Adressing Problems with External Validity of Repository Mining Studies Through a Smart Data Platform

Fabian Trautsch, Steffen Herbold, Philip Makedonski, Jens Grabowski
Institute for Computer Science, Georg-August-Universität Göttingen, Germany
{trautsch,herbold,makedonski,grabowski}@cs.uni-goettingen.de

ABSTRACT

Research in software repository mining has grown considerably the last decade. 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 external validity of results. 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. 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 the prototype SmartSHARK that implements our approach. Using SmartSHARK, we collected data from several projects and created different analytic examples. Within this article, we present SmartSHARK and discuss our experiences regarding the use of SmartSHARK and the mentioned problems.

CCS Concepts

•**Software and its engineering** → *Software evolution;*
•**Information systems** → *Data mining;*

Keywords

software mining, software analytics, smart data

1. INTRODUCTION

During the last decade, the collection of data from software repositories and the subsequent analysis of the data to perform software analytics became a highly topical research topic that spawned hundreds of publications. 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 [15], effort prediction [55], dependency analysis [50], developer social networks [54], and other topics related to the field of software evolution. The rise of this research area resulted in many tools for mining repositories [24, 25], data sets generated by researchers [64], analytics performed by researchers [20, 36], and insights into the process of software evolution [38, 47]. Due to the diversity of approaches, researchers face five major problems, when it comes to the external validity of the results.

(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 [45]. This allows good comparability between results, but poses a threat to the external validity, as, e.g., Shepperd et al. [70] 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 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. For a complex research proposal, a re-implementation can require a high amount of resources. Even for lower effort, 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 to the generalization of results.

(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 [66] for data mining or WEKA [61] for machine learning. These solutions range from prototypes

MSR'16, May 14-15, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2901739.2901753

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 proposed solution is a smart data platform for conducting experiments that provides a solution for both: data collection and storage, as well as flexible functionalities for data analytics. The smart data platform can serve as a common data store and thereby, directly enable researchers to share data sets, without additional effort, simply by using the platform. The integration of software analytics directly into the platform gives researchers a common ground for their implementations and thereby facilitates the sharing of approaches, without the problem of different environments for the analytics. Moreover, through automation of the data collection, constant maintenance of the underlying data, as well as the addition of new projects to the underlying database, problems due to heavy data re-use could be mitigated.

The Microsoft internal tool CODEMINE [18] demonstrates that in principle such a platform is possible. However, within the current state-of-the-art of repository mining, we found no publicly available approach that is similar to CODEMINE in terms of its capabilities. Therefore, we considered a CODEMINE for researchers who work on publicly available data, as our goal and started to built a platform similar to it 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: 1) is such a platform able to perform different analytic tasks?; 2) can such a platform help address problems with the external validity of studies?; and 3) which lessons additional to the ones provided by CODEMINE are important for researchers who try to develop such a platform? To this aim, we conducted an experience study to collect data for the evaluation of SmartSHARK. In addition to the three main questions above, we considered the usability of such a platform, as it is a central factor for the acceptance by other researchers.

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

- The SmartSHARK platform that combines automated data collection from different sources with an Application Programming Interface (API) for software analytics based on publicly available tools.
- An evaluation of the lessons learned from CODEMINE from a research perspective.
- An analysis based on an experience report of how such a platform can be used to resolve problems with the external validity.

The remainder of this paper is structured as follows. In Section 2, we give an overview of the related work. Then, we describe the SmartSHARK platform in Section 3. Afterwards, we give an experience report regarding the usage of SmartSHARK in Section 4. In Section 5, we discuss how SmartSHARK can contribute to resolve problems regarding the external validity of the results and discuss lessons learned from our experience. Finally, we conclude our paper and give an outlook on future work in Section 6.

