Embrace your issues: Comprising the software engineering landscape using bug reports

Markus Borg
Dept. of Computer Science
Lund University
Lund, Sweden
markus.borg@cs.lth.se

ABSTRACT
Software developers in large projects work in complex information landscapes, and staying on top of all relevant software artifacts is challenging. As software systems often evolve for years, a high number of issue reports is typically managed during the lifetime of a system. Efficient management of incoming issue requires successful navigation of the information landscape. In our work, we address two important work tasks involved in issue management: Issue Assignment (IA) and Change Impact Analysis (CIA). IA is the early task of allocating an issue report to a development team. CIA deals with identifying how source code changes affect the software system, a fundamental activity in safetycritical development. Our solution approach is to support navigation, both among development teams and software artifacts, based on information available in historical issue reports. We present how we apply techniques from machine learning and information retrieval to develop recommendation systems. Finally, we report intermediate results from two controlled experiments and an industrial case study.

Categories and Subject Descriptors
D.2.5 [Software Engineering]: Testing and Debugging—Debugging aids, tracing; D.2.9 [Software Engineering]: Management—Programming teams

Keywords
issue management, recommendation systems, machine learning, information retrieval

1. INTRODUCTION
The information landscape in large software engineering projects is complex and ever-changing. Development teams collaborate using information repositories such as requirements databases, test management systems, source code repositories, and general document management systems, in total storing thousands of artifacts [9, 25, 17]. Often the interoperability between the repositories is poor, i.e., the repositories turn into isolated information silos with little transparency. An intrinsic strength of software is that it is easy to modify, but source code changes impact other software artifacts as well. Thus, both source code and other stored information is continuously created and updated in large projects [14], and staying on top of the information landscape can constitute a significant challenge for both developers and managers [26].

Information overload, i.e., projects characterized by "a state where individuals do not have time or capacity to process all available information" [16], threatens productivity in large software engineering projects. Quick and concise access to information is crucial in knowledge intensive work such as software development and evolution. If the project environment does not provide sufficient support for navigation and retrieval, substantial effort is wasted on locating relevant information [23].

In some projects, issue repositories constitute examples of information silos that contain challenging amounts of software engineering data. The continuous inflow of issue reports, especially for public issue repositories [2], makes activities such as duplicate management, prioritization, and work allocation tedious and error-prone [12, 7, 22]. While some of the activities are supported in state-of-practice issue repositories (e.g., Bugzilla and HP Quality Center offer automated duplicate detection), most processing of incoming issue reports is still manual.

Previous research argues that the issue repository is a key collaborative hub in large software engineering projects. Cubranic et al. developed Hipikat, a recommendation system to help newcomers in open source communities navigate the existing information, and used the issue repository to build a "project memory" [15]. Anvik and Murphy presented automated decision support for several activities involved in issue triaging, all based on information stored in issue repositories [4].

We also view the issue repository as a key collaborative hub, but increase the granularity further by considering each individual issue report as an important juncture in the information landscape. We have previously shown that issue reports can connect software artifacts that are stored in separate databases [10], i.e., issue reports are a way to break information silos. In software engineering contexts where the change management process is rigid, every corrective change originating from fixing an issue report must be documented. As such, the trace links from issue reports to artifacts in various repositories turn into trails in the information land-
scape, created by past engineers as part of their everyday software development.

In line with previous work, we apply Machine Learning (ML) to detect patterns in the inflow of issue reports. As ML in general performs better the more data that are available [5], we aim to harness the daunting inflow of issue reports to assist navigation in the software engineering landscape. Early results indicate that our solutions improve as the number of issue reports grows, e.g., in issue assignment [21] and prioritization [24]. The rest of this paper presents how we support two work tasks in issue management: Issue Assignment (IA) and Change Impact Analysis (CIA).

2. PROBLEM AND SOLUTION APPROACH

Our work particularly addresses IA and CIA, as our industrial partners stressed the importance of these two activities.

