task of adding new data sources could be performed effectively, efficiently and with satisfaction.

[27] `https://wiki.apache.org/hadoop/HowToDebugMapReducePrograms`

[28] `http://www.yiiframework.com/doc-2.0/guide-structure-widgets.html`

## 6 Discussion

Within this section, we discuss how SmartSHARK can be used to address the five problems mentioned in Section 1. Furthermore, we compare the lessons learned from CODEMINE, with our own experience with SmartSHARK. At last, we present lessons that we learned during the design, development, and usage of SmartSHARK.

### 6.1 Addressing Threats to Validity and Replicability with SmartSHARK

The motivation of our work on SmartSHARK are the five major problems, which are discussed in Section 1. Based on our experience with SmartSHARK, we discuss the impact a platform like SmartSHARK can have on these problems, how it may help to overcome these problems, or which additional work is required.

**(1) Heavy Re-use of Data Sets.** A platform like SmartSHARK natively addresses this problem. Due to an automatic project data ETL process, new projects can be added without much effort. This way, SmartSHARK provides a foundation for a constantly growing database, which means that with every experiment, new data could be used. However, SmartSHARK also demonstrates the limitation regarding this research question: the desired analytics are a central part. The used platform must be able to collect the required data for the required analytics. If it does not provide such data it must be extensible in a way that researchers are able to plug-in their own implementations so that they can also be shared easily across different SmartSHARK instances.

Currently, our ETL process is extensible and, moreover, the results of Spark jobs can be stored in the MongoDB to further enrich the data, if required. Additionally, by providing a plug-in system, researcher are able to plug-in their existing programs (see: Section 7).

**(2) Non-availability of Data Sets.** SmartSHARK provides two possible ways to address this problem: the first is a shared cloud deployment, where all researchers have access to the same data. The second is to create and share backups of the underlying MongoDB.

**(3) Non-availability of Implementations.** All analytic jobs on SmartSHARK are provided as Apache Spark jobs. Therefore, published implementations can be executed on each instance of SmartSHARK, both private and public. Implementations can be shared both as source code or compiled Spark jobs. However, SmartSHARK itself currently does not directly offer the sharing of the implementation, this must be done at a third-party side. Therefore, one extension would be the addition of an implementation catalog, where researchers can upload their implementation to share them with other researchers.

Our experience shows, that the adding of new visualizations is easy, as our website uses the Yii2 framework, which provides a plug-in mechanism for

widgets. Therefore, if a researcher develops a visualization, it can easily be shared and installed in any SmartSHARK instance via this mechanism.

**(4) Small data sets.** Through the automated data collection, the addition of new data to SmartSHARK is not very time consuming for researchers, even if the collection itself might take a while. Hence, constantly growing data sets are not a problem, which will lead to a large body of rich data that includes the structure and meta information about the VCS, software metrics, defect information, and mailing list data.

**(5) Diverse Tooling.** SmartSHARK currently addresses this problem differently for data collection and analytic applications. For the data collection, we use a model-based framework for the harmonization of the heterogeneous data produced by diverse tooling. This allows us to abstract away from concrete low-level representations in diverse formats such as XML, relational databases, CSV files and other assets, and instead work with homogeneous high-level models that allow us to use model-to-model transformation to integrate and enrich data, while still suppotring translation between different concrete representations of the models. We have experienced, as described in Section 5.3.4, that this approach is easily extensible and also feasible for the data collection and combination.

For the analytics, we address this problem by using Apache Spark as analytic back-end. This means that for the execution, the problem of tool diversity is resolved, as Spark jobs can easily be shared and executed on any SmartSHARK instance.

## 6.2 Lessons from CODEMINE

While not many details about Microsoft's internal CODEMINE (Czerwonka et al 2013) were published, the authors gave a list of lessons learned during the creation of the platform. We now compare these lessons to our experience with SmartSHARK as a tool for the research community.

**Create an independent instance for each product team in the data platform.** Partitioning of the platform parts to, e.g., restrain access is one import lessons that the developers of CODEMINE learned. The target of SmartSHARK are not product teams, but rather research communities. Nevertheless, we built up our platform as modular as possible to be able to restrain access to certain parts.

**Have uniform interfaces for data analysis.** Czerwonka et al (2013) describe in their lessons learned, that a uniform interface is very important for such a platform. Furthermore, the evolution of the API for data analysis must be done carefully. This lesson lead to the central design decision of SmartSHARK to use Apache Spark as common API for the analytic tasks. This uniform interface allows more specific APIs to evolve for different research communities, e.g., to better support defect prediction.

**Encode process information.** Process information "includes release schedule (milestones and dates), organization of code bases, team structure, and so on" (Czerwonka et al 2013). Czerwonka et al. advise, that these information should be embedded into the platform's data store. For open source projects, this information is often not available or unclear. Still, the collection of such information for open-source projects would be valuable for SmartSHARK and will be part of future work.

**Provide flexibility and extensibility for collected data and deployed analytics.** At Microsoft, the requirements on such a platform in respect to the collected data or deployed analytics varies between product teams. We found this lesson to be very important for SmartSHARK, because this also holds true for our research community. Deployed analytics can be of any kind. Therefore, we designed the database to be as flexible as possible (NoSQL key/value storage), allow with Apache Spark all kinds of analytic programs, and allow a wide variety of different data collection plug-ins with our improved version (see: Section 7).

**Allow dynamic discovery of data platform's capabilities by application.** Czerwonka et al (2013) describe, that the platform should offer some kind of interface through which the available data can be identified by the analytical applications. This lesson was partially followed. While SmartSHARK does not provide a sandbox where researchers can test the capabilities, the incremental development of Apache Spark jobs allows them to experience the capabilities and test the limits. Furthermore, we are currently working on a specialized API, which allows to identify the available data that is stored in the platform.

**Support policies for security, privacy, and audit.** The platform must allow the setting of authorization, authentication, and other security measures to secure the data and the access, as Czerwonka et al (2013) point out. While SmartSHARK has a basic user access rights system, the current implementation is not enough and more diverse access rights are required, e.g., to restrict writing access only on parts of the database.

**Allow ongoing support and maintenance outside of CODEMINE.** At Microsoft, product teams take over the ownership and operation of a data platform instance. Therefore, the data platform and the offered services must be well defined. With our design of SmartSHARK, we tried to implement this experience. We know, that for acceptance in the research community it is mandatory to allow maintenance outside of SmartSHARK. SmartSHARK is open-source and we invite other researchers to work together with us on the extension of the prototype.

**Host as a cloud service.** Czerwonka et al (2013) suggest, that one should determine if such a platform should be hosted as a cloud service or on traditional servers, based on their needs for availability, load, etc. Following this lesson, we designed SmartSHARK as a cloud platform to be easily scalable in terms of computational and storage resources, as we believe that especially very complex analytic jobs are highly resource demanding.

**Know the data platform might not fulfill all data needs.** The platform might not satisfy all users, as some data might be not available for a certain analytics. For SmartSHARK we used this lesson as motivation to create an extensible approach for the ETL process. This way, if the platform is lacking in terms of data, it is designed in a way that the problem can be mitigated by extending the ETL process.

**Innovate at the right level of the stack.** Czerwonka et al. advise, that you should only use mature foundational technology as much as possible. For a platform aimed at researchers, like SmartSHARK, this lesson is only partially applicable, because it is part of research to innovate and test new technologies. However, due to the high level of integration between tools within SmartSHARK, we also think that a certain level of quality and maturity is required for everything that shall be part of the main branch of SmartSHARK.

