Risk Based Testing of Open Source Software (OSS)

Inbal Yahav*, Ron S. Kenett† and Xiaoying Bai‡

*Department of Information Systems
School of Business Administration, Bar Ilan University
Ramat Gan, Israel
Email: iyahav@biu.ac.il

†KPA Group, Israel, the University of Torino, Italy, and the NYU-Poly, NY, USA
Email: ron@kpa-group.com

‡Department of Computer Science and Technology
Tsinghua University
Beijing, China
Email: baixy@tsinghua.edu.cn

Abstract—Open Source Software (OSS) has become a strategic asset for a number of reasons, such as its short time-to-market software service and product delivery, reduced development and maintenance costs, introduction of innovative features and its customization capabilities. By 2016 an estimated 95% of all commercial software packages will include OSS components. This pervasive adoption is not without risks for an industry that has experienced significant failures in product quality, timelines and delivery costs. Exhaustive testing of any software system and, specifically, of open source software components is usually not feasible due to limitations in time and resources. In risk-based testing approach test cases are selected and scheduled based on software risk analysis. This research introduces the strategy of risk-based adaptive testing of OSS by combining information on the OSS community ecosystem with risk-driven test selection and scheduling strategy. A key feature of the proposed approach is the monitoring and analysis of OSS community dynamics, including chats and email communications, blogs, repositories of bugs and fixes, and more. The community and its dynamics are then monitored to detect anomaly communication between the community members. Our approach is demonstrated in the XWiki OSS, a Java-based environment that allows for the storing of structured data and the execution of server side scripts within the wiki interface. We illustrate our concepts, methods and approach behind risk based testing.

Keywords—Open Source Software, Social Networks, Testing, Risk, XWiki.

I. INTRODUCTION

The increasing adoption of open source software (OSS) components in software systems introduces new quality risks. OSS components are usually developed and maintained by open communities. The fluctuation of community members and structures can result in instability of the software quality. For example, voluntary developers may join and quit the community flexibly. There is no guarantee of consistent quality between different versions, code branches, and development groups. Hence, an investigation is necessary to analyze whether (and how) the social aspects of open communities, such as the dynamics in communications and content exchange, can shed light on and the quality of the OSS components. The in-depths analysis can drive a systematic quality insurance process and to facilitate decision making of OSS adaptation.

Software testing provides an objective, independent view of software quality and performance that allows the adopters to assess and understand the risks of software implementation. A software feature is risky if it has a high probability to fail or its failure will result in serious consequence. The Agile approach promotes test-drive development in which test cases are developed before coding to serve as the validation criteria of program. It enforces an iterative process with parallel development of production code and test scripts, small and continuous improvements and deliveries. The Agile testing style is thus well-suited to Internet-based software development like OSS. In addition to volunteer developers, OSS communities can also have self-organized testers. Different from product-oriented in-house organized test teams, for testers working in the open community, resources can be efficiently distributed and dynamically assigned to different projects, which gives them the ability to test faster, better and at lower costs, while knowledge and is accumulated across different projects and shared among the community members.

To address the new requirements and problems in OSS testing, the paper promotes a risk-based approach for test resource allocation. As exhaustive testing is infeasible in most cases due to time and resource limitations, selective testing techniques are usually needed to allocate test re-
sources to the most critical components and features. Risk-based testing is a technique to schedule tests according to the risk measurements on the object under tests [1], [2], [3]. In this research, we monitor the community dynamics to detect early signs of risk of an OSS component. These risk alert are then used to guide the components testing activities.

Figure 1 shows the overall approach in this paper. A monitor is instrumented to analyze community dynamics continuously, including communications like email and blogs, and repositories of bugs and fixes. It detects anomalies in the monitored behavior, such as increase in communication levels, unexpected content exchanges, etc. Risks are usually defined by two factors: the probability of failure, and the consequence of failure (see [4]). In this research, these factors are evaluated based on detected anomalies. Test cases are selected from pre-defined repositories and priorities based on the risk evaluation of the target OSS components. Risky components are allocated more test cases and ranked first for execution. The paper reports preliminary attempts in this research direction. Our approach is still experimental, but we thoroughly test the link between community changes and software failures with innovative data-mining algorithms. We illustrate the approach with the XWiki OSS projects. We focus here on (1) method for mining OSS communities (theoretical approach), and (2) the link between community changes in the community and reported bugs (case study).

