An Automatic Approach to Detect Unusual Events in Software Repositories

Larissa Leite, Christoph Treude, Fernando Figueira Filho

[1] Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte (UFRN) – Natal, RN – Brazil

{larissaleite,ctreude,fmarquesfilho}@gmail.com

Abstract. *This work presents an automatic approach to detect unusual events in software repositories. The approach collects data from source code repositories and analyzes new commits based on historical data in order to detect unusual events that are displayed to developers and managers in an awareness tool.*

1. Introduction

Software development teams use source control repositories and issue trackers to support their development processes and activities. Managers can use information extracted from such tools to become aware of the team productivity, and plan the cost and time of future releases [Weiss et al. 2007], while developers are fed with insight into workspaces of other developers [Treude and Storey 2010]. Such insight is usually provided by awareness tools. Awareness is defined as *"an understanding of the activities of others, which provide context for your own activity"* [Dourish and Bellotti 1992]. Since the success of software projects largely depends on the effectiveness of communication and coordination, software development teams need to maintain awareness of different aspects ranging from overall project status and process bottlenecks to current tasks and incoming artifacts [Treude and Storey 2010].

Despite the large amount of previous work on awareness, no tool or approach has specifically focused on awareness of unusual events in a software repository. Being aware of unusual events can be useful in preventing errors, but also in alerting developers and managers of events that may require justification or that can affect the work of other developers, especially when they relate to significant changes to the project. The motivation of this work comes not only from input from fellow developers, but also from a recent survey with 156 GitHub users [Treude et al. 2015], in which developers reported the need to be aware of unusual events: *"Commits that take particularly long might be interesting. If a developer hasn't committed anything in a while, his first commit after a long silence could be particularly interesting, for example, because it took him a long time to fix a bug. Also, important commits might have unusual commit messages, [...] indicating that the developer was emotional about that particular commit"*. Another developer added: *"Changes to files that haven't been changed in a long time or changes to a large number of files, a large number of deletions, etc."*. This work proposes a mechanism to automatically detect such unusual events, and make managers and developers aware of them.

2. Method to Detect Unusual Events

To validate our proposal, we are using data from the software repository of Superintendência de Informática (SINFO), a company that belongs to Universidade Federal do

Rio Grande do Norte (UFRN). SINFO is responsible for the development and maintenance of all information systems used by employees, students, and faculties at the university. There are more than 75 developers, testers, and requirement analysts working at SINFO, using Apache Subversion (SVN) as their source control repository.

Unusual – or unexpected – events are, by definition, events that are not in conformance with normality. The first step to detect such events is to determine what is normal, which, of course, depends on the context being analyzed. In our work, this is done by gathering historical data from software repositories, which is usually associated with commits, tasks, and issues or bugs. At the current step, our work relies mainly on commit-related data to investigate unusual events.

What is unusual depends on the development context, team size, work dynamics, software process, development cycle, domain, product size and criticality, as well as the development model (community-based open source or industry). Although this work is being conducted in the context of a specific software development team, we believe that the unusual events discussed in this work can be generalized to other contexts since we have chosen events that are likely to occur in any software project that uses source control. Future work needs to investigate this further. In the following, we describe different rules we use to detect unusual events in source control repositories.

Long time between commits. Time between commits is considered an indicator of project activity [Kolassa et al. 2013]. One or two working days without any commits from the whole team might be caused by infrastructure problems. From the developer point of view, time between commits can be a measurement of how difficult a task is or how challenging it was to fix a bug. In some cases, long time between commits may also be a potential cause of conflict when trying to incorporate changes from a local working copy to the current version of the project. We determine whether the time between commits is "long" in a given development context by calculating the mean time between commits in a project and by considering all those times that are longer than the mean plus two standard deviations.

Large number of files touched (added, modified, deleted) in the same commit. The number of files touched in a commit is especially important when inspecting added or deleted files, since modifying existing files is much more common than their creation or removal. Adding or deleting a considerable amount of files might be an indicator of changes to the software architecture and it is probably a sign of a refactoring or work on a disruptive task, i.e., a task that changes a lot and "disrupts" the current code. Again, we use the historical mean and standard deviation to define "large".

Large number of code modifications (LOC, methods, code complexity). Commits with a notably large number of changes to lines of code (LOC), methods, or code complexity may represent significant modifications to the code base, similar to what happens when many files are added, modified, or deleted in a single commit. Thus, it is important to notify the development team of such changes.

Modification in files without their related files. It is very common to have strongly coupled files that are often changed together in software development. ROSE [Zimmermann et al. 2005] is a tool for Eclipse aiming to: (i) suggest and predict likely changes; (ii) prevent errors due to incomplete changes; and (iii) de-

tect coupling undetectable by program analysis. ROSE uses the Apriori Algorithm [Agrawal and Srikant 1994] to compute association rules. Unlikely ROSE, the approach used in this work does not try to predict changes or prevent errors, but rather to notify developers of a possibly incomplete change after the commit.