### 6.3 Lessons from SmartSHARK

Because of the different focus of SmartSHARK on research instead of the support of product teams, as well as due to the difference in available resources for researchers in comparison to a company like Microsoft, we learned the following lessons.

**(L1) Use only mature tools which you can maintain by yourselves.** The development of SmartSHARK was challenging due to the quality and maturity of the tools used. For the ETL of the VCS data, we relied on the popular CVSAnalY. However, the tool once stopped for an extended period of time, which is problematic for a fully automated data collection process, where human intervention should be the exception and not the rule. However, since CVSAnalY is open source, we could pin point the source of the problem and can provide a solution in the future. For the proprietary InFamix on the other hand, we do not know the source of the failure for Mahout and have no means of fixing this problem. Additionally, InFamix is not available anymore, which means that no fix will be available in the future, leaving no choice but switching the tool.

**(L2) Model-based fact extraction works, but is very resource demanding.** SmartSHARK is based on DECENT, a model-based fact extraction and transformation framework (see Section 4.1.1). DECENT provided a good foundation for the integration of information. However, the created EMF model must fit completely into the RAM and therefore sufficient resources need to be available. This a threat to the scalability of the data collection, as the memory requirements of very large projects like the Linux kernel or Firefox are high. Nevertheless, there are solutions to this problem, such as Scheidgen et al (2012) and Benelallam et al (2014). Scheidgen et al (2012) showed, that it is possible to transparently fragment the different models and store these fragments, e.g., in a MongoDB. NeoEMF[29] (Benelallam et al 2014) has

[29] http://www.neoemf.com

evolved into a mature, efficient, and scalable multi-database storage solution, supporting on-demand loading and powerful querying for large EMF models.

**(L3) Support optimized database queries.** Since the amount of collected data grows rapidly with the number of projects, well-formulated database queries are required in order to keep the burden on the storage back-end and internal network traffic in the cloud low. Therefore, an API that already supports the most important database queries is important for a large-scale deployment with thousands of projects.

**(L4) Heavy resource demand on the infrastructure.** Even taking the above lessons into account, we experienced that our setup with four VMs is the bare minimum at which we can get SmartSHARK to run somewhat decently. For a large deployment that should be able to mine thousands of projects a larger cloud infrastructure is required with multiple worker nodes for the project data collection and a dedicated database back-end that is separated from the web-front-end.

**(L5) Allow visualization plug-ins.** While Apache Spark is a good solution for non-visual analytics, it does not support visual analytics, as we described in the experience report. To this aim, a good and very flexible plug-in system is required, that allows, e.g., visualization of trends, social networks, and dependencies within projects.

**(L6) Provide externally accessible APIs.** An enhanced API for job submission, which could be accessed via a script, directly from an IDE, or from within other applications (e.g., as part of other Java applications) could help to further improve the writing of analytic jobs. Moreover, an API that makes log information, which was collected during the execution of jobs, visible to users would offer another active feedback loop besides the visualizations. These APIs are on our list for future work.

## 7 Scaling Up the Prototype

Through the experience that we gathered with the SmartSHARK version presented in Section 4 and the gathered feedback we updated our platform. The update does not only change parts in the implementation, but also the design and focus of the platform is changed to address the feedback and several issues that we uncovered (see: Section 5).

### 7.1 Design

We want to address the issue, that only data from one project could be collected at a time and take care of the lessons that we have learned with our first version. Therefore, we changed our database design and added indexes to support optimized database queries.

Furthermore, we re-designed our infrastructure. In Figure 12 our new design is depicted. This new design was created based on lesson L4, as we have

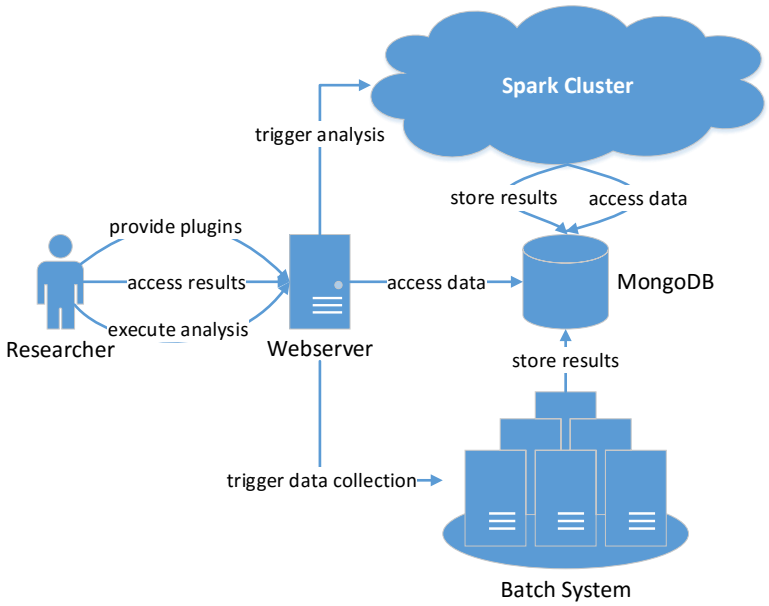


**Fig. 12** Improved SmartSHARK design.

experienced that front-end, database, and data collection should be physically and logically separated from each other. In contrast to Figure 1, which only gives a broad overview of the old system, Figure 12 includes details about the infrastructure and the implementation of the conceptual design. The central access point for the researcher is the webserver, where she can provide plugins, access the analysis results, and execute the analysis jobs. Furthermore, we separated the data collection from the data analysis: the data collection is executed on a batch system, which can be, e.g., a High Performance Computing (HPC) cluster while the data analysis is still executed on a Spark cluster. The plug-in that is executed on the batch system stores its results in the MongoDB, which can than be accessed by the analysis job that runs on the Spark cluster. Hence, this design provides us with a good scalability, enables us to collect data from more than one project at once (depending on the capability of the batch system), and addresses the heavy resource demand of the data collection.

This is different to our first design, as we have a clearer separation between data collection and data analysis. This leads to a wide variety of possible data collection plug-ins, as the batch system is not fixed to one programming language. Another central part in the new design is the MongoDB, which should be deployed in a distributed manner to cope with the high load. All results are stored in the MongoDB, while it is regularly accessed by the analysis jobs. In our first design, the webserver, MongoDB, and data collection server are placed on one single node, which did not scale well.

### 7.2 Implementation

As it is stated in Section 5.1 we had four different problems during the execution of our data collection process, which we wanted to address by developing new programs that can exchange the ones we used before. Three of these problems were connected to the tools that we used to execute it. The first problem was the unreliability of CVSAnalY, as it stopped working for the oryx project[30]. Furthermore, InFamix did not work for all revisions of Mahout, why we needed to exclude it from our first report. Additionally, with InFamix we were only able to collect source code metrics for one programming language per project, as it does not have multiple language support.

We addressed these issues and our lesson L1, by developing two new data collection plug-ins for SmartSHARK: vcsSHARK[31] and mecoSHARK[32]. The vcsSHARK is a tool that is able to collect the complete commit history of a project including, e.g., all commits on all branches, tags, differences between file revisions, and changed files. The collected data is then stored in a database. Currently, the vcsSHARK is able to collect data from projects, which use git as VCS[33] and stores the data in a MongoDB. The vcsSHARK's design is modular and offers a plug-in system that allows additional VCSs (e.g., Subversion (SVN) or Mecurial) or data storages (e.g., MySQL).

