ICPC 2012

Passau, Germany, June 11–13

— Program —

Events of ICPC 2012 Mon 2012-06-11 – Wed 2012-06-13 (Berlin)

	Mon 6/11	Tue 6/12	Wed 6/13
09:00	Opening 09:00 - 09:20 Invited: Agile Software Assessment (Invited	Invited: Studying Developers for Fun and Profit (Keynote Abstract) 09:00 - 10:15	Invited: A Retrospective View on: The Role of Empirical Studies 09:30 - 10:30
10:00	Coffee Break 10:30 - 11:00	Coffee Break 10:15 - 10:45 Measurement 10:45 - 12:30	Coffee Break 10:30 - 11:00
11:00	Cognitive Processes 11:00 - 12:45		Semantics and Traceability 11:00 - 12:45
12:00	Lunch Break 12:45 - 14:15	Lunch Break 12:30 - 13:45	Lunch Break 12:45 - 14:15
13:00		Open Steering-Committee Meeting	
14:00	The Role of the Developer 14:15 - 15:45	Understanding and Architecture 14:15 - 15:45	Source-Code Analysis 14:15 - 15:45
15:00	Excursion to Klettergarten @ Hochseilgarten Passau 15:45 - 22:00	Tea Break 15:45 - 16:15	Tea Break 15:45 - 16:15
16:00		Tools: ICPC 2012 16:15 - 17:00	Student: ICPC 2012 16:15 - 17:45
17:00		Posters: ICPC 2012 17:00 - 17:45	Closing 17:45 - 18:00
18:00		Banquet Cruise @ River Danube 18:30 - 22:00	
19:00			Org Committee Dinner 19:00 - 21:00
20:00			
21:00			

Invited Papers

Agile Software Assessment (Keynote)

(Mon, Jun 11, 09:20 – 10:30, Chair: Arie van Deursen / Mike Godfrey)

Oscar Nierstrasz and Mircea Lungu

University of Bern, Switzerland

Abstract: Informed decision making is a critical activity in software development, but it is poorly supported by common development environments, which focus mainly on low-level programming tasks. We posit the need for *agile software assessment*, which aims to support decision making by enabling rapid and effective construction of software models and custom analyses. Agile software assessment entails gathering and exploiting the broader context of software information related to the system at hand as well as the ecosystem of related projects, and beyond to include "big software data". Finally, informed decision making entails continuous assessment by monitoring the evolving system and its architecture. We identify several key research challenges in supporting agile software assessment by focusing on customization, context and continuous assessment.

Studying Developers for Fun and Profit (Keynote)

(Tue, Jun 12, 09:00 – 10:15, Chair: Dirk Beyer)

Robert DeLine

Microsoft Research, USA

Abstract: My group at Microsoft Research creates software development tools through user-centered design. This method creates a virtuous cycle: we study developers and their teams, which in turn inspires the tools we design, which we then evaluate with those developers and teams, seeking to improve the nature of their work. In this talk, we'll discuss some of the biggest problem areas we have observed, including information seeking, multitasking and disorientation, and look at some of the prototypes we have built in response. Code Canvas provides a zoomable map of a software project, allowing the programmer to zoom out to see structure and visualizations and zoom in to edit code. Debugger Canvas (a joint project with Brown University) provides a spatial representation of a programmer's task, like a debugging session, as it unfolds. Finally, Code Space uses a combination of touch screens, Kinects and mobile devices to allow fluid sharing of digital objects at development team meetings.

A Retrospective View on: The Role of Concepts in Program Comprehension (MIP Award)

(Wed, Jun 13, 09:00 – 09:30)

Václav Rajlich and Norman Wilde

Wayne State University, USA; University of West Florida, USA

Abstract: This retrospective briefly recapitulates highlights of the original paper that was published at IWPC 2002. Then it overviews research directions of the last 10 years: research in tools and techniques of concept location a that aim to support software developer, research of integrated model of software change, creation of software engineering course that emphasizes the role of software developer in iterative and agile software processes, and further basic research into the role and properties of concepts.

Main Research Track

Cognitive Processes

(Mon, Jun 11, 11:00 – 12:45, Chair: Chris Parnin)

Program Complexity Metrics and Programmer Opinions

Bernhard Katzmarski and Rainer Koschke

University of Bremen, Germany

Abstract: Various program complexity measures have been proposed to assess maintainability. Only relatively few em- pirical studies have been conducted to back up these assess- ments through empirical evidence. Researchers have mostly conducted controlled experiments or correlated metrics with indirect maintainability indicators such as defects or change frequency. This paper uses a different approach. We investigate whether metrics agree with complexity as perceived by programmers. We show that, first, programmers' opinions are quite similar and, second, only few metrics and in only few cases reproduce complexity rankings similar to human raters. Data-flow metrics seem to better match the viewpoint of programmers than control-flow metrics, but even they are only loosely correlated. Moreover we show that a foolish metric has similar or sometimes even better correlation than other evaluated metrics, which raises the question how meaningful the other metrics really are. In addition to these results, we introduce an approach and associated statistical measures for such multi-rater investiga- tions. Our approach can be used as a model for similar studies.