2. RELATED WORK

The research on software data collection and combination, as well as software analytics cover many different directions and aspects. Most related work only considers either the data collection (e.g., [12, 17, 35]), potentially including the provision of data sets or queryable databases (e.g., [51, 42, 43, 19, 44]) or the analytical aspect (e.g., defect prediction [15], effort prediction [55], developer social networks [54]), and very few studies deal with both aspects.

A proprietary and not publicly available approach for building a software analytics platform with integrated data collection is the Microsoft internal CODEMINE [18]. With our work, we tried to built 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 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 for building tools similar to CODEMINE. Our SmartSHARK platform uses the same general structure as CODEMINE and these lessons were vital for the development of SmartSHARK. Details on how the lessons learned influenced SmartSHARK are discussed in Section 5.2. 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 external validity of studies.

Dyer et al. [25, 26] 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 [6] 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, e.g., based on software metrics, social aspects, or similar, which are possible with SmartSHARK. Such data 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. Furthermore, Boa heavily uses MapReduce for its queries, whereas SmartSHARK uses Apache Spark [80, 79]. It is reported that *Hadoop* MapReduce is inefficient for interactive analytics [80] as they are intended to be performed with Boa. This is the main problem that should be resolved by Apache Spark. 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 external validity can be improved.

Gousios and Spinellis [41] developed the *Alitheia Core* platform to perform "large-scale software quality evaluation studies" [41]. The architecture of Alitheia Core is divided into three different layers: (1) result presentation, (2) system core, and (3) data mirroring, storage, and 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. 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 very 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 the data analytics through Apache Spark's Mllib [9] and GraphX [8] libraries. Moreover, same as Boa, the authors did not consider how the platform can improve the external validity of studies.

There are also commercial approaches that try to combine software data collection with software analytics. Bitergia [13] 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 [14]. 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. 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[1]. 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.

[1]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.

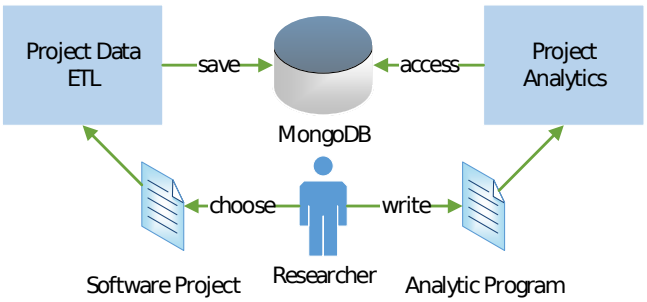

Figure 1: Design of SmartSHARK

3.1 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 information is supplied, the ETL process for this project can be started and is performed automatically.

Currently, SmartSHARK supports GIT as VCS. The programming language is required by InFamix [52] for the calculation of software metrics. During the ETL process, the progress can be observed in the web front-end to give the user a feedback about it. Furthermore, each project has its own configuration files, which can optionally be edited via the web front-end to customize the ETL process.

3.1.1 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 hard to adapt to different circumstances, because of the tight coupling between the extraction process and the application. Makedonski et al. [63] proposed a model-based software mining infrastructure, called DECENT, to circumvent these problems. The framework relies on "homogeneous high-level domain-specific models of facts extracted from raw assets" [63]. These different domain-specific models are then combined and transformed into models, that are related to a certain assessment task. The models make use of the Eclipse Modeling Framework (EMF). DECENT uses different tools, which are widely used in research, to extract facts from raw assets in the extraction step. These tools include CVSAnaly [65] for extracting information out of source code repository logs (e.g., used in [32, 48]), InFamix [52] for calculating software quality metrics (e.g., used in [4]), and DuDe [24, 76] for duplication detection.

Extracted facts are transformed into different (high-level) facts models and afterwards combined and linked together into a single DECENT model. This DECENT model is further transformed into an intermediate EMF model (called DECENTMongo), which represents the structure of the MongoDB used by SmartSHARK. From this model, the collected data is finally loaded into the MongoDB for later usage within analytic programs.

3.1.2 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.