During the development of vcsSHARK we compared its results with the data that we got via CVSAnalY and the data that is stored on GitHub using its web interface. We tested the vcsSHARK by comparing the collected data with the data we got from CVSAnalY, as well as the original data on GitHub using GitHub's web interface. Through this procedure, we discovered another bug in CVSAnalY, which sometimes caused commits to be placed in the wrong branch of the extracted data.

We discovered, that the data collected by vcsSHARK matched the data that is stored on GitHub, but the results of CVSAnalY are different to it. E.g., in CVSAnalY some commits are placed on other branches than it is displayed at GitHub. Because of this observation and the first problem mentioned above, we exchanged CVSAnalY with the vcsSHARK in our new data collection process.

The mecoSHARK is a tool that is able to collect very detailed metrics of files in a project. Currently, it works with Java and Python projects. It checks out every revision of the project, automatically detects the used programming languages, executes the metric calculation plug-ins that correspond to the used programming languages, executes a clone detection algorithm[34],

[30] This problem does not occur anymore with the current version of CVSAnalY.

[31] `https://github.com/smartshark/vcsSHARK`

[32] `https://github.com/smartshark/mecoSHARK`

[33] We use the official git library for analysis: `https://libgit2.github.com/`

[34] Currently, mecoSHARK is able to detect Type-2 clones, which are clones that are syntactically identical except for variations in layout, comments, whitespaces, type references, identifier names, and literals. Details can be found in the SourceMeter documentation: (FrontEndART Ltd. 2016a)

and stores the collected data in a MongoDB. The programming language is detected with sloccount (Wheeler 2004), the metric calculation itself is performed by SourceMeter (FrontEndART Ltd. 2016c), a free to use metric calculation and clone detection tool. The mecoSHARK's design is also modular like the vcsSHARK and can be extended via plug-ins, e.g., to use other tools than SourceMeter for metric calculation. This allows the extension of mecoSHARK for additional programming languages, which can be measured with other tools. Table 4 lists all currently calculated metrics for Java and Python projects. These metrics are calculated on different levels of abstraction, e.g., class, method, and/or function level. Furthermore, the table shows which of these metrics are also available in InFamix. Note, that InFamix does not calculate clone metrics. Table 5 reports all clone metrics that are calculated with mecoSHARK. A exact description of each metric is given in the documentation of SourceMeter (FrontEndART Ltd. 2016a,b).

**Table 4** Calculated source code metrics for mecoSHARK (Java/Python) and InFamix.

| Category | Metric Name | Java | Python | InFamix |
|---|---|---|---|---|
| Cohesion | Lack of Cohesion in Methods 5 | x | x | |
| | Lack of Cohesion in Methods | | | x |
| | Tight Class Cohesion | | | x |
| Complexity | Access to Foreign Data | | | x |
| | Access to Local Data | | | x |
| | Average Method Weight | | | x |
| | Class Weight | | | x |
| | Dispersion Ratio | | | x |
| | McCabe's Cyclomatic Complexity | x | x | x |
| | Nesting Level | x | x | x |
| | Nesting Level Else-If | x | x | |
| | Specialisation Index | | | x |
| | Weighted Methods per Class | x | x | |
| Coupling | Base Class Usage Ratio | | | x |
| | Capsules Providing Foreign Data | | | x |
| | Coupling Between Object classes | x | x | x |
| | Coupling Between Object classes Inverse | x | x | |
| | Incoming Coupling Dispersion for an Operation | | | x |
| | Incoming Coupling Intensity for an Operation | | | x |
| | Locality of Data Accesses | | | x |
| | Loose Class Cohesion | | | x |
| | Number of Incoming Invocations | x | x | |
| | Number of Outgoing Invocations | x | x | |
| | Outgoing Coupling Dispersion for an Operation | | | x |
| | Outgoing Coupling Intensity for an Operation | | | x |
| | Response set For Class | x | x | x |
| Documentation | API Documentation | x | | |
| | Comment Density | x | x | |
| | Comment Lines of Code | x | x | |
| | Documentation Lines of Code | x | x | |

Continued on next page

**Table 4 – continued from previous page**

| Category | Metric Name | Java | Python | InFamix |
|---|---|---|---|---|
| | Public Documented API | x | | |
| | Public Undocumented API | x | | |
| | Total API Documentation | x | | |
| | Total Comment Density | x | x | |
| | Total Comment Lines of Code | x | x | x |
| | Total Public Documented API | x | | |
| | Total Public Undocumented API | x | | |
| Inheritance | Base Class Overriding Ratio | | | x |
| | Depth of Inheritance Tree | x | x | x |
| | Number of Ancestors | x | x | |
| | Number of Children | x | x | x |
| | Number of Descendants | x | x | |
| | Number of Parents | x | x | |
| Size | Lines of Code | x | x | x |
| | Logical Lines of Code | x | x | |
| | Number of Abstract Methods | | | x |
| | Number of Accessed Variables | | | x |
| | Number of Accessor Methods | | | x |
| | Number of Added Services | | | x |
| | Number of Attributes | x | x | x |
| | Number of Classes | x | x | |
| | Number of Enums | x | | |
| | Number of Getters | x | | |
| | Number of Interfaces | x | | |
| | Number of Local Attributes | x | x | |
| | Number of Local Getters | x | | |
| | Number of Local Methods | x | x | |
| | Number of Local Public Attributes | x | | |
| | Number of Local Public Methods | x | | |
| | Number of Local Setters | x | | |
| | Number of Methods | x | x | x |
| | Number of Outgoing Calls | | | x |
| | Number of Overriding Methods | | | x |
| | Number of Packages | x | x | |
| | Number of Parameters | x | x | x |
| | Number of Protected Attributes | | | x |
| | Number of Protected Methods | | | x |
| | Number of Public Attributes | x | | x |
| | Number of Public Methods | x | | x |
| | Number of Setters | x | | |
| | Number of Statements | x | x | |
| | Percentage of Newly Added Services | | | x |
| | Total Lines of Code | x | x | |
| | Total Logical Lines of Code | x | x | |
| | Total Number of Attributes | x | x | |
| | Total Number of Classes | x | x | |
| | Total Number of Directories | x | x | |
| | Total Number of Enums | x | | |
| | Total Number of Files | x | x | |
| | Total Number of Getters | x | | |
| | Total Number of Interfaces | x | | |
| | Total Number of Local Attributes | x | x | |

Continued on next page

**Table 4 – continued from previous page**

| Category | Metric Name | Java | Python | InFamix |
|---|---|---|---|---|
| | Total Number of Local Getters | x | | |
| | Total Number of Local Methods | x | x | |
| | Total Number of Local Public Attributes | x | | |
| | Total Number of Local Public Methods | x | | |
| | Total Number of Local Setters | x | | |
| | Total Number of Methods | x | x | |
| | Total Number of Packages | x | x | |
| | Total Number of Public Attributes | x | | |
| | Total Number of Public Classes | x | | |
| | Total Number of Public Enums | x | | |
| | Total Number of Public Interfaces | x | | |
| | Total Number of Public Methods | x | | |
| | Total Number of Setters | x | | |
| | Total Number of Statements | x | x | |
| | Weighted Operation Count | | | x |

Through the exchange of InFamix in favor of mecoSHARK we were able to collect metrics for Mahout and we are also able to collect source code metrics for different programming languages in one project, as we automatically determine the languages that are used and execute the corresponding mecoSHARK plugins. Furthermore, we have more metrics available on different levels of abstraction than with InFamix.