Women and Men- Different but Equal: On the Impact of Identifier Style on Source Code Reading

Zohreh Sharafi, Zéphyrin Soh, Yann-Gaël Guéhéneuc, and Giuliano Antoniol

École Polytechnique de Montréal, Canada; University of Ngaoundéré, Cameroon

Abstract: Program comprehension is preliminary to any program evolution task. Researchers agree that identifiers play an important role in code reading and program understanding activities. Yet, to the best of our knowledge, only one work investigated the impact of gender on the memorability of identifiers and thus, ultimately, on program comprehension. This paper reports the results of an experiment involving 15 male subjects and nine female subjects to study the impact of gender on the subjects' visual effort, required time, as well as accuracy to recall Camel Case versus Underscore identifiers in source code reading.

We observe no statistically-significant difference in term of accuracy, required time, and effort. However, our data supports the conjecture that male and female subjects follow different comprehension strategies: female subjects seem to carefully weight all options and spend more time to rule out wrong answers while male subjects seem to quickly set their minds on some answers, possibly the wrong ones. Indeed, we found that the effort spent on wrong answers is significantly higher for female subjects and that there is an interaction between the effort that female subjects invested on wrong answers and their higher percentages of correct answers when compared to male subjects.

A Lightweight Visualization of Interprocedural Data-Flow Paths for Source Code Reading

Takashi Ishio, Shogo Etsuda, and Katsuro Inoue

Osaka University, Japan

Abstract: To understand the behavior of a program, developers must read source code fragments in various modules. For developers investigating data-flow paths among modules, a call graph is too abstract since it does not visualize how parameters of method calls are related to each other. On the other hand, a system dependence graph is too fine-grained to investigate interprocedural data-flow paths. In this research, we propose an intermediate-level of visualization; we visualize interprocedural data-flow paths among method parameters and fields with summarized intraprocedural data-flow paths. We have implemented our visualization as an Eclipse plug-in for Java. The tool comprises a lightweight data-flow analysis and an interactive graph viewer using fractal value to extract a small subgraph of data-flow related to variables specified by a developer. A case study has shown our visualization enabled developers to investigate more data-flow paths in a fixed time slot. In addition, we report our lightweight data-flow analysis can generate precise data-flow paths for 98% of Java methods.

Is the Derivation of a Model Easier to Understand Than the Model Itself?

Janet Feigenspan, Don Batory, and Taylor Riché

University of Magdeburg, Germany; University of Texas, USA; National Instruments, USA

Abstract: Software architectures can be presented by graphs with components as nodes and connectors as edges. These graphs, or models, typically encode expert domain knowledge, which makes them difficult to understand. Hence, instead of presenting a complete complex model, we can derive it from a simple, easy-to-understand model by a set of easy-to-understand transformations. In two controlled experiments, we evaluate whether a derivation of a model is easier to understand than the model itself.

The Role of the Developer

(Mon, Jun 11, 14:15 – 15:45, Chair: Andy Begel)

Evaluating Forum Discussions to Inform the Design of an API Critic

Chandan R. Rupakheti and Daqing Hou

Clarkson University, USA

Abstract: Learning to use a software framework and its API (Application Programming Interfaces) can be a major endeavor for novices. To help, we have built a critic to advise the use of an API based on the formal semantics of the API. Specifically, the critic offers advice when the symbolic state of the API client code triggers any API usage rules. To assess to what extent our critic can help solve practical API usage problems and what kinds of API usage rules can be formulated, we manually analyzed 150 discussion threads from the Java Swing forum. We categorize the discussion threads according to how they can be helped by the critic. We find that API problems of the same nature appear repeatedly in the forum, and that API problems of the same nature can be addressed by implementing a new API usage rule for the critic. We characterize the set of discovered API usage rules as a whole. Unlike past empirical studies that focus on answering why frameworks and APIs are hard to learn, ours is the first designed to produce systematic data that have been directly used to build an API support tool.

Mining Source Code Descriptions from Developer Communications

Sebastiano Panichella, Jairo Aponte, Massimiliano Di Penta, Andrian Marcus, and Gerardo Canfora

University of Sannio, Italy; Universidad Nacional de Colombia, Colombia; Wayne State University, USA

Abstract: Very often, source code lacks comments that adequately describe its behavior. In such situations developers need to infer knowledge from the source code itself or to search for source code descriptions in external artifacts.
We argue that messages exchanged among contributors/developers, in the form of bug reports and emails, are a useful source of information to help understanding source code. However, such communications are unstructured and usually not explicitly meant to describe specific parts of the source code. Developers searching for code descriptions within communications face the challenge of filtering large amount of data to extract what pieces of information are important to them. We propose an approach to automatically extract method descriptions from communications in bug tracking systems and mailing lists.
We have evaluated the approach on bug reports and mailing lists from two open source systems (Lucene and Eclipse). The results indicate that mailing lists and bug reports contain relevant descriptions of about 36% of the methods from Lucene and 7% from Eclipse, and that the proposed approach is able to extract such descriptions with a precision of up to 79% for Eclipse and 87% for Lucene. The extracted method descriptions can help developers in understanding the code and could also be used as a starting point for source code re-documentation.