Furthermore, 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 labels. The software metrics include static source code metrics, change metrics, and social metrics (e.g. developer cooperation factors). A list of the metrics can be found in the SmartSHARK documentation online [72]. Furthermore, 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 [34] of the commit. Especially these diffs can be important for the development of new analytic approaches, as shown by Walden et al. [75].

Additionally, we determine bug labels (artifact is bug-prone at this change or not) and a confidence indicator label for each artifact at each change. The confidence label indicates, if the resulting bug label can be trusted. These bug and confidence labels are generated by DECENT via origin analysis [62].

3.1.3 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. [49]. 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. [20].

3.2 Software Analytics

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 backend. 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 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.

3.3 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 [77]. 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 has an own 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.

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.

3.4 Cloud Deployment

Currently, we are running SmartSHARK in a small private cloud that we operate locally for research purposes. The infrastructure of our SmartSHARK deployment is depicted in Figure 2. Currently, we have a web-server that also serves as database server and data collection server and an Apache Hadoop cluster to execute analytic jobs. The platform can be automatically deployed via our deployment scripts using the DevOps technologies Vagrant [46] and Ansible [5]. Details regarding the used software versions and more information regarding the deployment are given on the homepage of SmartSHARK [28].

4. 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

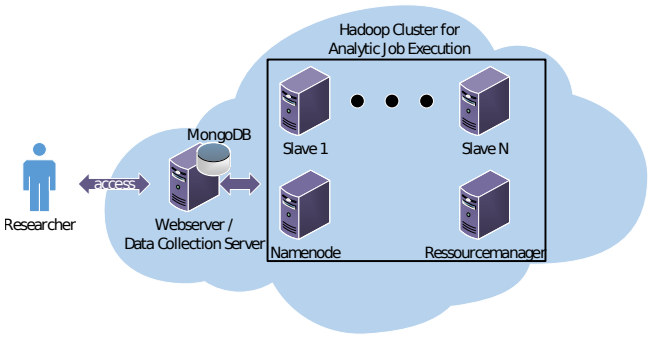

Figure 2: Infrastructure of SmartSHARK.

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. The last part of our experience report is related to an often neglected, but important attribute of such platforms: the usability.

4.1 Data Collection

To evaluate the data collection, we randomly selected 23 projects from GitHub [37]. 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 a project was that they were programmed in Java, C, or C++. Table 1 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.
- InFamix did not work for all revisions of the Mahout [11] project. We were not able to track down the problem, because while InFamix is available for free, it 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.

4.2 Software Analytics

The second part of our experience report is 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.

4.2.1 Visualization of Evolution Trends

We created five 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.

File-level Bug Overview Visualization. This visualization depicts the number of changed files over the project lifespan together with the number of defects.

Mailing List Activity - Number of sent Messages. For projects that have a mailing list, 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.

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.

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 4.2.2). We show which changes were predicted as defective by our created model in comparison to which changes 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.

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. One example for the mailing list activity visualization, which shows the number of sent messages in the project log4j, is depicted in Figure 3. Other visualizations, like the defect prediction results visualization, are accessible via the SmartSHARK website [28].

4.2.2 Defect Prediction

The second analytic example implemented on SmartSHARK is a defect prediction model. We selected a change-based defect prediction model based on a recent publication by Tan et al. [71]. The approach by Tan et al. suggests 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.

The data required for the defect prediction was readily available in the MongoDB back-end of SmartSHARK. Using the functionality provided by Spark, creating the data splits as required for the approach by Tan et al. [71] as well as the training of the defect prediction model was straightforward. After fetching the data from the MongoDB, a Map job was used to prepare the data. Then, the defect prediction model was trained using the machine learning package *spark.ml* of Apache Spark [9]. We evaluated our classifier