We measured the execution times of the vcsSHARK and the mecoSHARK in comparison to the previous version of the prototype based on CVSAnalY and InFamix. To this aim, we executed all tools on a machine with 8 Intel Xeon cores (3.6GHz each) and 16GB of RAM, which is similar to the old deployment of SmartSHARK. Additionally, we measured the execution time of the

**Table 5** Calculated clone metrics by mecoSHARK.

| Metric Name | Java | Python |
|---|---|---|
| Clone Age | x | x |
| Clone Classes | x | x |
| Clone Complexity | x | x |
| Clone Coverage | x | x |
| Clone Elimination Effort | x | x |
| Clone Elimination Gain | x | x |
| Clone Embeddedness | x | x |
| Clone Instances | x | x |
| Clone Line Coverage | x | x |
| Clone Lines of Code | x | x |
| Clone Logical Line Coverage | x | x |
| Clone Risk | x | x |
| Clone Variability | x | x |
| Lines of Duplicated Code | x | x |
| Logical Lines of Duplicated Code | x | x |
| Normalized Clone Radius | x | x |

**Table 6** Used nodes of the HPC cluster

| #Nodes | CPU | Cores | Frequency | Memory |
|---|---|---|---|---|
| 5 | Magny-Cours, AMD Opteron 6174 | 4x12 | 2.20 GHz | 128GB |
| 48 | Abu-Dhabi, AMD Opteron 6378 | 4x16 | 2.40 GHz | 256GB |
| 1 | Haswell, Intel E7-4809 v3 | 4x8 | 2.00 GHz | 2TB |
| 15 | Broadwell, Intel E5-2650 v4 | 2x12 | 2.20 GHz | 512GB |
| 5 | Haswell, Intel E5-4620 v3 | 4x10 | 2.00 GHz | 1.5TB |

vcsSHARK and mecoSHARK in a HPC cluster as batch system. The different nodes and their capabilities of the used HPC cluster are listed in Table 6. Note, that we could not reserve the whole HPC cluster for our measurements, as it is a cluster available for all researchers from Göttingen University and hosted by our datacenter[35]. Therefore, the results may vary due to different work loads. Note, that we needed to change our plug-ins to enable a nearly fair comparison. First, we excluded the calculation of file differences between revisions for the vcsSHARK, as CVSAnalY does not provide this data. Furthermore, we excluded the last step of mecoSHARK (saving the data in the MongoDB) for this analysis, as this step is not done by InFamix, but by an additional transformation script based on DECENT (see Section 4.1.1). Instead, we measured the time required only for the calculation of metrics values for mecoSHARK.

We needed to exclude the project "Minesweeper" that we have used in our earlier work (Trautsch et al 2016), as it is no longer available on GitHub. Additionally, for the comparison of mecoSHARK to InFamix, we could only use Java projects, as mecoSHARK currently only supports Python and Java projects.

Figure 13 compares the execution time of CVSAnalY to the vcsSHARK. CVSAnalY is faster for all but two projects. We see several reasons for this: first, our plug-in is designed for bigger projects and heavy computation load, as we spawn different processes. If we exclude the only task that takes computational time (comparing file revisions) the different processes block each other, which results in a slower run time. Furthermore, as we have explained above, CVSAnalY does not provide as much data as the vcsSHARK (e.g., comments on tags) and the data that CVSAnalY collects is not equal to the data which GitHub provide via their web interface.

Furthermore, vcsSHARK on the new deployment (HPC cluster) is the slowest among the three tested plug-ins/setups. We see several reasons for this: first, we could not control for the load of the HPC cluster. Therefore, this could have an influence on the run time. Another possible reason is the overhead in the communication. CVSAnalY, as well as vcsSHARK in the old deployment were communicating with a database, which was running locally on the machine that also executed the plug-ins. In case of the vcsSHARK, executed on our new deployment, this was not possible. Therefore, all results needed to be communicated through the network to the database server. Nev-

[35] `http://gwdg.de`

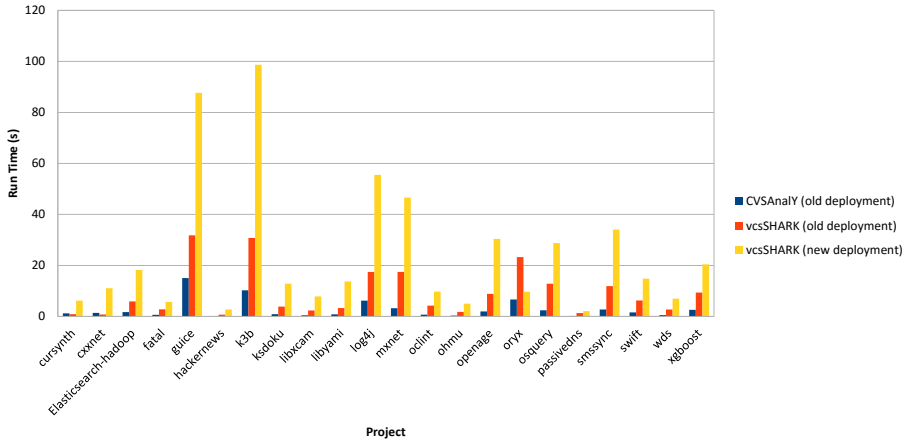


**Fig. 13** Comparison of run times of CVSAnalY (single node), vcsSHARK (single node), as well as vcsSHARK running on the HPC cluster.

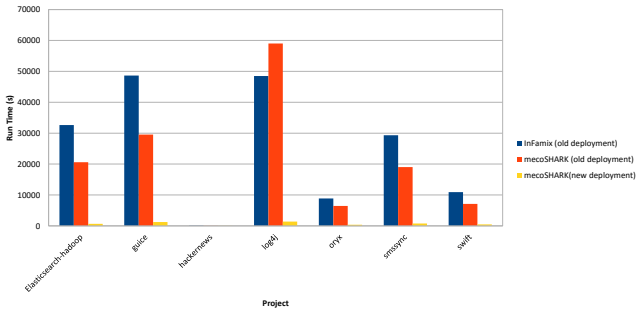


**Fig. 14** Comparison of run times of InFamix (single node), mecoSHARK (single node), as well as mecoSHARK running on the HPC cluster.

ertheless, we note that all run times are below 100 seconds regardless of the deployment and whether we use CVSAnalY or the vcsSHARK. Hence, both tools as well as deployment scale well.

Figure 14 compares the execution time of InFamix and mecoSHARK. InFamix is slower than our newly developed mecoSHARK for every project except log4j. The reason for the difference for log4j is unclear and requires further investigation. The mecoSHARK, executed on our new deployment, is up to 50 times faster than InFamix and up to 42 times faster than mecoSHARK on the old deployment (depending on the project). The reason for this massive speed up is that we have an embarrassingly parallel workload: if we are using the HPC cluster, several jobs can be submitted (for each revision one job), which executes the plug-in on the given revision. This results in a massive parallelism and in a drastic reduction of the execution time.