Measuring Programming Experience

Janet Feigenspan, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg

University of Magdeburg, Germany; Philipps University of Marburg, Germany; University of Passau, Germany; University of Duisburg-Essen, Germany

Abstract: Programming experience is an important confounding parameter in controlled experiments regarding program comprehension. In literature, ways to measure or control programming experience vary. Often, researchers neglect it or do not specify how they controlled it. We set out to find a well-defined understanding of programming experience and a way to measure it. From published comprehension experiments, we extracted questions that assess programming experience. In a controlled experiment, we compare the answers of 128 students to these questions with their performance in solving program-comprehension tasks. We found that self estimation seems to be a reliable way to measure programming experience. Furthermore, we applied exploratory factor analysis to extract a model of programming experience. With our analysis, we initiate a path toward measuring programming experience with a valid and reliable tool, so that we can control its influence on program comprehension.

Measurement

(Tue, Jun 12, 10:45 – 12:30, Chair: Denys Poshyvanyk)

High-MCC Functions in the Linux Kernel

Ahmad Jbara, Adam Matan, and Dror G. Feitelson

Hebrew University of Jerusalem, Israel

Abstract: McCabe's Cyclomatic Complexity (MCC) is a widely used metric for the complexity of control flow. Common usage decrees that functions should not have an MCC above 50, and preferably much less. However, the Linux kernel includes more than 800 functions with MCC values above 50, and over the years 369 functions have had an MCC of 100 or more. Moreover, some of these functions undergo extensive evolution, indicating that developers are successful in coping with the supposed high complexity. We attempt to explain this by analyzing the structure of such functions and showing that in many cases they are in fact well-structured. At the same time, we observe cases where developers indeed refactor the code in order to reduce complexity. These observations indicate that a high MCC is not necessarily an impediment to code comprehension, and support the notion that complexity cannot be fully captured using simple syntactic code metrics.

Understanding Registration-Based Abstractions: A Quantitative User Study

John-Jose Nuñez and Gregor Kiczales

University of British Columbia, Canada

Abstract: The adoption of programming language innovation is impeded because all program processing tools in the tool chain must support any new or altered language features. Registration-based abstractions (RBAs) were proposed to address this difficulty by allowing the editor to transiently superimpose new language abstractions on existing code. Individual programmers can choose where and when to see a new language abstraction, while at all times the underlying code remains written in the original language. Prior work demonstrated the feasibility of RBAs, but left important questions unanswered regarding how users would interact with such an approach.
We asked 50 undergraduate students to answer basic program comprehension questions with and without RBAs. Our results show that participants can quickly and easily understand new abstractions without additional training, and suggest that this will extend to the general programmer community. The results also support a comparison across RBAs, and an initial discussion of specific features that facilitate or confound understanding.

Concern-Based Cohesion: Unveiling a Hidden Dimension of Cohesion Measurement

Bruno Silva, Claudio Sant'Anna, Christina Chavez, and Alessandro Garcia

Federal University of Bahia, Brazil; PUC-Rio, Brazil

Abstract: Cohesion has been avidly recognized as a key property of software modularity. Ideally, a software module is considered to be cohesive if it represents an abstraction of a single concern of the software. Modules with several concerns may be harder to understand because developers must mentally separate the source code related to each concern. Also, modules implementing several concerns are more likely to undergo changes as much as distinct development tasks may target its different concerns. The most well-known cohesion metrics are defined in terms of the syntactical structure of a module, and as a consequence fail to capture the amount of concerns realized by the module. In this context, we investigated the potential of a new metric, called Lack of Concern-based Cohesion. This metric explicitly counts the number of concerns realized by each module. We compared this metric with other five structural cohesion metrics by applying them over six open source software systems. We studied how those metrics are associated with module changes by mining over 16,000 repository revisions. Our results pointed out that the concern-based metric captured a cohesion dimension that is not reflected by structural metrics, and, as a consequence, adds to the association of cohesion and change-proneness.

Understanding Reuse in the Android Market

Israel J. Mojica Ruiz, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan

Queen's University, Canada; École Polytechnique de Montréal, Canada

Abstract: Mobile apps are software products developed to run on mobile devices, and are typically distributed via app stores. The mobile app market is estimated to be worth billions of dollars, with more than hundred of thousands of apps, and still increasing in number. This explosion of mobile apps is astonishing, given the short time span that they have been around. One possible explanation for this explosion could be the practice of software reuse. Yet, no research has studied such practice in mobile app development. In this paper, we intend to analyze software reuse in the Android mobile app market along two dimensions: (a) reuse by inheritance, and (b) class reuse. Since app stores only distribute the byte code of the mobile apps, and not the source code, we used the concept of Software Bertillonage to track code across mobile apps. A case study on thousands of mobile apps across five different categories in the Android Market shows that almost 23% of the classes inherit from a base class in the Android API, and 27% of the classes inherit from a domain specific base class. Furthermore, on average 61% of all classes in each category of mobile apps occur in two or more apps, and 217 mobile apps are reused completely by another mobile app in the same category.

