07491 Abstracts Collection
Mining Programs and Processes
— Dagstuhl Seminar —

A. Bernstein[1], H.Gall[2] and A. Zeller[3]

[1] Univ. of Zürich, CH
bernstein@ifi.unizh.ch
[2] Univ. of Zürich,
[3] Univ. of Saarbrücken, DE
zeller@cs.uni-sb.de

Abstract. From 02.12. to 17.12.2007, the Dagstuhl Seminar 07491 "Mining Programs and Processes" was held in Schloss Dagstuhl – Leibniz Center for Informatics. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Mining software archives, data mining, machine learning, empirical software engineering

07491 Executive Summary – Mining Programs and Processes

The main goal of the seminar "Mining Programs and Processes" was to create a synergy between researchers of three communities, namely mining software repositories, data mining and machine learning, and empirical software engineering.

Keywords: Mining software archives, data mining, machine learning, empirical software engineering

Joint work of: Bernstein, Abraham; Gall, Harald; Zeller, Andreas

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2009/2246

Mining Application Code for Domain-Specific Languages

Krzysztof Czarnecki (University of Waterloo, CA)

Domain-specific languages (DSLs) have recently gained widespread attention of the software engineering community as promising approach to boost productivity and quality in specialized application areas. Unfortunately, designing DSLs is a challenging task and not much engineering science exists to support it. In well-defined, mature domains DSLs may be extracted form the established practices of domain experts or created based on an existing mathematical theory that underlies the domain. However, perhaps the majority of domains have an emerging nature: they emerge from the practice of a particular organization or community and are a moving target because of market evolution.

In this talk I will report on research that I recently embarked on. The goal of the research is to demonstrate the feasibility of mining application code for framework-specific modeling languages (FSMLs). FSMLs are is a special kind of DSLs that were designed to represent a set of concepts provided by an object-oriented framework. The long-term goal of this research is to achieve the incremental grow and evolution of higher-level DSL from the usage of lower-level abstractions, which could be an effective approach to develop DSLs for emerging domains.

Interest in Mining

Serge Demeyer (University of Antwerpen, B)

In the Lab on Reengineering (LORE) we have been using mining techniques for two purposes. On the one hand, we mined version repositories for finding traces of refactorings (duplicated code that has been removed, ...). On the other hand, we mined execution traces of running systems to calculate coupling and cohesion and derive key classes in the system design. One area of future research is performing joint case-studies to derive common benchmarks.

Keywords: Mining refactorings, mining execution traces, benchmarks

Is this Anytheeeeeeng? (with apologies to David Letterman)

Premkumar T. Devanbu (Univ. of California - Davis, USA)

We all want to do something, something important. But first, we have to know what <nothing> is. We also have to know that what we have found is not <nothing>. Once we know it's something, we have to make sure it's not something we found by looking too hard; after that, we have to make sure we didn't find something just an artifact of the tools we were using to find something.

Finally we have to make sure that what we found is something useful.

Keywords: Humor, null hypothesis

Abstract

Thomas Fritz (University of British Columbia - Vancouver, CA)

When building a software system, software developers each contribute a flow of information that together forms the system. As they work, programmers continuously consult various facts (knowledge) about this information to answer their questions about the system. The knowledge most easily accessed today in a programming environment involves facts about the structure of the program. However, the knowledge required by a programmer is broader than just structure; it also includes knowledge about design, requirements, and the development process, to name just a few other sources. To enable developers to access this knowledge more efficiently, our goal is to develop a model for programming environments that allows various fragments of different kinds of knowledge to be configured flexibly.

This model would enable new presentations to show these knowledge fragments in ways that more directly answer programmers' questions.

Evolizer: A framework for software evolution analysis

Harald Gall (Universität Zürich, CH)

Our research in software engineering is focused on technologies that enable the development of large, complex, and long-living software systems. For that, software development methodologies and paradigms are needed to provide evolvability and maintainability characteristics of software. We mine all kinds of software repositories to explore ways of learning from the past.

We have built an analysis framework called Evolizer that povides a framework for change type analysis, architectural analysis, metrics, social networks, and release history based visualizations.

Keywords: Software evolution, software visualization

Fast, Cheap, and Under Control: Evaluating Revision Data Reliably

Michael W. Godfrey (University of Waterloo, CA)

Measuring characteristics of software systems can be as deep or as shallow as time and budget allow.

However, industrial users have a strong preference for techniques that are lightweight and fast. In this talk, we will present our approach to measuring some characteristics of software revisions, and also show how our background work has established the reliability of these techniques using a deep, heavyweight evaluation.

Keywords: Understanding software revisions

Joint work of: Hindle, Abram; Godfrey, Michael W.; Holt, Richard C.