Our comparison shows that we get a massive improvement of the run times through our new deployment, if the plug-ins needs to be executed for each

**Table 7** #Documents and used storage space in the MongoDB after vcsSHARK was executed. The results are reported for each project separately.

| **Project** | **#Documents (vcsSHARK)** | **Storage (vcsSHARK)** |
|---|---|---|
| cursynth | 3.502 | 3,131 MB |
| cxxnet | 10.797 | 9,603 MB |
| elasticsearch-hadoop | 30.336 | 14,785 MB |
| fatal | 17.550 | 12,501 MB |
| guice | 195.043 | 790,116 MB |
| HackerNews | 1.144 | 0,560 MB |
| k3b | 187.771 | 161,092 MB |
| ksudoku | 15.599 | 12,28 MB |
| libxcam | 11.440 | 6,222 MB |
| libyami | 22.421 | 14,962 MB |
| log4j | 85.790 | 57,182 MB |
| mxnet | 129.532 | 90,622 MB |
| oclint | 18.770 | 6,973 MB |
| ohmu | 10.336 | 6,361 MB |
| openage | 60.953 | 26,923 MB |
| oryx | 11.120 | 6,513 MB |
| osquery | 79.326 | 48,743 MB |
| passivedns | 3.013 | 1,209 MB |
| SMSSync | 49.192 | 38,453 MB |
| swift | 32.744 | 44,713 MB |
| wds | 13.381 | 4,451 MB |
| xgboost | 56.796 | 64,34 MB |
| **Overall** | 1.046.556 | 1.421,735 MB |

revision. Furthermore, it shows that our developed mecoSHARK is faster than InFamix in most cases, whereas the vcsSHARK is slower than CVSAnalY, but provides more data.

Additionally to the execution time we also measured the used storage space for each project and for each plugin. Tables 7 and 8 depict the number of documents that are created in the MongoDB for each plugin and the storage space that these documents take. Please note, that the concrete used storage space is lower, as MongoDB optimizes the storing of the documents. Furthermore, as we also stated above, the project Minesweeper is no longer available and therefore we could not gather data using our new plugins. Tables 7 and 8 highlight, that the bigger the projects the more documents are created and the more storage space is needed. Furthermore, it also highlights the differences between both of our plugins in respect to the needed storage space: vcsSHARK needs less storage space as mecoSHARK. The reason for this is the difference in the stored data. vcsSHARK stores mostly meta-data about the commits of a project, but also the concrete textual change that was made. But, mecoSHARK stores one document for every code entity (e.g., class, method, interface) for every commit together with its calculated metric data. Therefore, a lot more storage space is used.

**Table 8** #Documents and used storage space in the MongoDB after mecoSHARK was executed. The results are reported for each project separately.

| **Project** | **#Documents (mecoSHARK)** | **Storage (mecoSHARK)** |
|---|---|---|
| elasticsearch-hadoop | 7.611.833 | 4.878,285 MB |
| guice | 13.574.467 | 10.012,43 MB |
| HackerNews | 9.220 | 6,14 MB |
| log4j | 22.442.724 | 13.747,25 MB |
| oryx | 2.060.098 | 1.368,7 MB |
| SMSSync | 6.107.692 | 3.824,99 MB |
| swift | 2.591.012 | 1.727,42 MB |
| **Overall** | 54.366.710 | 35.550,43 MB |

### 7.3 Further Extensions of the Prototype

In addition to the new vcsSHARK and mecoSHARK, which just replaced previously already existing functionality in the prototype, we also developed the issueSHARK[36] and the coastSHARK[37]. The issueSHARK is able to collect data from ITSs (e.g., issue title, description, changes on the issue, etc.) and currently supports GitHub Issues, Bugzilla, and Jira. The coastSHARK generates the ASTs for Java (or Python) files and gathers the AST node counts for all files. This enables the custom definition of new metrics. Count-based metrics can be calculated directly from the MongoDB, structural metrics could be added by extending the coastSHARK. Both plug-ins further demonstrate the extensibility of our prototype. The complete available data of the current state of the prototype is documented online[38].

That such data is useful for research is without question. For example, the complete body of work on software defect prediction, which encompasses hundreds of publications (see, e.g., the surveys by (Catal and Diri 2009), (Hall et al 2012), and (Herbold 2017)), can be supported by such a prototype as demonstrated in Section 5.2.2. Additionally, natural language processing tasks like topic modeling require the commit messages, issue comments or mailing list contents, which are all already available, covering another popular research topic (Sun et al 2016). Moreover, work on developer social networks, based on commonly changed files, ITS information, as well as emails is supported. While we unfortunately cannot cite a survey to demonstrate the impact of this topic, one of the early papers by Bird et al (2006) was already cited 512 times[39]. All of these are high-impact research topics, for which now new data can be continuously generated and shared using SmartSHARK.

The extensible nature of our platform allows us to further grow the number of possible applications. Since we have no lock-in regarding the data collection technology and basically allow command line tools, only very little overhead is

[36] https://github.com/smartshark/issueSHARK
[37] https://github.com/smartshark/coastSHARK
[38] http://smartshark2.informatik.uni-goettingen.de/documentation/
[39] According to Google Scholar on 2017-07-06.

created for developers of data collection tools, if they want to extend SmartSHARK. They only need to make sure that they store the data compliant to the already available data to avoid redundancies and ensure a consistent schema. As a benefit, they then get the integration with the other data collection plug-ins. As for the analysis of the data, we already support Spark as a powerful analysis engine. Depending on the demands by the community, it would be possible to open this up and allow more technologies, e.g., native Python or R, without using Spark as a backend. For the visualizations, we already demonstrated the versatility in Section 5.2.1 based on JavaScript using HighCharts and visJS. Together with other libraries like D3.js[40] JavaScript supports many kinds of graphical representation of data. Still, we are aware that our platform will never be able to support all MSR research, e.g, research that also requires developer surveys. However, we are confident that such a large, diverse, and extensible database coupled with an analysis frontend that allows sharing implementations can help researchers produce replicable research results that do not suffer from the problems with external validity we outline in this article.

### 7.4 Limitations and Future Work

The current version of SmartSHARK is able to execute different plug-ins (e.g., our vcsSHARK or mecoSHARK) for an arbitrary git-using project. Furthermore, we can analyze the data via Spark. However, there are some limitations of our platform on which we are working at the moment.

When it come to the non-availability of data sets and implementations, SmartSHARK by itself cannot tackle this problem. Nevertheless, we alleviate this issue by providing a tool, which supports the sharing of data and implementations. Furthermore, a tool such as SmartSHARK could be integrated into the existing publication process. E.g., researchers that submit a paper must provide their implementation (for SmartSHARK: a Apache Spark job) and access to their SmartSHARK instance. Hence, in a peer-review process, reviewers can re-run the jobs on the opened SmartSHARK instance and evaluate the results and the used data. Moreover, it would be possible to compare the results with other available approaches for which Spark jobs are existing. These jobs could be acquired by the catalogs that we mentioned in Section 6. Hence, reviewers could run these jobs to compare the results on the same data set.

Furthermore, for solving the problem of the non-availability of implementations our goal is just partly fulfilled. Currently, only Spark jobs and data collection plug-ins can be shared, but visualization plug-ins are not part of the platform at the moment. Additionally, we want to provide the catalogs, that were mentioned in Section 6. But for this, the general design and structure must be fixed and evaluated by a larger group of users.