Modification in files changed by many different developers. Files created and/or changed by many developers are more bug-prone than files only maintained by one or two developers, which was evidenced by a tool called Seesoft [Balsiger 2010]. In the security domain, source code files changed by many developers are also more likely to have at least one post-release security vulnerability [Meneely 2011]. Therefore, modifications to these specific files are worth mentioning to the development team. Similar to the rules presented above, we use the historical mean and standard deviation to determine what is "many" in a given development context.

Modification in files that had many modifications. The number of modifications made to a file during the lifetime of the project is a commonly analyzed factor in the area of software maintenance. [Graves et al. 2000] state that the number of times that the code has been changed is a good indication of how many faults it will contain. Although this work does not aim to predict defects, a modification to a file that has already been changed many times can be an indicator for instability in the code, and, thus, it is considered an important notification to the development team.

Modification in files not modified in a long time. Files that have not been modified in a long period of time can indicate two things: either the code is stable or it has been "forgotten" and it is not up to date with the current version/status of the project (architecture, requirements, etc).

Our approach stores data about unusual events in a database and presents it to managers and developers using a web application. The Data Extraction process is supported by a Java project called UEMiner, which consists of three main components: (i) Miner, (ii) DataCollector, and (iii) DataAnalyzer. Miner is responsible for accessing a source code repository and for communicating with the database in order to save the retrieved data. The DataCollector component consists of an infrastructure to collect and prepare data to allow the identification of unusual events. Since the data related to each event is different, there is one collector for each type of unusual event. To store the relevant data, the DataCollector component creates spreadsheets, along with a few related statistics – especially mean and standard deviation. The DataAnalyzer component analyzes the data collected and prepared by DataCollector aiming to identify unusual events by: (i) getting statistical information about the historical data from the spreadsheets, and (ii) comparing such information with data of the current commit being analyzed. If an outlier is identified, the event is saved to the database, allowing it to be accessed by the web application. The DataAnalyzer component constantly monitors the repository for new commits, but the analysis process can be triggered by other factors, such as specific days of the week. The whole process is illustrated in Figure 1.

3. Future Work

The next steps for this work involve the evaluation of the proposed approach. Events are displayed to developers and managers in an awareness tool called UEDashboard, which shows the events in a notification feed. Managers and developers can provide feedback

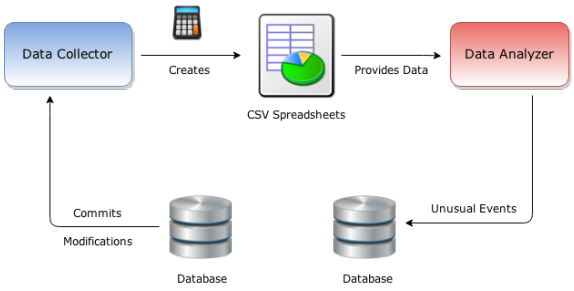

Figure 1. Process for Data Extraction and Analysis.

by classifying the event as useful or not useful, and they can write comments about each notification. In future work it might be possible for the tool to learn from these inputs to understand what is relevant for the development team and provide better notifications. We also intend to interview development teams to deeply understand what is behind unusual events. Additionally, we aim to apply our method to different development teams, since teams with different characteristics – team size, project age, development process – may require different analysis and could bring up various other types of unusual events. We also plan to extend our work beyond version control systems by additionally analyzing data from issue trackers, communication channels, and release management systems.

References

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, pages 487–499.

Balsiger, M. (2010). Representing software features in the Eclipse IDE. Univ. of Bern.

Dourish, P. and Bellotti, V. (1992). Awareness and coordination in shared workspaces. In *Proc. of the Conf. on Computer-supported Cooperative Work*, pages 107–114.

Graves, T. L., Karr, A. F., Marron, J. S., and Siy, H. (2000). Predicting fault incidence using software change history. *IEEE Trans. on Software Engineering*, 26(7):653–661.

Kolassa, C., Riehle, D., and Salim, M. A. (2013). The empirical commit frequency distribution of open source projects. In *Proc. of the 9th Intl. Symp. on Open Collaboration*, pages 18:1–18:8.

Meneely, A. (2011). *Investigating the Relationship Between Developer Collaboration and Software Security*. PhD thesis. North Carolina State Univ.

Treude, C., Figueira Filho, F., and Kulesza, U. (2015). Summarizing and measuring development activity. *Submitted to Symp. on the Foundations of Software Engineering*.

Treude, C. and Storey, M.-A. (2010). Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *Proc. of the 32nd Intl. Conf. on Software Engineering*, pages 365–374.

Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. (2007). How long will it take to fix this bug? In *Proc. of the 4th Intl. Workshop on Mining Software Repositories*, page 1.

Zimmermann, T., Zeller, A., Weissgerber, P., and Diehl, S. (2005). Mining version histories to guide software changes. *IEEE Trans. on Software Engineering*, 31(6):429–445.