Project	Lang.	#Commits	Size	#Files	#ML Messages	Type
guice [39]	Java	1442	89 MB	713	n/a	Programming
openage [69]	C++	1761	8 MB	560	n/a	Game
Minesweeper [60]	Java	65	7 MB	207	n/a	Game
HackerNews [59]	Java	12	1 MB	78	n/a	News
SMSSync [74]	Java	1395	40 MB	557	n/a	Messaging
cursynth [73]	C++	219	3 MB	185	n/a	Audio
passivedns [33]	C	220	1 MB	50	n/a	Network
oryx [16]	Java	372	48 MB	537	n/a	Machine Learning
ohmu [40]	C++	226	3 MB	185	n/a	Programming
libxcam [1]	C++	250	3 MB	242	n/a	Camera
libyami [2]	C	487	4 MB	248	108	Media
wds [3]	C++	238	3 MB	193	27	Media
oclint [67]	C++	733	3 MB	349	n/a	Programming
xgboost [23]	C++	1847	8 MB	373	n/a	Mathematics
elasticsearch-hadoop [27]	Java	1243	9 MB	576	n/a	Search Engine
cxxnet [21]	C++	852	4 MB	173	n/a	Machine Learning
mxnet [22]	C++	236	1 MB	124	n/a	Machine Learning
osquery [30]	C++	2208	9 MB	555	n/a	Instrumentation
swift [31]	Java	496	8 MB	427	n/a	Programming
fatal [29]	C++	401	3 MB	141	n/a	Programming
k3b [56]	C++	5508	47 MB	1069	587	Media
ksudoku [58]	C++	668	7 MB	233	12544[2]	Game
log4j [10]	Java	3266	18 MB	643	54546	Programming

Table 1: Information about the examined projects.

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: 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 mode 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.

4.2.3 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 analytic job takes a list of projects as input. To create this 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.

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.

While not part of the sample analytic, SmartSHARK would also allow to give insights on the expected number of source code revisions and, e.g., the relationship to the number of unique developers. This way, SmartSHARK would allow to gain further insights and expand upon this very simple approach to estimate the effort for a project.

4.3 Usability

Usability is defined through its major aspects: a given task must be executed with effectiveness, efficiency and satisfaction [53]. To assess these attributes we conducted different experiments with SmartSHARK. In this Section, the usability of the four most important tasks is evaluated.

4.3.1 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 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.

[2]There is no ksudoku specific list. Instead we collected the whole kde-games-devel mailing list [57] data.