Understanding and Architecture

(Tue, Jun 12, 14:15 – 15:45, Chair: Abram Hindle)

Programmer Information Needs after Memory Failure

Chris Parnin and Spencer Rugaber

Georgia Tech, USA

Abstract: Despite its vast capacity and associative powers, the human brain does not deal well with interruptions. Particularly in situations where information density is high, such as during a programming task, recovering from an interruption requires extensive time and effort. Although modern program development environments have begun to recognize this problem, none of these tools take into account the brain's structure and limitations. In this paper, we present a conceptual framework for understanding the strengths and weaknesses of human memory, particularly with respect to it ability to deal with work interruptions. The framework explains empirical results obtained from experiments in which programmers were interrupted while working. Based on the framework, we discuss programmer information needs that development tools must satisfy and suggest several memory aids such tools could provide. We also describe our prototype implementation of these memory aids.

Identifying Computational Phases from Inter-process Communication Traces of HPC Applications

Luay Alawneh and Abdelwahab Hamou-Lhadj

Concordia University, Canada

Abstract: Understanding the behaviour of High Performance Computing (HPC) systems is a challenging task due to the large number of processes they involve as well as the complex interactions among these processes. In this paper, we present a novel approach that aims to simplify the analysis of large execution traces generated from HPC applications. We achieve this through a technique that allows semi-automatic extraction of execution phases from large traces. These phases, which characterize the main computations of the traced scenario, can be used by software engineers to browse the content of a trace at different levels of abstraction. Our approach is based on the application of information theory principles to the analysis of sequences of communication patterns found in HPC traces. The results of the proposed approach when applied to traces of a large HPC industrial system demonstrate its effectiveness in identifying the main program phases and their corresponding sub-phases.

Tracking and Visualizing Information Flow in Component-Based Systems

Amir Reza Yazdanshenas and Leon Moonen

Simula Research Laboratory, Norway

Abstract: Component-based software engineering is aimed at managing the complexity of large-scale software development by composing systems from reusable parts. In order to understand or validate the behavior of a given system, one needs to acquire understanding of the components involved in combination with understanding how these components are instantiated, initialized and interconnected in the particular system. In practice, this task is often hindered by the heterogeneous nature of source and configuration artifacts and there is little to no tool support to help software engineers with such a system-wide analysis.

This paper contributes a method to track and visualize information flow in a component-based system at various levels of abstraction. We propose a hierarchy of 5 interconnected views to support the comprehension needs of both safety domain experts and developers from our industrial partner. We discuss the implementation of our approach in a prototype tool, and present an initial qualitative evaluation of the effectiveness and usability of the proposed views for software development and software certification. The prototype was already found to be very useful and a number of directions for further improvement were suggested. We conclude by discussing these improvements and lessons learned.

Empirical Studies

(Wed, Jun 13, 09:30 – 10:30, Chair: Thomas Zimmermann)

Do Static Type Systems Improve the Maintainability of Software Systems? An Empirical Study

Sebastian Kleinschmager, Stefan Hanenberg, Romain Robbes, Éric Tanter, and Andreas Stefik
University of Duisburg-Essen, Germany; University of Chile, Chile; Southern Illinois University at Edwardsville, USA

Abstract: Static type systems play an essential role in contemporary programming languages. Despite their importance, whether static type systems influence human software development capabilities remains an open question. One frequently mentioned argument for static type systems is that they improve the maintainability of software systems—an often used claim for which there is no empirical evidence. This paper describes an experiment which tests whether static type systems improve the maintainability of software systems. The results show rigorous empirical evidence that static type are indeed beneficial to these activities, except for fixing semantic errors.

Professional Status and Expertise for UML Class Diagram Comprehension: An Empirical Study

Zéphyrin Soh, Zohreh Sharafi, Bertrand Van den Plas, Gerardo Cepeda Porras, Yann-Gaël Guéhéneuc, and Giuliano Antoniol
École Polytechnique de Montréal, Canada; University of Ngaoundéré, Cameroon; University of Namur, Belgium; Université de Montréal, Canada

