Report on MSR 2005: International Workshop on Mining Software Repositories

Stephan Diehl
Computer Science
Catholic University Eichstätt
Eichstätt, Germany
diehl@acm.org

Ahmed E. Hassan and Richard C. Holt
Software Architecture Group (SWAG)
School of Computer Science
University of Waterloo
Waterloo, Canada
{aeehassa, holt}@plg.uwaterloo.ca

Abstract

A one-day workshop on the topic of Mining Software Repositories (MSR) was held at ICSE 2005 in St. Louis, Missouri. Researchers and practitioners in the MSR field try to transform static record keeping software repositories to active ones. These repositories permit researchers to gain empirically based understanding of software development, while software practitioners use these repositories to predict and plan various aspects of their project.

Following the success of last year's workshop, MSR 2005 had a large number of high quality submissions and a great number of participants. 22 papers were accepted from 38 submissions – 11 papers were presented as Lightning talks (5 mins) and another 11 papers were presented as regular talks (15 mins). The Lighting talks were followed with a walk-around demo and discussion session.

This report includes an overview of the presentations made during the day and a summary of the issues raised throughout the workshop.

Introduction

Software repositories such as source control systems, archived communications between project personnel, and defect tracking systems are used to help manage the progress of software projects. Software practitioners and researchers are beginning to recognize the potential benefit of mining this information to support the maintenance of software systems, improve software design/reuse, and empirically validate novel ideas and techniques. Research is now proceeding to uncover the ways in which mining these repositories can help to understand software development, to support predictions about software development, and to plan various aspects of software projects.

Scope and Topics of Interest

We sought position papers that address issues along the general themes, including but not limited to the following:

- Approaches to study the quality of the mined data along with guidelines to ensure the quality of the recovered data
- Proposals for exchange formats, meta-models, and infrastructure tools to facilitate the sharing of extracted data and to encourage reuse and repeatability
- Models for social and development processes that occur in large software development projects
- Search techniques to assist developers in finding suitable components for reuse
- Techniques to model reliability and defect occurrences
- Analysis of change patterns to assist in future development
- Case studies on extracting data from repositories of large long lived projects
- Suggestions for benchmarks, consisting of large software repositories, to be shared among the community

Workshop Format

We received 38 papers from 14 countries. Papers were reviewed by the workshop's program committee in terms of their relevance to the aims of the workshop and their technical content. Accepted papers were posted on the workshop's web site prior to the workshop at:

`http://msr.uwaterloo.ca`

The workshop program was broken into five sessions: 4 with regular talks and one combined Lightning talks and demo session.

Regular talks were 15 minutes with one clarification question. At the end of each session we had an open discussion of all papers in that session. In contrast the lightning talks were only 5 minutes long with no clarification questions. The lightning talks were followed by a one hour walkaround demos and discussion session.

Workshop Sessions

Session 1: Evolution and Change Patterns

The papers in this session investigated how changes occur in evolving software systems and how to deal with the large amount of data.

Neamtiu et al. studied the effect of changes on the abstract syntax trees and found that change increases with depth within the tree. Williams et al. recovered system specific function usage patterns from the change history.

In a case study Fischer et al. looked at the BSD operating system and its offspring (NetBSD, FreeBSD, and OpenBSD) to see how the evolution of product families differs from that of single programs.

Finally, by looking at the evolution of code clones Kim and Notkin characterized different kinds of clones and found that there are actually good clones – those that are foreseen to diverge later in the development process.

Session 2: Defect Analysis

In general defect analysis is concerned with where bugs come from and how well we can predict them.

Sliwerski and Zimmermann tried to identify bug-inducing changes. These changes are followed later by bug-fixing changes which correct the bug-inducing changes. In particular they found, that bug-inducing changes occur more frequently on Fridays.

Görg and Weißgerber developed a method to automatically detect incomplete refactorings. Some of the incomplete refactorings that they found lead to compilable, but erroneous programs. So their version analysis could detect problems, that static analysis could not.

Session 3: Education

Papers in this session mined student projects to allow teachers to understand better the progress of students, instead of mining archives of medium to large open source programs.

Spacco et al. investigated the ability of warnings produced by different static analyzers to predict exceptions raised when testing the program.

Mierle et al. extracted various quantitative measures from 200 CVS archives of student projects and tried to relate these with grades. Surprisingly, no predictors stronger than simple lines-of-code were found.

Session 4: Lightning Talks

The talks in this session were divided into four themes:

4a) Text Mining Ohba and Gondow suggested to mine for concept keywords in identifiers. The key idea is that identifiers are divided into terms. Terms that occur frequently in a document are characteristic for the document. They applied the technique to relate bug reports and source code.

In a case study Ying et al. found different kinds of Eclipse comments that start with `TODO` and argued that these and other source code comments provide important information: "Someone left a note for you in the code". Hayes et al. undertook a pilot study to examine the impact of analyst decisions on the final outcome of the text mining process.

4b Software Changes and Evolution Kim et al. developed a taxonomy of function signature change patterns. They analyzed 8 open source software systems to see how often each of these change patterns occur.

Ratzinger et al. identified two bad change smells, i.e. bad practices of how to change code, and showed in a case study that these can be used to find bad smells in the source code.

In a case study by Antoniol et al., two techniques from signal processing, namely Linear Predictive Coding and Cepstral analysis, are used to identify files with similar size changes patterns.

4c) Process and Collaboration VanHilst et al. argue that mining software repositories provides useful process metrics without adding overhead to the process itself.

Huang and Liu applied social network analysis to divide modules into conceptual kernel and non-kernel modules, as well as developers into core and none-core teams. Huang and Liu received the Lightning award for the best presentation in this session.

4d) Taxonomies and Formal Representations Two different taxonomies were proposed:

Kagdi et al. proposed a taxonomy based on technical aspects, e.g. the kinds and granularity of mining, whereas German et al. considered the goals and context, i.e. the different kinds of users and their needs.

Hindle and German designed a query language to formulate hypotheses and reason about data in software repositories. Their language is based on a formal model of repositories consisting of four entities: authors, modification request, revisions, and files.

Session 5: Integration and Collaboration

Robles and González-Barahona combined various sources of data to map the different identities used by a developer in one or more open source projects to a single person.

Ohira et al. developed a graph-based tool to analyze and visualize the relationship among projects and developers. They found that about 66 percent of all projects at SourceForge had only one developer.

Conklin et al. reported about a repository for researchers to store and share meta-data (developer names, platforms, licence types, etc.) extracted from general repositories like SourceForge, GNU Savannah and the like.

1 Conclusions

Tools and approaches were presented that extract various kinds of information from software archives, identify potential bugs, or discover who are the core-team members of a project. The techniques applied ranged from classical data mining to signal processing. In the discussions the question of what are the underlying heuristics and concerns about their validity, were raised several times. Also privacy issues

were discussed: although the archives are publicly available, should the mining results be publicly available as the results often provide condensed information about individual developers.

By looking at the data stored in software archives, researchers found that there are good code clones, that students who put spaces after commas get better grades and that programmers should not work on Fridays. As more of these tools become available, they will enable us to see whether these findings generalize to our own projects or students.

Finally, there are plans to hold MSR as a two days workshop in 2006 including a posters session and a mining challenge task so we can gain a better understanding of the strength and weakness of the various proposed approaches in the field.