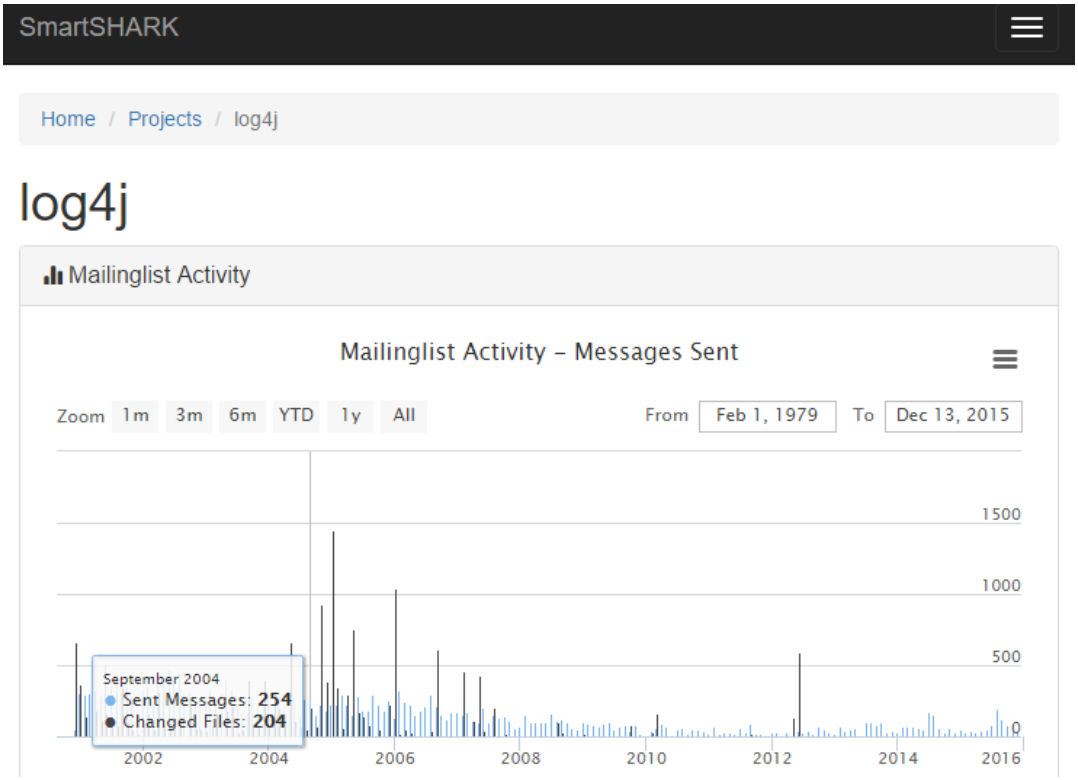

Figure 3: Screenshot of the mailing list activity of log4j mined by SmartSHARK.

4.3.2 *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 [7]. 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.

4.3.3 *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.

For all options the user needs to get active, i.e., access the MongoDB, downloading the results from the file system, or gather the log file to see the print output. We have shown one way to overcome this problem by directly show the defect prediction results on the SmartSHARK website.

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 retrieving 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 [78], which are plug-ins that implement the logic to gather and display the data. Furthermore, these plug-ins can easily be shared.

4.3.4 *Addition of New Data Sources*

We developed SmartSHARK in two increments: the first increment contained only the ETL of VCS data. We then started with the collection of data for all projects. Then, we extended SmartSHARK in the second increment with the ETL of mailing list data. For all projects that contain a mailing list, the new data could be added without problem.

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.

5. DISCUSSION

Within this section, we discuss how SmartSHARK can be used to address problems with external validity as well as lessons learned.

5.1 Addressing External Threats to Validity with SmartSHARK

The motivation of our work on SmartSHARK are the five major problems regarding the external validity of results as we discussed in the introduction. 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, since the body of data can be automatically extended with new projects. 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 platform must be able to collect the required data for the desired analytics. But our experience shows, that the ETL process is extensible and, moreover, the results of Spark jobs can be stored in the MongoDB to further enrich the data, if required.

(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 [78]. 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 mining, the addition of new data to SmartSHARK is not very time consuming for researchers, even if the mining 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 diverse tooling. We have experienced, as described in Section 4.3.4, 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.

5.2 Lessons from CODEMINE

While not many details about Microsoft's internal CODEMINE [18] 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. The target of SmartSHARK are not product teams, but rather research communities. We believe that ideally only a single instance of SmartSHARK for each community or even multiple communities is better suited. Nevertheless, we have taken this lesson into account by designing SmartSHARK as an independent platform, which can be set up easily via DevOps tools to support separate instances for research communities or teams.

Have uniform interfaces for data analysis. 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" [18]. 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. We found this lesson to be very important for SmartSHARK, because in research the kinds of analytics can be almost anything. Therefore, we designed the database to be as flexible as possible (NoSQL key/value storage) and allow with Apache Spark all kinds of analytic programs.

Allow dynamic discovery of data platform's capabilities by application. 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.

Support policies for security, privacy, and audit. 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. 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. Following this lesson, we designed SmartSHARK as a cloud platform to be easily scalable in terms of computational and storage resources.

Know the data platform might not fulfill all data needs. 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.

5.3 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.

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 [65]. However, the tool once stopped for an extended period of time, which is problematic for a fully automated mining 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, except exchanging InFamix with another tool.