Abstract: Professional experience is one of the most important criteria for almost any job offer in software engineering. Profes-sional experience refers both to professional status (practitioner vs. student) and expertise (expert vs. novice). We perform an experiment with 21 subjects including both practitioners and students, and experts and novices. We seek to understand the relation between the speed and accuracy of the subjects and their status and expertise in performing maintenance tasks on UML class diagrams. We also study the impact of the formulation of the maintenance task. We use an eye-tracking system to gather the fixations of the subjects when performing the task. We measure the subjects' comprehension using their accuracy, the time spent, the search effort, the overall effort, and the question comprehension effort. We found that (1) practitioners are more accurate than students while students spend around 35 percent less time than practitioners, (2) experts are more accurate than novices while novices spending around 33 percent less time than experts, (3) expertise is the most important factor for accuracy and speed, (4) experienced students are more accurate and spend around 37 percent less time than experienced practitioners, and (5) when the description of the task is precise, the novice students can be accurate. We conclude that it is an illusion for project managers to focus on status only when recruiting a software engineer. Our result is the starting point to consider the differences between status and expertise when studying software engineers' productivity. Thus, it can help project managers to recruit productive engineers and motivated students to acquire the experience and ability in the projects.

Semantics and Traceability

(Wed, Jun 13, 11:00 – 12:45, Chair: Andy Kellens)

Modeling the Ownership of Source Code Topics

Christopher S. Corley, Elizabeth A. Kammer, and Nicholas A. Kraft

University of Alabama, USA

Abstract: Exploring linguistic topics in source code is a program comprehension activity that shows promise in helping a developer to become familiar with an unfamiliar software system. Examining ownership in source code can reveal complementary information, such as who to contact with questions regarding a source code entity, but the relationship between linguistic topics and ownership is an unexplored area. In this paper we combine software repository mining and topic modeling to measure the ownership of linguistic topics in source code. We conduct an exploratory study of the relationship between linguistic topics and ownership in source code using 10 open source Java systems. We find that classes that belong to the same linguistic topic tend to have similar ownership characteristics, which suggests that conceptually related classes often share the same owner(s). We also find that similar topics tend to share the same ownership characteristics, which suggests that the same developers own related topics.

A Semantic Relatedness Approach for Traceability Link Recovery

Anas Mahmoud, Nan Niu, and Songhua Xu

Mississippi State University, USA; Oak Ridge National Laboratory, USA

Abstract: Human analysts working with automated tracing tools need to directly vet candidate traceability links in order to determine the true traceability information. Currently, human intervention happens at the end of the traceability process, after candidate traceability links have already been generated. This often leads to a decline in the results' accuracy. In this paper, we propose an approach, based on semantic relatedness (SR), which brings human judgment to an earlier stage of the tracing process by integrating it into the underlying retrieval mechanism. SR tries to mimic human mental model of relevance by considering a broad range of semantic relations, hence producing more semantically meaningful results. We evaluated our approach using three datasets from different application domains, and assessed the tracing results via six different performance measures concerning both result quality and browsability. The empirical evaluation results show that our SR approach achieves a significantly better performance in recovering true links than a standard Vector Space Model (VSM) in all datasets. Our approach also achieves a significantly better precision than Latent Semantic Indexing (LSI) in two of our datasets.

Using IR Methods for Labeling Source Code Artifacts: Is It Worthwhile?

Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella

University of Salerno, Italy; University of Sannio, Italy; University of Molise, Italy

Abstract: Information Retrieval (IR) techniques have been used for various software engineering tasks, including the labeling of software artifacts by extracting "keywords" from them. Such techniques include Vector Space Models, Latent Semantic Indexing, Latent Dirichlet Allocation, as well as customized heuristics extracting words from specific source code elements.

This paper investigates how source code artifact labeling performed by IR techniques would overlap (and differ) from labeling performed by humans. This has been done by asking a group of subjects to label 20 classes from two Java software systems, JHotDraw and eXVantage. Results indicate that, in most cases, automatic labeling would be more similar to human-based labeling if using simpler techniques-e.g., using words from class and method names-that better reflect how humans behave. Instead, clustering-based approaches (LSI and LDA) are much more worthwhile to be used on source code artifacts having a high verbosity, as well as for artifacts requiring more effort to be manually labeled.

A TraceLab-Based Solution for Creating, Conducting, and Sharing Feature Location Experiments

Bogdan Dit, Evan Moritz, and Denys Poshyvanyk

College of William and Mary, USA

Abstract: Similarly to other fields in software engineering, the results of case studies involving feature location techniques (FLTs) are hard to reproduce, compare, and generalize, due to factors such as, incompatibility of different datasets, lack of publicly available implementation or implementation details, or the use of different metrics for evaluating FLTs. To address these issues, we propose a solution for creating, conducting, and sharing experiments in feature location based on TraceLab, a framework for conducting research. We argue that this solution would allow rapid advancements in feature location research because it will enable researchers to create new FLTs in the form of TraceLab templates or components, and compare them with existing ones using the same datasets and the same metrics. In addition, it will also allow sharing these FLTs and experiments within the research community. Our proposed solution provides (i) templates and components for creating new FLTs and instantiating existing ones, (ii) datasets that can be used as inputs for these FLTs, and (iii) metrics for comparing these FLTs. The proposed solution can be easily extended with new FLTs (in the form of easily configurable templates and components), datasets, and metrics.