[40] https://d3js.org/

Moreover, the problem of too small data sets only applies for some approaches. Empirical software engineering is not only about analyzing big data, but there exist a lot of papers that base their approaches on, e.g., surveys, questionnaires, interviews with developers, etc., like in (Smith et al 2016). This data is called "thick data" and such studies can be supported by SmartSHARK (e.g., by complementing survey results with mined project data like in (Devanbu et al 2016)), but a full replication only using SmartSHARK is not possible. Because such data (e.g., survey data) is not stored or connected within the platform.

For the diverse tooling, we also see some limitations. First of all, one essential step in analyzing data, such as the data collected by SmartSHARK, is understanding and processing it. SmartSHARK supports the understanding of data by providing several visualizations (see: Section 5.2.1), but this can not be automated fully as human interaction is mandatory. Furthermore, the processing (e.g., cleaning) of data is only supported via Apache Spark jobs. Hence, SmartSHARK by itself only collects raw data (e.g., metrics data from files, commits from the VCS) and stores it in the MongoDB. If the data should be analyzed or used in an approach, it must be cleaned and pre-processed first. This step belongs to the analysis step and must be therefore put into the Apache Spark job that can be shared later on.

For future work, we plan to perform an evaluation of SmartSHARKs capabilities with students of the courses that we teach. We want to evaluate how novice MSR researchers can make use of SmartSHARK. This does not only include an usability analysis, but we want to get feedback regarding SmartSHARKs capabilities especially from novice researchers, as it is hard to get into MSR research, because of the complex working environment that needs to be built up beforehand.

Furthermore, we are planing to improve our ETL process. First of all, we want to pre-process the data by combining different user identities. Moreover, we plan to classify done commits, i.e., by putting commits into different classes like "bugfix", "bug inducing", etc.).

We described several improvements for SmartSHARK that we are currently planning. But, for SmartSHARK to be able to significantly help in replicating a large portion of papers, we need to improve our plug-in development process. This improvement, together with the planned catalogs and the promotion of the platform, is essential to make this platform interesting for a larger set of researchers and will result in more plug-ins developed for the platform. Hence, one focus is the improvement of the usability of the platform, as we will not be able to provide plug-ins for every kind of data collection or analysis task by ourselves. We plan an ecosystem, which is developed around SmartSHARK together with the community.

## 8 Threats to Validity

### 8.1 Internal Validity

The main threat to the internal validity of our work is an inappropriate choice of collected data and analysis techniques. In case the data we collected is not of interest for the community, this would invalidate all of our findings and insights related to the data collection and combination. Similarly, in case our analytic examples are inappropriate, our conclusions regarding the usability of the platform may be invalid, especially in terms of required libraries. However, to our mind these threats are minimal, as both metric data as well as historic data about the usage of GitHub are frequently used in MSR research and visualizations, defect prediction, and effort predictions have been of constant interest to researchers in the past decade.

Furthermore, the categorization of papers, as done in Section 3, is a manual task and, by the nature of it, there can be errors. Nevertheless, Section 3 is not meant to be an exhaustive, systematic literature review, but it should only show a trend and that the problems mentioned in Section 1 are still topical.

### 8.2 External Validity

Threats to the external validity are concerned with the ability to generalize results. In our analysis we used SmartSHARK to collect data from 23 different projects. Although we have chosen projects, which have a wide variety of characteristics (e.g., size, number of revisions, field of application), the results may vary if other projects are chosen.

Furthermore, for our analysis of the current state of practice we only have analyzed the ICSE, MSR, ESEC/FSE, and ESEM from 2015. The results may vary if other conferences or journals are chosen. Additionally to that, SmartSHARK, in its current state, is not able to replicate all of the papers mentioned in Section 3. For that, more plug-ins for the data collection and for the analysis need to be written by the research community and implemented in SmartSHARK to support the replication of various approaches.

### 8.3 Construct Validity

Construct threats are concerned with the degree to which our analysis really analyses what we are claiming it does. We developed different tools, which we connected to build up our platform. We carefully tested these tools with manually written tests and manually-curated data samples. Nevertheless, defects in our program can still exist, which might have an influence on the results presented in this paper.

Furthermore, our usability evaluation was done only by the authors of this paper and, therefore, may be biased. Hence, an usability evaluation needs to

be done with more and unbiased people to get additional insights regarding the usability of SmartSHARK, e.g., students or other researchers.

Additionally, we compared the run time of our plugins with established programs like inFamix or CVSAnalY. In this comparison, we tried to create an environment which enables a fair comparison. Nevertheless, a complete fair comparison can not be established, as the compared tools are different in, e.g., the number of metrics they calculate or data they fetch and process.

## 9 Conclusion

Within this paper, we discussed how a platform that combines automated data collection with a flexible analytic front-end can help to solve the problems of insufficient replicability and problematic validity of repository mining studies.

To show that these problems are still current, we analyzed the publications of the ICSE 2015, MSR 2015, ESEC/FSE 2015, and ESEM 2015 in respect to the identified problems. Our analysis showed, that these problems still exist nowadays.

To address these problems, we created the platform SmartSHARK, which is inspired by CODEMINE (Czerwonka et al 2013). Due to the differences between research and industrial exploitation of such a platform, we focused our evaluation on three aspects: 1) we showed through the definition of visualizations, defect prediction, and effort analysis, based on two different data sources (VCS and mailing lists) that SmartSHARK can be used to define analytics; 2) we outlined how SmartSHARK can help to improve the replicability and validity of studies; and 3) we gave insights on research-specific lessons learned that are important for building such a platform that were not addressed by the proprietary CODEMINE. SmartSHARK is open-source and we invite all interested researchers to contribute to our platform in order to go from a proof-of-concept prototype, to a reliable and valuable tool for the research community that solves some of the problems we are having. We moved all our development infrastructure to GitHub to enable others to contribute easily[41].

Furthermore, we described our improvements of the platform based on our experience report and comments from other researchers. We adapted our design and our implementation and improved our data collection process to address the raised issues.

**Acknowledgements** The authors would like to thank Fabian Glaser, Michael Göttsche, and Gunnar Krull for their support regarding the cloud technologies and the deployment as well as the GWDG for allowing us to use their HPC resources.