A. Theoretical Approach: Mining OSS Community

In this work, risk, and risk levels are extracted from communication between developers and users in the OSS community. Communication channels usually include mail, chats, blogs, and/or repositories of bugs and fixes. The main challenge in such communities is that only a small portion of the developers have defined roles (such as contributor, committer, maintainer, etc), yet the majority of the developers and contributors choose their own role, defined by the code that they contribute to the OSS. It is therefore unclear, without a sophisticated analysis, which member is in charge of what component.

To understand the role of each member, we first contract a social network of communication. Nodes in the network can have one of three types:

1) **Team member**: information on team members, such as user maintainer, is usually publicly available in OSS communities.
2) **Developer**: contributor and committer.
3) **User**: user of an OSS component, or more.

Social ties in the network are defined as communication exchange between the members. It is important to note that the network has a temporal dimension. That is, the network changes over time, or, the dynamics in the community develops.

Next, we group members in the community based on (1) member type, and (2) communication. The idea here is to cluster members with similar roles into clusters (or sub-communities). Since clustering is done based on communications, each of these clusters is likely to handle/use the same set of components. Clustering is based on a mixed knowledge-based (about members’ roles) and data-driven approach, using standard community detection algorithms (for e.g., [5]). This will later serve as a ground for our anomaly detection.

Once the network is constructed and clustered, it is monitored for anomalies in its dynamics. Monitoring is done in two levels:

1) The network topology, including traffic and traffic levels within and across clusters of members (who talks with who, and how often), and
2) The content exchanged in the different communication channels (what they are talking about).

Examples for anomalies are two clusters that commonly do not communicate suddenly chat frequently, or a clusters of members that normally discuss issues on component $X$, raise problems with component $Y$.

Once an anomaly is signaled, we analyze the anomaly source (e.g., component) and severity, to define test cases accordingly. Since signaling is a new approach that requires an intensive learning process, experts opinion is also proposed in initial stages for verification and modification of the test selection method.

In the next section we analyse the XWiki OSS community. We discuss the practical challenges in processing the data: how to extract the social network of communication and how to determine the role of members in the community. We then discuss the link between the communication exchanges and bug reported. That serves as a proof of concept to our approach: if we can predict bugs in a specific component before the actual report, we can use this information to automatically run test cases related to this component, and fix the bug before it affects the users.

II. The XWiki Case Study

XWiki (http://www.xwiki.com) is a Java-based environment that allows for the storing of structured data and the execution of server side script within the wiki interface.
XWiki was originally written by Ludovic Dubost in 2003. In 2006, the Apache Maven developer, Vincent Massol became the lead developer of the OOS XWiki. In this work, we use XWiki data to demonstrate our testing approach. The data consists of: mailing lists archives of users and developers, IRC chat archives, commits via git, code review comments, and information about bugs and releases.

To illustrate our method we focus on users and developers email communications to describe the community (the input to our approach), and on bugs and fixed repository as labeling to the required tests (the output of our approach). Emails are available since 2005 to 2013, a total of nearly 60,000 messages (including original messages and replies). We generate the social network of communication, generate and analyze clusters of members, and monitor their within- and across- community communications.

A. Data Preprocessing: Network Construction and Clustering

Email data, by nature, is very noisy. Users can use different accounts to post and reply messages. For example Ludovic Dubost can use two distinct email addresses in different email channels: ‘LudovicDubost@xwiki.com’ and ‘LudovicD@xwiki.com’. Some users use over 5 different email addresses. Content exchanged is also a free text, and as such contains typos, synonyms, plurals, and more. A massive data pre-processing step is required in order to generate an informative social network.