Source-Code Analysis

(Wed, Jun 13, 14:15 – 15:45, Chair: Massimiliano Di Penta)

Can Clone Detection Support Test Comprehension?

Benedikt Hauptmann, Maximilian Junker, Sebastian Eder, Elmar Jürgens, and Rudolf Vaas

TU Munich, Germany; CQSE, Germany; Munich Re, Germany

Abstract: Tests are central artifacts of software systems. Therefore, understanding tests is essential for activities such as maintenance, test automation, and efficient execution. Redundancies in tests may significantly decrease their understandability. Clone detection is a technique to find similar parts in software artifacts. We suggest using this technique to gain a better understanding of tests and to provide guidance for testing activities. We show the capabilities as well as the limits of this approach by conducting a case study analyzing more than 4000 tests of seven industrial software systems.

A Controlled Experiment on Software Clones

Jan Harder and Rebecca Tiarks

University of Bremen, Germany

Abstract: Most software systems contain sections of duplicated source code-clones-that are believed to make maintenance more difficult. Recent studies tested this assumption by retro- spective analyses of software archives. While giving important insights, the analysis of historical data relies only on snapshots and misses the human interaction in between. We conducted a controlled experiment to investigate how clones affect the programmer's performance in common bug-fixing tasks. While our results do not exhibit a decisive difference in the time needed to correct cloned bugs, we observed many cases in which cloned bugs were not corrected completely.

Code Querying by UML

Carlos Noguera, Coen De Roover, Andy Kellens, and Viviane Jonckers

Vrije Universiteit Brussel, Belgium

Abstract: The need to identify source code that exhibits particular characteristics is essential to program comprehension. In this paper we introduce ARABICA, a tool for querying Java code using UML class and sequence diagrams. Our use of UML diagrams avoids the need for developers to familiarize themselves with yet another language. In contrast to tools that rely on dedicated query languages, ARABICA encodes querying semantics in a dedicated, minimal UML profile. Stereotyped class and sequence diagrams, characterizing structural and behavioral properties respectively, are translated into logic program queries. Using examples from the JHotDraw framework, we illustrate the utility of ARABICA in validating design invariants, finding design pattern implementations and exploring extension points. We present a pre/post-test quasi experiment as a preliminary assessment of our approach.

Tool Demonstrations

(Tue, Jun 12, 16:15 – 17:00, Chair: Abram Hindle / Chris Parnin)

CriticAL: A Critic for APIs and Libraries

Chandan R. Rupakheti and Daqing Hou

Clarkson University, USA

Abstract: It is well-known that APIs can be hard to learn and use. Although search tools can help find related code examples, API novices still face other significant challenges such as evaluating the relevance of the search results. To help address the broad problems of finding, understanding, and debugging API-based solutions, we have built a critic system that offers recommendations, explanations, and criticisms for API client code. Our critic takes API usage rules as input, performs symbolic execution to check that the client code has followed these rules properly, and generates advice as output to help improve the client code. We demonstrate our critic by applying it to a real- world example derived from the Java Swing Forum.

Supporting Comprehension Experiments with Human Subjects

Janet Feigenspan and Norbert Siegmund

University of Magdeburg, Germany

Abstract: Experiments with human subjects become more and more important in software engineering. To support planning, conducting, and replicating experiments targeting program comprehension, we developed PROPHET. It allows experimenters to easily define and customize experimental settings as well as to export settings such that others can replicate their results. Furthermore, PROPHET provides extension points, which allow users to integrate additional functionality.

SeByte: A Semantic Clone Detection Tool for Intermediate Languages

Iman Keivanloo, Chanchal K. Roy, and Juergen Rilling

Concordia University, Canada; University of Saskatchewan, Canada

Abstract: SeByte is a semantic clone detection tool which accepts Java bytecode (binary) as input. SeByte provides a complementary approach to traditional pattern-based source code level clone detection. It is capable of detecting clones missed by existing clone detection tools since it exploits both pattern and content similarity at binary level.

CRat: A Refactoring Support Tool for Form Template Method

Keisuke Hotta, Yoshiki Higo, Hiroshi Igaki, and Shinji Kusumoto

Osaka University, Japan

Abstract: Refactoring is important for efficient software maintenance. However, manual operations for refactoring are complicated, and human-related errors easily occur. Tool support can help users to apply such a complicated refactoring. This paper proposes a refactoring support tool with Form Template Method pattern. The developed tool automatically identifies method pairs that can be refactored with Form Template Method, and suggests information that is required for Form Template Method application. It also has a function that metrics-based filtering for detected method pairs. The function helps users to select method pairs that should be refactored.