2.1 Automated IA using Ensemble Learning

The initial decision in issue triaging is typically to decide which development team should investigate a certain issue report, an activity referred to as IA. Several studies report that manual bug assignment is labor-intensive and error-prone [6, 19, 8], resulting in “bug tossing” (i.e., reassigning issue reports to other developers) and delayed issue corrections.

We support IA using supervised ML. In line with previous work, we train a classifier on the textual content in issue repositories [3, 19, 1]. However, while previous work has applied individual classifiers, we combine several classifiers in an ensemble learner using Stacked Generalization (SG). SG is a versatile ensemble learner able to combine classifiers of different types, and it was for example heavily used in the winning contribution of the well-known Netflix Prize [27].

Previous research on IA has primarily targeted Open Source Software (OSS) projects. OSS development is indeed interesting to study, but it is often treated as the default software engineering context. In fact, it differs in several aspects such as development processes, team organization and developer incentives. As presented in Section 3, we instead evaluate our approach in proprietary issue management contexts.

2.2 A Recommendation System for CIA

When the issue triager has concluded that a corrective fix is required for an issue report, a CIA is required. Especially in safety-critical development contexts, CIA is a formal analysis that must be completed before any changes to the source code is allowed [18]. Several researchers have studied CIA on the source code level [13]. However, in many proprietary development contexts, it is critical to also determine impact previously reported. As ImpRec has several parameters that need to be configured, and as automated solutions in software engineering typically are highly dependent on the dataset [11], we aim to publish guidelines on how to sys-

--

Lucene.apache.org

3. EVALUATION METHOD

Our idea is to evaluate automated support for issue management in two proprietary organizations. The first company, Comp Telecom, develops telecom infrastructure and employs thousands of software engineers. Allocating resources is a major challenge, and bug tossing is an explicit problem. The second company, Comp Auto, develops safety-critical solutions for the process automation industry. The development process follows safety standards mandating rigorous process requirements on traceability. CIA is an important work task for the developers, but it requires much effort.

We evaluate automated IA in a controlled experiment using five proprietary datasets. Four datasets originate from different projects at Comp Telecom, and one dataset originates from Comp Auto. In total more than 50,000 issue reports are used in the experiment. We train classifiers on different subsets of historical issue reports, and test our approach on more recent issue reports.

Initial evaluations of ImpRec were also conducted in a controlled in-vitro setting. We extracted 27,000 issue reports from Comp Auto, constituting 12 years of development of an industrial automation system. We mined a knowledge base from the previous CIA reports, in total about 5,000 reports, and compared the ImpRec output to the “true” impact previously reported. As ImpRec has several parameters that need to be configured, and as automated solutions in software engineering typically are highly dependent on the dataset [11], we aim to publish guidelines on how to sys-
tematically tune tools such as ImpRec as an outcome of the in-vitro evaluation.

ImpRec is also evaluated in an in-vivo setting. We are currently conducting an industrial case study at Comp Auto in Sweden (Case Sweden). We have selected one development team that performs frequent CIAs as part of their work. This team consists of four members, and all of them have different development roles installed ImpRec on their local machines in March 2014. The team members are encouraged to use ImpRec when they perform CIAs, and to report their experiences from using the tool. As the developers use ImpRec, all user actions are collected. Before the tool was deployed, we conducted seven interviews to better understand the current CIA process. Moreover, in August 2014 we will deploy ImpRec in a team at the Comp Auto development site in Bangalore, India, to enable a replication of the case study in another part of the organization (Case India). While the two cases under study will be different, the involved developers work on the same automation system and use the same issue repository. Thus, the required reconfiguration of ImpRec will be minimal.

4. INITIAL RESULTS

Our controlled experiment of automated IA shows that combining classifiers under SG consistently outperforms individual classifiers with regard to prediction accuracy [20]. Furthermore, our results confirm that it is worthwhile to strive for a diverse set of individual classifiers in the ensemble, consistent with recommendations in the general ML research field. Our approach reaches prediction accuracies of 50% to 90% for the five different datasets, in line with the prediction accuracy of the current manual processes. The learning curves for all five datasets display similar behaviour: IA for all datasets benefits from more training data, but reaches different maximum prediction accuracy. However, as all learning curves start to flatten out at about the same size of the training set, we present an empirically based rule of thumb: when training a classifier for automated IA, aim for at least 2,000 issue reports in the training set.