Document, document, document! Another problem that we encountered is the insufficient documentation of tools, libraries and API descriptions of tools published by researchers, together with often sparse source code documentation. This made their reuse to built a larger platform extremely difficult. Hence, quality of documentation should also be a factor for research prototypes to facilitate their re-use.

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 3.1.1). 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 is a solution to this problem, which was presented by Scheidgen et al. [68]. This approach showed, that it is possible to transparently fragment the different models and store these fragments, e.g., in a MongoDB.

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.

Heavy resource demand on the infrastructure. Even taking the above lessons into account, we experienced that our setup with four Virtual Machines (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.

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.

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.

5.4 Threads to Validity

There are several threads to validity regarding our work. First, the usability evaluation was done only by authors of this paper and, therefore, may be biased. Hence, a usability evaluation needs to be done with more and unbiased people (e.g., students, other researchers) to get additional insights regarding the usability of SmartSHARK. Furthermore, we used SmartSHARK to collect data from 23 projects. We described the problems, that we encountered during the data collection process and we need to evaluate it for more projects. Furthermore, the scalability of the platform (in both, data collection and analysis) could not be tested, as we do not have the required infrastructure for such tests at the moment.

6. CONCLUSION

Within this paper, we discussed how a platform that combines automated data collection with a flexible analytic frontend can be used to address problems regarding the external validity of studies. To this aim, we created the platform SmartSHARK, inspired by CODEMINE [18]. 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) outlined how SmartSHARK help improve the external validity of studies; and 3) 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 the future, we plan to build upon our results and extend SmartSHARK. SmartSHARK was developed as combination of a feasibility study regarding whether such a platform is possible, as well as for the overall analysis whether such a platform makes sense for researchers. Furthermore, we wanted to gain insights into which design decision are important. Using our knowledge, we plan to extend SmartSHARK to a fully fledged platform to be used by other re-

searchers. This includes, but is not limited to the following aspects:

- migration of SmartSHARK to a larger cloud environment to be able to massively collect project data, expand the database, and analyze the scalability of the platform;
- a plug-in system for the definition of visualizations;
- inclusion of new data sources, especially Issue Tracking Systems (ITSs);
- enhanced APIs for job submission and debugging;
- enhanced APIs built on top of Apache Spark to further support the definition and evaluation of analytic problems, e.g., the generation of developer social networks or defect prediction models;
- usage of the platform by additional user groups, e.g., students as part of lectures or other researchers to get feedback about the usability and extend the overall functionality of the platform.

Acknowledgements

The authors would like to thank Fabian Glaser, Michael Göttsche, and Gunnar Krull for their support for regarding the cloud technologies and deployment.

7. REFERENCES

[1] 01org. Libxcam GitHub. https://github.com/01org/libxcam. [accessed 22-January-2015].

[2] 01org. Libyami GitHub. https://github.com/01org/libyami. [accessed 22-January-2015].

[3] 01org. Wds GitHub. https://github.com/01org/wds. [accessed 22-January-2015].

[4] C. V. Alexandru and H. C. Gall. Rapid Multi-Purpose, Multi-Commit Code Analysis. In *Proceedings of the IEEE/ACM 37th International Conference on Software Engineering (ICSE)*, pages 635–638. IEEE/ACM, 2015.

[5] Ansible Inc. Ansible Documentation. http://www.ansible.com/. [accessed 22-January-2015].

[6] Apache Software Foundation. Apache Hadoop. https://hadoop.apache.org/. [accessed 22-January-2015].

[7] Apache Software Foundation. Apache Hadoop Wiki. https://wiki.apache.org/hadoop/HowToDebugMapReducePrograms. [accessed 22-January-2015].

[8] Apache Software Foundation. Apache Spark GraphX. http://spark.apache.org/graphx/. [accessed 01-March-2016].

[9] Apache Software Foundation. Apache Spark MLLib. http://spark.apache.org/docs/latest/mllib-guide.html. [accessed 22-January-2015].