Preprocessing the data for the purpose of constructing the social network described before is composed of two parts. The first part is defining the members of the community (removing duplicate emails, and associating each communication with the correct node), and their role (team member, developer, or user). The second part is clustering the users based on the content of their communication (what they are talking about).

In this case study, we use a name-scheme matching approach to detect multiple user’s accounts. In specific, we use full and partial texts comparisons to find plausible users matches and their probability. We then score our results with manual intervention. For example, the pair ‘LudovicDubost@xwiki.com’ and ‘LudovicD@xwiki.com’ receives a score of 8, based on the mutual string ‘LudovicD’, denoting a high probability of name matching.

Content analysis is based on keywords extraction from XWiki’s list of projects (https://github.com/xwiki) and contributed projects (https://github.com/xwiki-contrib). Mentions of keywords in an email communication (title only) are counted. Emails are then classified to zero or more projects, based on these keywords. For example, the text ‘LDAP authentication not working’ is classified as related to the project ‘LDAP’, which is part of ‘platform’ (https://github.com/xwiki/xwiki-platform/tree/master/xwiki-platform-core/xwiki-platform-ldap).

Three types of clusters makes up the social network: team members, developers (contributors) and and users.

For team members, we use the official roles, available on the XWiki website (www.xwiki.com/lang/en/Company/Team), to cluster the users. For example, all ‘Sales & Client Project Managers’ members are clustered into one group.

For simplicity (this is a preliminary analysis), all users are clustered together into a single cluster.

Developers are clustered using Clauset-Newman-Moore [5] community detection algorithm. The algorithm clusters group of developers based on the existence/ absence of communication between them, and its frequency. The communities that were detected by the algorithm are given in Figure 2. The figure depicts the directed graph of developers communication, divided into 6 communities (G1 through G6). The size of the nodes in the figure correspond to the importance of each node, or member, measures by its degree centrality (i.e., the total number of other members that communicates with that member). Arcs in the figure are email exchanged via the developers’ email channel (dark arcs), of the users’ email channel (light arcs). It is clear that there is normal communication both within and across clusters. This communication constitute the baseline communication, as opposed to anomalous communication.

From the 6 communities detected, the first community (G1) has no communication within the community, or with other developer communities. G1, therefore, gathers small developers that communicate with either users or members of the OSS team. The other 5 communities (G2 through G6) communicate both within the community and across the developers communities. All of the communities (apart from G1) are centered around few main developers, marked as big gray circles in the figure.

To understand the role of each community, we count project-related keywords that are raised in the communities’
emails, as explained earlier. A detailed level focus is exemplified for community G4 in Table I, where top mentions (on normal base) of projects, and terms related to these projects are counted.

Figure 3 illustrates the entire process of data preprocessing and community detection in the XWiki data. Once the network in generated, we monitor the communities to detect anomalous communication that requires enhanced testing attention. In order to determine where and how to boost testing effort, the information derived from the community is fed to a test case data bank to identify and prioritize specific tests. This ensures that testing efforts are focused on error-prone areas of the software application, where the OSS is being incorporated (see Figure 4).

For example, a monitoring method used to flag anomalies is the number of total communications over time that were initiated by a community (G4 in our example), and sent as replies to other communities (see Figure 5). A communication spike is noticed on March-April 2008, around the release of XE (enterprise) 1.3.1. Looking at projects mentions in the email communication (Figure 6), we can see that ‘Enterprise’ is indeed mentioned more than expected. Though this increase is not statistically significant (the statistical test used is the M-Test [6]), the content analysis exhibits an area of increased activity that deserved attention. An area which did show a significant increase is the Dev-Tools (from 3.4% to 7.5% of project mentions), thereby enhancing the priority of Dev-Tools test-case related utilization.

To generalize the conceptual framework of monitoring the community with the objective to detecting anomalies, we next propose a formal quantitative analysis. Our analysis uses the communication as signal (predictor) for test case selection, and the actual reported bugs as a proxy for required tests. We will show that if one uses our approach...
s/he can predict which projects are of high risk, and consecutively, which test case should be selected at any given time (assuming a pre-defined pool of software tests).