[41] https://github.com/smartshark

## References


Alexandru CV, Gall HC (2015) Rapid Multi-Purpose, Multi-Commit Code Analysis. In: Proceedings of the IEEE/ACM 37th International Conference on Software Engineering (ICSE), IEEE/ACM, pp 635–638

Alliance O (2007) Osgi service platform, core specification, release 4, version 4.3. https://www.osgi.org/release-4-version-4-3/

Arcuri A, Fraser G, Galeotti JP (2015) Generating tcp/udp network data for automated unit test generation. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 155–165

Avdiienko V, Kuznetsov K, Gorla A, Zeller A, Arzt S, Rasthofer S, Bodden E (2015) Mining apps for abnormal usage of sensitive data. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 426–436

Bang L, Aydin A, Bultan T (2015) Automatically computing path complexity of programs. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 61–72

Benelallam A, Gómez A, Sunyé G, Tisi M, Launay D (2014) Neo4emf, a scalable persistence layer for emf models. In: Proceedings of the 10th European Conference on Modelling Foundations and Applications - Volume 8569, Springer-Verlag New York, Inc., New York, NY, USA, pp 230–241, DOI 10.1007/978-3-319-09195-2_15, URL http://dx.doi.org/10.1007/978-3-319-09195-2_15

Bevan J, Whitehead Jr EJ, Kim S, Godfrey M (2005) Facilitating software evolution research with kenyon. In: ACM SIGSOFT Software Engineering Notes, ACM, vol 30, pp 177–186

Beyer D, Dangl M, Dietsch D, Heizmann M, Stahlbauer A (2015) Witness validation and stepwise testification across software verifiers. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 721–733

Bird C, Gourley A, Devanbu P, Gertz M, Swaminathan A (2006) Mining email social networks. In: Proceedings of the 2006 international workshop on Mining software repositories, ACM, pp 137–143

Catal C, Diri B (2009) A systematic review of software fault prediction studies. Expert Systems with Applications 36(4):7346–7354

Cavalcanti G, Accioly P, Borba P (2015) Assessing semistructured merge in version control systems: A replicated experiment. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement 2015 (ESEM), IEEE, pp 1–10

Claes M, Mens T, Di Cosmo R, Vouillon J (2015) A historical analysis of debian package incompatibilities. In: IEEE/ACM 12th Working Conference on Mining Software Repositories 2015 (MSR), IEEE, pp 212–223

Coelho R, Almeida L, Gousios G, van Deursen A (2015) Unveiling exception handling bug hazards in android based on github and google code issues. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR), IEEE, pp 134–145

Čubranić D, Murphy GC, Singer J, Booth KS (2005) Hipikat: A project memory for software development. IEEE Transactions on Software Engineering 31(6):446–465

Czerwonka J, Nagappan N, Schulte W (2013) CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. IEEE Software 30(4):64–71

Devanbu P, Zimmermann T, Bird C (2016) Belief & evidence in empirical software engineering. In: Proceedings of the 38th international conference on software engineering, ACM, pp 108–119

Dhar A, Purandare R, Dhawan M, Rangaswamy S (2015) Clotho: saving programs from malformed strings and incorrect string-handling. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 555–566

Di Ruscio D, Kolovos DS, Korkontzelos I, Matragkas N, Vinju J (2015) Ossmeter: A software measurement platform for automatically analysing open source software projects. In: ESEC/FSE 2015 Tool Demonstrations Track

Di Sorbo A, Panichella S, Visaggio C, Di Penta M, Canfora G, Gall H (2015) Development emails content analyzer: Intention mining in developer discussions. In: Proceedings of the IEEE/ACM 30th International Conference on Automated Software Engineering (ASE)

Draisbach U, Naumann F (2010) Dude: The duplicate detection toolkit. In: Proceedings of the International Workshop on Quality in Databases (QDB)

Dyer R, Nguyen HA, Rajan H, Nguyen TN (2013) Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In: Proceedings of the IEEE/ACM 35th International Conference on Software Engineering (ICSE)

Dyer R, Nguyen HA, Rajan H, Nguyen T (2015) Boa: Ultra-Large-Scale Software Repository and Source Code Mining. ACM Transactions on Software Engineering and Methodology forthcoming

Eichberg M, Hermann B, Mezini M, Glanz L (2015) Hidden truths in dead software paths. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 474–484

Fernandez-Ramil J, Izquierdo-Cortazar D, Mens T (2009) What does it take to develop a million lines of open source code? In: Open Source Ecosystems: Diverse Communities Interacting, Springer, pp 170–184

Foucault M, Palyart M, Blanc X, Murphy GC, Falleri JR (2015) Impact of developer turnover on quality in open-source software. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 829–841

FrontEndART Ltd (2016a) SourceMeter Documentation for Java. `https://www.sourcemeter.com/resources/java`, [accessed 02-August-2016]

FrontEndART Ltd (2016b) SourceMeter Documentation for Python. `https://www.sourcemeter.com/resources/python`, [accessed 02-August-2016]

FrontEndART Ltd (2016c) SourceMeter Webpage. `https://www.sourcemeter.com/`, [accessed 02-August-2016]

Gallaba K, Mesbah A, Beschastnikh I (2015) Don't call us, we'll call you: Characterizing callbacks in javascript. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement 2015 (ESEM), IEEE, pp 1–10

German DM (2004) Mining CVS repositories, the softChange experience. Evolution 245(5,402):92–688

Giger E, Pinzger M, Gall H (2010) Predicting the fix time of bugs. In: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE), ACM, pp 52–56

Gong L, Pradel M, Sen K (2015) Jitprof: Pinpointing jit-unfriendly javascript code. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 357–368

González-Barahona JM, Robles G (2012) On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Empirical Software Engineering 17(1-2):75–89

Gousios G, Spinellis D (2009) Alitheia core: An extensible software quality monitoring platform. In: Proceedings of the IEEE/ACM 31st International Conference on Software Engineering (ICSE)

Gousios G, Spinellis D (2012) Ghtorrent: Github's data from a firehose. In: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR), IEEE, pp 12–21

Gousios G, Kalliamvakou E, Spinellis D (2008) Measuring developer contribution from software repository data. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories, ACM, New York, NY, USA, MSR '08, pp 129–132, DOI 10.1145/1370750.1370781, URL `http://doi.acm.org/10.1145/1370750.1370781`

Gousios G, Vasilescu B, Serebrenik A, Zaidman A (2014) Lean ghtorrent: Github data on demand. In: Proceedings of the 11th IEEE Working Conference on Mining Software Repositories (MSR), ACM, pp 384–387

Gupta M, Sureka A, Padmanabhuni S, Asadullah AM (2015) Identifying software process management challenges: Survey of practitioners in a large global it company. In: Proceedings of the 12th Working Conference on Mining Software Repositories, IEEE Press, pp 346–356

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. ACM SIGKDD Explorations Newsletter 11(1):10–18, DOI 10.1145/1656274.1656278

Hall T, Beecham S, Bowes D, Gray D, Counsell S (2012) A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering 38(6):1276–1304, DOI 10.1109/TSE.2011.103

He P, Li B, Liu X, Chen J, Ma Y (2015) An empirical study on software defect prediction with a simplified metric set. Information and Software Technology 59:170–190

Herbold S (2017) A systematic mapping study on cross-project defect prediction. CoRR abs/1705.06429, URL `https://arxiv.org/abs/1705.06429`, `arXiv:1705.06429`

Hermann B, Reif M, Eichberg M, Mezini M (2015) Getting to know you: Towards a capability model for java. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 758–769

Herraiz I, Robles G, Amor JJ, Romera T, González Barahona JM (2006) The processes of joining in global distributed software projects. In: Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner, ACM, pp 27–33

Herraiz I, Gonzalez-Barahona JM, Robles G (2007) Forecasting the number of changes in Eclipse using time series analysis. In: Proceedings of the 4th IEEE Working Conference on Mining Software Repositories (MSR)

Honsel V, Honsel D, Herbold S, Grabowski J, Waack S (2015) Mining Software Dependency Networks for Agent-Based Simulation of Software Evolution. In: Proceedings of the 4th International Workshop on Software Mining (SoftMine)

Howison J, Conklin MS, Crowston K (2005) Ossmole: A collaborative repository for floss research data and analyses. In: Proceedings of the 1st International Conference on Open Source Software