Poster Presentations

(Tue, Jun 12, 17:00 – 17:45, Chair: Chanchal Roy / Stefan Löwe)

Toward Structured Location of Features

Hiroshi Kazato, Shinpei Hayashi, Satoshi Okada, Shunsuke Miyata, Takashi Hoshino, and Motoshi Saeki

NTT, Japan; Tokyo Institute of Technology, Japan

Abstract: This paper proposes structured location, a semiautomatic technique and its supporting tool both for locating features and exposing their structures in source code, using a combination of dynamic analysis, sequential pattern mining and formal concept analysis.

Extraction and Improvement of Conditionally Compiled Product Line Code

Bo Zhang

University of Kaiserslautern, Germany

Abstract: Conditional Compilation (CC) is one of the most widely used variation mechanisms in the development of software product lines (SPLs). However, a problem in SPL maintenance is that conditionally compiled code blocks are often overly scattered, nested, and tangled, which makes the code difficult to understand. Moreover, if variant code is evolved independently of the corresponding variability model, there is a risk that the two may become inconsistent. As a countermeasure, this paper proposes a maintenance process, consisting of variability extraction, error detection, and refactoring, to improve the quality of product line implementation.

Applying Bioinformatics in the Analysis of Software Variants

Vasil L. Tenev and Slawomir Duszynski

Fraunhofer IESE, Germany

Abstract: Analysis of software similarity is a lively research topic, particularly in the context of software maintenance and software reuse. There exist several approaches to detecting similar code inside one software system and across many systems. While working on similarity analysis of software variants, we observed many analogies between the approaches for analyzing evolution of software and of biological organisms. Hence, we applied bioinformatics concepts used in genome similarity analysis, such as alignments and phylogenetic trees, to software variants. We present the usefulness of these concepts by applying them to a group of related systems from the BSD Unix family.

Parallel Code Clone Detection Using MapReduce

Hitesh Sajnani, Joel Ossher, and Cristina Lopes

UC Irvine, USA

Abstract: Code clone detection is an established topic in software engineering research. Many detection algorithms have been proposed and refined but very few exploit the inherent parallelism present in the problem, making large scale code clone detection difficult. To alleviate this shortcoming, we present a new technique to efficiently perform clone detection using the popular MapReduce paradigm. Preliminary experimental results demonstrates speed-up and scale-up of the proposed approach.

Student Research Symposium

(Wed, Jun 13, 16:15 – 17:45, Chair: Massimiliano Di Penta / Denys Poshyvanyk)

Automatic Software Architecture Recovery: A Machine Learning Approach

Hitesh Sajnani

UC Irvine, USA

Abstract: Automatically recovering functional architecture of the software can facilitate the developer's understanding of how the system works. In legacy systems, original source code is often the only available source of information about the system and it is very time consuming to understand source code. Current architecture recovery techniques either require heavy human intervention or fail to recover quality components. To alleviate these shortcomings, we propose use of machine learning techniques which use structural, runtime behavioral, domain, textual and contextual (e.g. code authorship, line co-change) features. These techniques will allow us to experiment with a large number of features of the software artifacts without having to establish a priori our own insights about what is important and what is not important. We believe this is a promising approach that may finally start to produce usable solutions to this elusive problem.

Toward an Effective Automated Tracing Process

Anas Mahmoud

Mississippi State University, USA

Abstract: The research on automated tracing has noticeably advanced in the past few years. Various methodologies and tools have been proposed in the literature to provide automatic support for establishing and maintaining traceability information in software systems. This movement is motivated by the increasing attention traceability has been receiving as a de jure standard in software quality assurance. Following that effort, in this research proposal we describe several research directions related to enhancing the effectiveness of automated tracing tools and techniques. Our main research objective is to advance the state of the art in this filed. We present our suggested contributions through a set of incremental enhancements over the conventional automated tracing process, and briefly describe a set of strategies for assessing these contributions impact on the process.

Characterization of the Linux Configuration System

Ahmad Jbara

Hebrew University of Jerusalem, Israel

Abstract: Variability in software systems is often expressed using the C pre-processor (CPP). However, CPP has been identified as problematic. We argue that CPP is not as bad as its reputation suggests, and indeed many large systems use it effectively. We perform a deep analysis of the Linux configuration options, and find significant inconsistencies between the source code and the configuration control system. We found that the distribution of the source code config options is heavy-tailed, with some options having more than a thousand instances in the code. Such wide use seems to imply a massive coupling between different parts of the system. However, we argue that employing a purely syntactic analysis, as is commonly done in recent work, is insufficient. By involving semantic considerations, we find that in reality the coupling induced by the very frequent options is limited. We believe that deep characterization as well as semantic consideration are a good basis for future developing of different metrics for the CPP complexity.

Leveraging Clone Detection for Internet-Scale Source Code Search

Iman Keivanloo

Concordia University, Canada

Abstract: Different approaches to search for source code on the Internet exist. In this paper, we propose techniques that support a special type of Internet-scale code search using two novel syntactical and semantic clone search and detection methods. The syntactical search focuses on providing a scalable real-time engine, while the semantic clone detection is being used to enrich our knowledge base during the offline processing step.