Using Text Mining and Link Analysis for Software Mining

Miha Grcar (Jozef Stefan Institute - Ljubljana, SLO)

Many data mining techniques are these days in use for ontology learning Ű text mining, Web mining, graph mining, link analysis, relational data mining, and so on. In the current state-of-the-art bundle there is a lack of Şsoftware miningŤ techniques. This term denotes the process of extracting knowledge out of source code. In this paper we approach the software mining task with a combination of text mining and link analysis techniques. We discuss how each instance (i.e. a programming construct such as a class or a method) can be converted into a feature vector that combines the information about how the instance is interlinked with other instances, and the information about its (textual) content. The so-obtained feature vectors serve as the basis for the construction of the domain ontology with OntoGen, an existing system for semi-automatic data-driven ontology construction.

Keywords: Software mining, text mining, link analysis, graph and network theory, feature vectors, ontologies, OntoGen, machine learning

Joint work of: Grcar, Miha; Grobelnik, Marko; Mladenic, Dunja

See also: Workshop on Mining Complex Data, ECML/PKDD 2007

Using Term-matching Algorithms for the Annotation of Geo-services

Miha Grcar (Jozef Stefan Institute - Ljubljana, SLO)

This paper presents an approach for automating semantic annotation within service-oriented architectures that provide interfaces to databases of spatial-information objects. The automation of the annotation process facilitates the transition from the current state-of-the-art architectures towards semantically-enabled architectures. We see the annotation process as the task of matching an arbitrary word or term with the most appropriate concept in the domain ontology. The term matching techniques that we present are based on text mining. To determine the similarity between two terms, we first associate a set of

documents [that we obtain from a Web search engine] with each term. We then transform the documents into feature vectors and thus transition the similarity assessment into the feature space. After that, we compute the similarity by training a classifier to distinguish between ontology concepts. Apart from text mining approaches, we also present two alternative techniques, namely hypothesis checking (i.e. using linguistic patterns such as "term1 is a term2" as a query to a search engine) and Google Distance.

Keywords: Geo-services, semantic annotation, text mining, search engine querying, machine learning, term matching

Joint work of: Grcar, Miha; Klien, Eva

See also: Workshop on Web Mining 2.0, ECML/PKDD 2007

Automatic Verification of Load Tests

Zhen Ming Jiang (University of Victoria, CA)

Load testing mimics multiple users performing tasks at the same time, and usually lasts for many hours or a few days. In practice, a load test run is considered successful if the system did not crash or did not suffer significant delays. However, these success criteria are not sufficient to verify whether the system has behaved properly throughout the course of the load test run. In this talk, we propose mining the large number execution logs produced during a load test to identify execution anomalies. We propose several log mining techniques and evaluate their success in identifying errors during load testing of large industrial projects.

Keywords: Load Test, Log Analysis, Log Mining, Anomaly Detection

Joint work of: Jiang, Zhen Ming; Ahmed E. Hassan

Discovering and Representing Logical Structure in Code Changes

Miryung Kim (University of Washington, USA)

There is a significant gap between how programmers think about code change and how change is represented in most software engineering tools. Programmers often think about code change in terms of structure: which code elements changed and how their structural dependencies are affected by the change. By reasoning about structural information within and around changed code, they recognize high-level systematic changes such as refactorings and crosscutting changes. Yet, most software engineering tools are based on a textual representation of code change.

To bridge this gap, we propose a novel rule-based delta representation that explicitly and concisely captures systematic changes to a program's structure, along with an engine that automatically infers such rules.

Our logical structural delta (LSD) can complement existing uses of textual deltas: e.g., understanding another programmer's modification, reviewing a patch before submission, and writing change documentation. We believe that LSD may serve as a basis for many software engineering tools that can benefit from explicit logical structure in code change: a bug finding tool, a refactoring reconstruction tool, a dependency removal checker, etc.

Keywords: Code change, delta, systematic changes, software evolution

Learn to refactor business processes

Volker Klingspor (FH Bochum, D)

The last years have shown that programming business applications from scratch is a very risky task. Most of the projects didn't succeed. They did not finished in time, in budget or did not finished at all. Thus, there has been a drift in the overall architecture of software towards service oriented architectures (SOA), and to the integration of as much existing (legacy) functionality as possible. If software is seen as a collection of services, the next step is to model the flow of activities in a business process graphically, instead of programming it using a classical programming language. From this idea, business modeling notations like EPK and BPMN and model execution languages (like BPML and BPEL) come into being.

Even if programming can be replaced by graphical modeling, the need for flexible and extensible software demands for modeling tools allowing a high agility. Especially, it should be easy to restructure previously modeled processes to be able to react to the changes in the business market.