ISO/IEC (1998) 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs). ISO/IEC 9241-14

Jermakovics A, Sillitti A, Succi G (2011) Mining and visualizing developer networks from version control systems. In: Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), ACM, New York, NY, USA, CHASE '11, pp 24–31, DOI 10.1145/1984642.1984647, URL `http://doi.acm.org/10.1145/1984642.1984647`

Joblin M, Mauerer W, Apel S, Siegmund J, Riehle D (2015) From developer networks to verified communities: a fine-grained approach. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 563–573

Jorgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Transactions on Software Engineering 33(1):33–53, DOI 10.1109/TSE.2007.256943

Kouroshfar E, Mirakhorli M, Bagheri H, Xiao L, Malek S, Cai Y (2015) A study on the role of software architecture in the evolution and quality of software. In: Proceedings of the 12th Working Conference on Mining Software Repositories, IEEE Press, pp 246–257

Le DM, Behnamghader P, Garcia J, Link D, Shahbazian A, Medvidovic N (2015) An empirical study of architectural change in open-source software systems. In: Proceedings of the 12th Working Conference on Mining Software Repositories, IEEE Press, pp 235–245

Lin Z, Whitehead J (2015) Why power laws? an explanation from fine-grained code changes. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR), IEEE, pp 68–75

Linares-Vásquez M, Bavota G, Cárdenas CEB, Oliveto R, Di Penta M, Poshyvanyk D (2015a) Optimizing energy consumption of guis in android apps: A multi-objective approach. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 143–154

Linares-Vásquez M, White M, Bernal-Cárdenas C, Moran K, Poshyvanyk D (2015b) Mining android app usages for generating actionable gui-based execution scenarios. In: Proceedings of the 12th Working Conference on Mining Software Repositories, IEEE Press, pp 111–122

Long F, Rinard M (2015) Staged program repair with condition synthesis. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 166–178

Makedonski P, Grabowski J (2016) Weighted Multi-Factor Multi-Layer Identification of Potential Causes for Events of Interest in Software Repositories. In: Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution (SATToSE) 2015, forthcoming 2016

Makedonski P, Sudau F, Grabowski J (2015) Towards a model-based software mining infrastructure. ACM SIGSOFT Software Engineering Notes 40(1):1–8

Menzies T, Rees-Jones M, Krishna R, Pape C (2015) The promise repository of empirical software engineering data. http://openscience.us/repo. North Carolina State University, Department of Computer Science [accessed 22-January-2015]

Nanz S, Furia CA (2015) A comparative study of programming languages in rosetta code. In: IEEE/ACM 37th IEEE International Conference on Software Engineering 2015 (ICSE), IEEE, vol 1, pp 778–788

Nguyen HV, Kästner C, Nguyen TN (2015a) Cross-language program slicing for dynamic web applications. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 369–380

Nguyen TH, Grundy J, Almorsy M (2015b) Rule-based extraction of goal-use case models from text. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 591–601

Park J, Esmaeilzadeh H, Zhang X, Naik M, Harris W (2015) Flexjava: Language support for safe and modular approximate programming. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 745–757

Robles G (2010) Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings. In: Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, IEEE, pp 171–180

Robles G, González-Barahona JM, Cervigón C, Capiluppi A, Izquierdo-Cortázar D (2014) Estimating development effort in free/open source software projects by mining software repositories: A case study of openstack. In: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, New York, NY, USA, MSR 2014, pp 222–231, DOI 10.1145/2597073.2597107, URL http://doi.acm.org/10.1145/2597073.2597107

Safi G, Shahbazian A, Halfond WG, Medvidovic N (2015) Detecting event anomalies in event-based systems. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 25–37

Samak M, Ramanathan MK (2015) Synthesizing tests for detecting atomicity violations. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 131–142

Scheidgen M, Zubow A, Fischer J, Kolbe TH (2012) Automated and transparent model fragmentation for persisting large models. Springer

Shepperd M, Song Q, Sun Z, Mair C (2013) Data Quality: Some Comments on the NASA Software Defect Datasets. IEEE Transactions on Software Engineering 39(9):1208–1215

Shi A, Yung T, Gyori A, Marinov D (2015) Comparing and combining test-suite reduction and regression test selection. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 237–247

Shull FJ, Carver JC, Vegas S, Juristo N (2008) The role of replications in empirical software engineering. Empirical Software Engineering 13(2):211–218

Siegmund J, Siegmund N, Apel S (2015a) Views on internal and external validity in empirical software engineering. In: Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, IEEE, vol 1, pp 9–19

Siegmund N, Grebhahn A, Apel S, Kästner C (2015b) Performance-influence models for highly configurable systems. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 284–294

Sjøberg DI, Hannay JE, Hansen O, Kampenes VB, Karahasanovic A, Liborg NK, Rekdal AC (2005) A survey of controlled experiments in software engineering. Software Engineering, IEEE Transactions on 31(9):733–753

Smith EK, Barr ET, Le Goues C, Brun Y (2015) Is the cure worse than the disease? overfitting in automated program repair. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 532–543

Smith EK, Bird C, Zimmermann T (2016) Beliefs, practices, and personalities of software engineers: a survey in a large software company. In: Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering, ACM, pp 15–18

Sun X, Liu X, Li B, Duan Y, Yang H, Hu J (2016) Exploring topic models in software engineering data analysis: A survey. In: 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed

Computing (SNPD), pp 357–362, DOI 10.1109/SNPD.2016.7515925
Tan M, Tan L, Dara S, Mayeux C (2015) Online Defect Prediction for Imbalanced Data. In: Proceedings of the IEEE/ACM 37th International Conference on Software Engineering (ICSE)
Tao Y, Kim S (2015) Partitioning composite code changes to facilitate code review. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR), IEEE, pp 180–190
Thomas JJ, Cook KA (2006) A visual analytics agenda. IEEE computer graphics and applications 26(1):10–13
Trautsch F, Herbold S, Makedonski P, Grabowski J (2016) Adressing problems with external validity of repository mining studies through a smart data platform. In: Proceedings of the 13th International Workshop on Mining Software Repositories, ACM, pp 97–108
Van Rysselberghe F, Demeyer S (2004) Studying software evolution information by visualizing the change history. In: Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on, IEEE, pp 328–337
Walden J, Stuckman J, Scandariato R (2014) Predicting vulnerable components: Software metrics vs text mining. In: Proceedings of the IEEE 25th International Symposium on Software Reliability Engineering (ISSRE), IEEE, pp 23–33
Wheeler DA (2004) Sloccount Documentation. `http://www.dwheeler.com/sloccount/sloccount.html`, [accessed 02-August-2016]
Xu T, Jin L, Fan X, Zhou Y, Pasupathy S, Talwadker R (2015) Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 307–319
Yang W, Xiao X, Andow B, Li S, Xie T, Enck W (2015) Appcontext: Differentiating malicious and benign mobile app behaviors using context. In: IEEE/ACM 37th IEEE International Conference on Software Engineering 2015 (ICSE), IEEE, vol 1, pp 303–313
Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)
Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Network System Design and Implementation (NSDI)
Zhu J, He P, Fu Q, Zhang H, Lyu MR, Zhang D (2015) Learning to log: Helping developers make informed logging decisions. In: IEEE/ACM 37th IEEE International Conference on Software Engineering 2015 (ICSE), IEEE, vol 1, pp 415–425