[10] Apache Software Foundation. Log4j GitHub. https://github.com/apache/log4j. [accessed 22-January-2015].

[11] Apache Software Foundation. Mahout GitHub. https://github.com/apache/mahout. [accessed 22-January-2015].

[12] J. Bevan, E. J. Whitehead Jr, S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 177–186. ACM, 2005.

[13] Bitergia. Bitergia. http://bitergia.com/. [accessed 22-January-2015].

[14] Black Duck Software, Inc. Open HUB. https://www.openhub.net/. [accessed 22-January-2015].

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

[16] Cloudera. Oryx GitHub. https://github.com/cloudera/oryx. [accessed 22-January-2015].

[17] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Transactions on Software Engineering*, 31(6):446–465, 2005.

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

[19] D. Di Ruscio, D. S. Kolovos, I. Korkontzelos, N. Matragkas, and J. Vinju. Ossmeter: A software measurement platform for automatically analysing open source software projects. In *ESEC/FSE 2015 Tool Demonstrations Track*, 2015.

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

[21] Distributed Machine Learning Common. Cxxnet GitHub. https://github.com/dmlc/cxxnet. [accessed 22-January-2015].

[22] Distributed Machine Learning Common. Mxnet GitHub. https://github.com/dmlc/mxnet. [accessed 22-January-2015].

[23] Distributed Machine Learning Common. Xgboost GitHub. https://github.com/dmlc/xgboost. [accessed 22-January-2015].

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

[25] R. Dyer, H. A. Nguyen, H. Rajan, and T. Nguyen. Boa: Ultra-Large-Scale Software Repository and Source Code Mining. *ACM Transactions on Software Engineering and Methodology*, forthcoming, 2015.

[26] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. 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)*, 2013.

[27] Elasticsearch BV. Elasticsearch-hadoop GitHub. https://github.com/elastic/elasticsearch-hadoop. [accessed 22-January-2015].

[28] Fabian Trautsch. SmartSHARK Homepage. http://smartshark.informatik.uni-goettingen.de. [accessed 22-January-2015].

[29] Facebook Inc. Fatal GitHub. https://github.com/facebook/fatal. [accessed 22-January-2015].

[30] Facebook Inc. Osquery GitHub. https://github.com/facebook/osquery. [accessed 22-January-2015].

[31] Facebook Inc. Swift GitHub. https://github.com/facebook/swift. [accessed 22-January-2015].

[32] J. Fernandez-Ramil, D. Izquierdo-Cortazar, and T. Mens. What does it take to develop a million lines of open source code? In *Open Source Ecosystems: Diverse Communities Interacting*, pages 170–184. Springer, 2009.

[33] E. Fjellskål. Passivedbs GitHub. https://github.com/gamelinux/passivedns. [accessed 22-January-2015].

[34] Free Software Foundation. GNU Diffutils. http://www.gnu.org/software/diffutils/. [accessed 22-January-2015].

[35] D. M. German. Mining CVS repositories, the softChange experience. *Evolution*, 245(5,402):92–688, 2004.

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

[37] I. GitHub. GitHub. https://github.com/.

[38] M. Godfrey and Q. Tu. Tracking structural evolution using origin analysis. In *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE)*, 2002.

[39] Google. Guice GitHub. https://github.com/google/guice. [accessed 22-January-2015].

[40] Google. Ohmu GitHub. https://github.com/google/ohmu. [accessed 22-January-2015].

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

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

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

[44] I. Grigorik. GitHub Archive. https://www.githubarchive.org/. [accessed 22-January-2015].

[45] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, Nov 2012.

[46] HashiCorp. Vagrant. https://www.vagrantup.com/. [accessed 22-January-2015].

[47] G. Hecht, B. Omar, R. Rouvoy, N. Moha, and L. Duchien. Tracking the software quality of android applications along their evolution. In *Proceedings of the IEEE/ACM 30th International Conference on Automated Software Engineering (ASE)*, page 12. IEEE, 2015.