B. Bugs and Social Networks: Formal Analysis

We study the relationship between the social network of OSS community and reported bug. At first, we ignore temporality and examine a snapshot of the entire data at aggregation.

Deploying common social network analysis measurements, we examine the link between the centrality of members in the OSS community and their usefulness in predicting bugs. There are different measurements for centrality in social networks. In this analysis, we use the *Betweenness* measure, which examine the importance of members in transferring information in the network. Members with high Betweenness are crucial for information to diffuse in the network quickly. In other words, removing them from the network, will slow down the rate of which information is transferred to other members.

Linking central members to bugs, sheds light on which type of members (in terms of centrality) should be monitored more closely, perhaps qualitatively (using experts). Monitoring the communication of this members only, will enhance the prediction of bugs (of the efficiency of tests selection), with minimal effort.

Figure 7 depicts the relationship between reported bugs per project and the centrality of the people who discuss them. Each ‘dot’ in the picture is a specific project. The *x* axis is the mean centrality of members that talk about the project. The *y* axis is the (log) number of reported bug related to this project. The figure shows that the more central are the people that talk of a project, the larger is the number of bugs reported in the project. There are two plausible explanations to this relationship: (1) risky projects attract central developers, or (2) experts users and developers find more bugs (in other words, if a project is used by central, frequent users, it is more likely that its bugs will be detected). In the first explanation, bugs drive communication. In the second explanation, use drives communication that drives bug detection. A combination of the two is also likely.

Next, we add the time dimension to our analysis. For that, we develop a prediction model that predicts whether a bug will be reported on a given project, in a certain day. This information can be used as the basis of efficient test selection. We model a bug occurrence indicator at time $t$ with a logistic regression of the email communications in different communities and the time of the last reported bug.

To avoid model over-fitting, we consider projects with over 20 reported bugs, and mentioned over 20 times in email communications. The fitted model is given in Table II. From the table one can see that the significant coefficients correspond to Project XCONTRIB, Community Research, Community Sales and Client Projects. Several communities communications, such as that of G2, G3, Research group and others, appear informative in predicting the reporting of bugs. That is, if anomalous communication is detected, it is likely that a reported bug will follow, and a test was required.
The regression analysis mathematically illustrates our theoretical concept: monitoring the community is useful in the selection prioritization of tests in OSS components. In practice, the communication between the members of the community and its users, its structure and dynamics can be used as a signal for the OSS components quality.

### III. SUMMARY AND CONCLUSIONS

Classical testing is driven by requirements documents representing the goals of the software development effort. In incorporating OSS within an in-house application, the approach needs to be revisited. OSS offers opportunities for cost-reduction and time-to-market efficiencies, within a new paradigm where the behaviour of the OSS community can affect the adopters’ organizations’ performance. The management of an OSS community requires active monitoring and tracking.

Social communication monitoring opens an opportunity for developing a sophisticated risk-base testing tool that is based on typical information channels, such as repositories of bugs and fixes, as well as less common channels (in the context of testing), such as emails, chats, and blogs. The later allows access to up-to-date, and potentially future contextual data that is annotated with both developers and adopter (also referred to as social sensors [7]), and meta-data on software units. For an analysis of risks using structured and unstructured data in the context of operational risks see [8].

The objective of this study is to evaluate the feasibility of driving test efforts of OSS components on the basis OSS community data. The goal of our case study is to predict occurrence of bugs using email communication traffic and social network dynamics with data from an OSS community. In this preliminary attempt we illustrated the efficiency of our approach for the XWiki data. We divided the developers to communities according to their official rules and communication and monitored their communication regarding different projects over time. We showed that their communication and communication anomaly is useful in detecting bugs in the main XWiki projects.

**ACKNOWLEDGMENT**

The work of RSK was partially funded by the RISCOSS project under the EC 7th Framework Programme FP7/2007-2013 under the agreement number 318249.

**REFERENCES**