The idea of restructuring software isn't new. In the 90th the concept of refactoring aroused in "classical" programming. What is necessary to transfer the results of refactoring of object oriented programs to refactoring of business process models? Firstly, we have to gather a set of situations, where the structure of a business process model has room for improvement. Secondly, we must describe for each of these situations, how it can be described, in which way the process could be restructured, and what the benefits of the restructuring are. And thirdly, to support the business modeler, the modeling environment should autonomously analyze business models and suggest refactoring candidates. That means, the modeling system itself should propose restructuring steps. Therefore, we need a set of formal features describing exactly the characteristics of business processes and parts of the processes with respect to the applicability of refactoring. And we need heuristics to decide, for which part of a process a dedicated refactoring step seems sensible. Maybe, we can learn these heuristics automatically from previous refactoring steps.

An example for a business model refactoring pattern could be the pattern of sub processes. If a process is large (i.e., it consists of more elements than some threshold), and it contains a part, that is relatively autonomous (i.e. for example, this part needs relatively few data from the remaining process and it doesn't branch out without being merged), then this part of the process should be sourced out to a new sub process. Furthermore, a new interface for this sub process must be specified, and the sub process must be called from original process.

Having such restructuring capabilities embedded in the business process modeling software, the development of processes could be much more incremental and more agile.

Sourcerer and CodeGenie: Collecting, Slicing and Dicing Open Source code

Cristina Lopes (Univ. California - Irvine, USA)

Sourcerer is an infrastructure for collecting and indexing open source code available on the internet. I will describe the studies and tools we are developing with this infrastructure.

Machine Learning Supports Processes

Katharina Morik (Universität Dortmund, D)

The old dream of learning programs from input-output examples has changed over the years. On one hand, XML has become the way to express input as well as output of programs and is the key to web application programs. On the other hand, descriptions of programs allow standard learning algorithms to support developers.

The talk gives some examples of applying machine learning to software engineering.

Can Developer Social Networks Predict Failures?

Martin Pinzger (Universität Zürich, CH)

Software teams should follow a well defined goal and keep their work focused. Work fragmentation is bad for efficiency and quality. In this paper we empirically investigate the relationship between the fragmentation of developer contributions and the number of post-release failures. Our approach is to represent the structure of developer contributions with a contribution network.

We use social network centrality measures to measure the degree of fragmentation of developer contributions. Fragmentation is determined by the centrality of software modules in the contribution network. Our claim is that central software modules are more likely to be failure-prone than modules located in surrounding areas of the contribution network. We analyze this hypothesis by exploring the network centrality of Microsoft Windows Vista modules using several social network centrality measures as well as linear and logistic regression analysis. In particular, we investigate which centrality measures are significant to predict the probability and number of post-release failures.

Results of our experiments show that central modules are more failure-prone than modules located in surrounding areas of the network. The basic centrality measures, number of authors and number of commits, are significant predictors for the probability of failures. For predicting the number of post-release failures the closeness centrality measures are most significant.

Keywords: Failure Prediction, Social Network Analysis, Developer Contribution Network, Network Centrality Measures

Joint work of: Pinzger, Martin; Nagappan, Nachiappan; Murphy, Brendan

Backstory: A Search Tool for Software Developers Supporting Scalable Sensemaking

Gina Venolia (Microsoft Corp. - Redmond, USA)

Software developers have many information needs that that could be answered using written text in the various repositories at their disposal, but they underutilize these knowledge resources for a variety of good reasons. Backstory is a search tool for software developers aimed at addressing those reasons, and so to improve knowledge flow among teammates. The results of a survey of developers current search habits and desires for a new tool are presented as background. A case study of root-cause analysis is also presented, which informs the design of Backstory. Additionally the case study adds detail to an accepted model of sensemaking. Finally, the Backstory UI design suggests that a tool can support sensemaking in such a way that it's not intimidating or distracting for simple investigations, but has mechanisms that the user may employ incrementally as the complexity of an investigation increases - a characteristic referred to here as scalable sensemaking.

Collaborative Software Evolution Analysis

Marco d'Ambros (University of Lugano, CH)

We present Churrasco, a framework to support collaborative software evolution analysis and visualization. The main features of the framework are:

- Flexible and extensible meta-model support. The meta-model used to describe the evolution of a software system can be dynamically changed and/or extended.

- Accessibility. The framework is fully web-based, i.e., the entire analysis of a software system, from the initial model creation to the final study, can be performed from a web browser, without having to install or configure any tool.

- Collaboration. Churrasco relies on a centralized database and supports annotations. Thus, the knowledge of a system, gained during the analysis, can be incrementally stored into the model of the system itself. We show, through a simple example scenario, how Churrasco supports collaborative software evolution analysis and visualization.

Keywords: Software Evolution, Software Visualization, Collaborative Analysis, Metamodeling