[48] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Forecasting the number of changes in Eclipse using time series analysis. In *Proceedings of the 4th IEEE Working Conference on Mining Software Repositories (MSR)*, 2007.

[49] I. Herraiz, G. Robles, J. J. Amor, T. Romera, and J. M. González Barahona. The processes of joining in global distributed software projects. In *Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner*, pages 27–33. ACM, 2006.

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

[51] J. Howison, M. S. Conklin, and K. Crowston. Ossmole: A collaborative repository for floss research data and analyses. In *Proceedings of the 1st International Conference on Open Source Software*, 2005.

[52] Intooitus. InFamix. https://www.intooitus.com/company/news/introducing-infamix-free-ccjava-parser-moose. [accessed 22-January-2015].

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

[54] A. Jermakovics, A. Sillitti, and G. Succi. 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)*, CHASE '11, pages 24–31, New York, NY, USA, 2011. ACM.

[55] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, Jan 2007.

[56] KDE. K3b GitHub. https://github.com/KDE/k3b. [accessed 22-January-2015].

[57] KDE. KDE games developer mailing list. https://mail.kde.org/pipermail/kde-games-devel. [accessed 22-January-2015].

[58] KDE. Ksudoku GitHub. https://github.com/KDE/ksudoku. [accessed 22-January-2015].

[59] E. Lawlor. HackerNews GitHub. https://github.com/lawloretienne/HackerNews. [accessed 22-January-2015].

[60] E. Lawlor. Minesweeper GitHub. https://github.com/lawloretienne/Minesweeper. [accessed 22-January-2015].

[61] Machine Learning Group at the University of Waikato.

WEKA. http://www.cs.waikato.ac.nz/ml/weka/. [accessed 22-January-2015].

[62] P. Makedonski and J. Grabowski. 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.

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

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

[65] Metrics Grimoire. CVSAnaly GitHub. http://github.com/MetricsGrimoire/CVSAnalY. [accessed 22-January-2015].

[66] R Foundation. R Project. https://www.r-project.org/. [accessed 22-January-2015].

[67] R. Saito. Oclint GitHub. https://github.com/oclint/oclint. [accessed 22-January-2015].

[68] M. Scheidgen, A. Zubow, J. Fischer, and T. H. Kolbe. *Automated and transparent model fragmentation for persisting large models*. Springer, 2012.

[69] SFTtech. OpenAge GitHub. https://github.com/SFTtech/openage. [accessed 22-January-2015].

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

[71] M. Tan, L. Tan, S. Dara, and C. Mayeux. Online Defect Prediction for Imbalanced Data. In *Proceedings of the IEEE/ACM 37th International Conference on Software Engineering (ICSE)*, 2015.

[72] F. Trautsch. SmartSHARK MongoDB Design. http://smartshark.informatik.uni-goettingen.de/index.php?r=site%2Fmongodesign. [accessed 22-January-2015].

[73] M. Tytel. Cursynth GitHub. https://github.com/mtytel/cursynth. [accessed 22-January-2015].

[74] Ushahidi. SMSSync GitHub. https://github.com/ushahidi/SMSSync. [accessed 22-January-2015].

[75] J. Walden, J. Stuckman, and R. Scandariato. Predicting vulnerable components: Software metrics vs text mining. In *Proceedings of the IEEE 25th International Symposium on Software Reliability Engineering (ISSRE)*, pages 23–33. IEEE, 2014.

[76] R. Wettel. DuDe. http://www.inf.usi.ch/phd/wettel/dude.html. [accessed 22-January-2015].

[77] Yii Software LLC. Yii Framework. http://www.yiiframework.com/. [accessed 22-January-2015].

[78] Yii Software LLC. Yii Framework - Widgets. http://www.yiiframework.com/doc-2.0/guide-structure-widgets.html. [accessed 22-January-2015].

[79] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. 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)*, 2012.

[80] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, 2010.