We first evaluated ImpRec in a controlled setting, and explored how to configure the involved parameters. Our initial results suggest that using multiple starting points (>15) in the knowledge base, and considering impacted artifacts several trace links away (>5), increases the prediction accuracy. However, considering a high number of impact candidates requires a good ranking function. In our ranking function, the most important features are the network centrality, and the textual similarity between the newly submitted issue report and the historical issue reports used as starting points. Using feasible feature weighting, ImpRec predicts more than 30% of the previously reported impact among the first 5 recommendations, and more than 40% among the first 10 recommendations. Some recommendations beyond the first 10 tend to be correct, but we typically do not value candidates far down the list, as we suspect that developers are unlikely to browse too long lists of potentially impacted artifacts.

A preliminary analysis of the interviews conducted in Case Sweden indicates that developers see the CIA work task as a rather negative necessity, sometimes conducted just to comply with the safety process. The interviewees confirm that it is a manual time-consuming activity, and highlight two fundamental challenges: 1) recognizing the value of formal CIA, and 2) feeling confident in the answers provided in a CIA. Furthermore, one developer claims that the most difficult artifacts to assess for potential impact are the individual requirements rather than the source code. The interviews thus suggest that our work addresses important challenges. First, using ImpRec could motivate developers to conduct high-quality CIA, as the information will be reused to kickstart CIAs in the future [10]. Second, ImpRec can be used to verify a manual CIA as a way to boost confidence. Finally, ImpRec supports recommending potentially impacted requirements, as well as other non-code software artifacts.

Since ImpRec was deployed in Case Sweden, the collected log files reveal that the four developers have used ImpRec to search for impacted artifacts 23 times. The case study has so far been running for 11 weeks, i.e., 44 man-weeks, showing that the developers on average conducted about one IA every second week (assuming they followed our instructions). While this number is lower than what we hoped for, it is in line with the expectations expressed by the developers during the interviews. During this first part of the case study, ImpRec has recommended information considered valuable to the developers in 37.5% of the use cases. As the data collected from the tool are rich, containing both ranking output and time stamps, we plan to perform a deeper analysis when user data has been collected also from Case India.

5. CONCLUSION

Software developers in large projects must navigate complex information spaces as part of their everyday work. One activity that forces developers to locate relevant information is issue management, i.e., resolving incoming issue reports. While the volume of issue reports in a project can be daunting, we suggest harnessing the intrinsic navigational value of historical issue reports. IA is an early manual step in issue management. We train an ensemble learner on previous issues, and use the identified patterns to automate team assignment. Our results are in line with the current manual process, i.e., our automated solution selects the correct team for about 50-80% of the issue reports. Thus, the accuracy of automated team assignment is not higher, but it is much faster as the system can suggest team assignments for newly submitted issue reports in an instant.

CIA requires a high degree of information seeking, both in the source code and the project documentation. Analyzing impact prior to changing source code is mandated by development processes in safety-critical domains, but it is a tedious and error-prone activity. We present a recommendation system that supports CIA for changes caused by correcting reported issues. The system, ImpRec, builds a knowledge base by extracting previously reported impact originating from the corrective fixes of old issue reports. An industrial case study is ongoing, but initial results from an in-vitro evaluation indicate that more than 40% of the “true” impact is identified among the 10 first recommendations.

Future work includes completing the industrial case study on CIA at an industrial partner in Sweden and conducting post-mortem interviews. Before that however, we plan to initiate a replication of the case study in India, to enable additional feedback from another set of developers. Finally, we hope to explore how issue reports can be used to provide decision support for other navigational challenges in the software engineering context, such as fault localization and analysis of feedback from end-users.
Acknowledgments
This work was funded by the Industrial Excellence Center EASE – Embedded Applications Software Engineering.

6. REFERENCES

7http://ease.cs.lth.se