Contact and Locations

Conference Venue

Nikolakloster
Room 403
Contact phone: **++49 / 1573 / 720 22 13**
Secretary phone: ++49 / 851 / 509 30 91

Campusmap reference: 1

Social Events

Guided Tour through Passau (Sunday, 17:30)

- Meeting point: portal of Passau's cathedral (“Domplatz”)

Meeting at the beer garden “Bayerischer Löwe” (Sunday, 19:00)

- Location: Dr.-Hans-Kapfinger-Straße 3 (close to central bus status)

Excursion to High Ropes Course Passau (Monday, 15:45)

- We leave together following the session on Monday afternoon at 15:45
- Departure by bus at the central bus station (“ZOB”) at 16:00
- Location: Wald-Erlebnis-Park Passau, Karlsbaderstraße 17

Banquet Cruise on the river Danube (Tuesday, 18:30)

- Boarding at the boarding site (“Schiffsanlegestelle”) 7 and 8 near the city hall at 18:30
- Departure time: 19:00

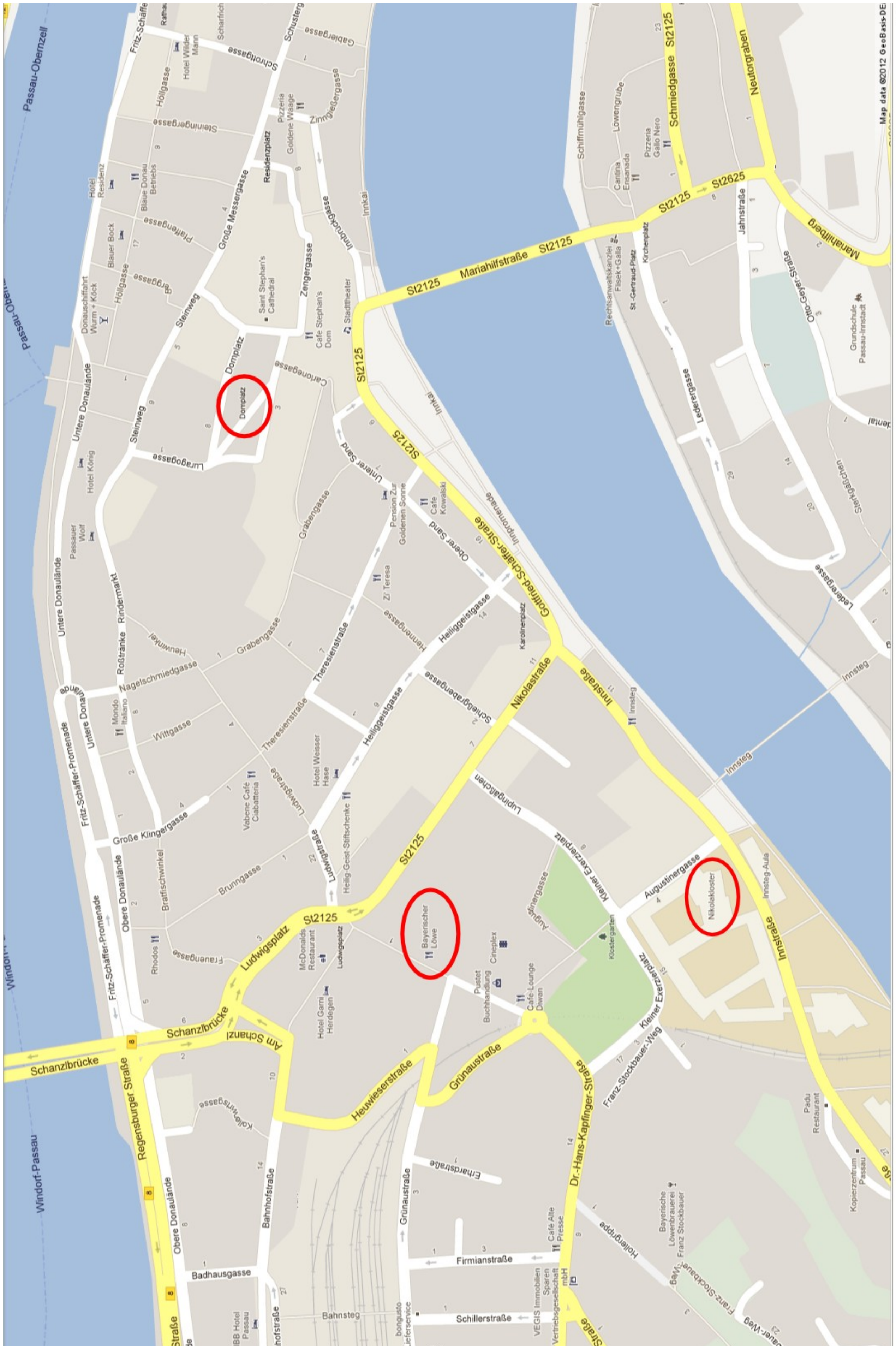
Campusmap reference: F

Hotels

	phone	**address**
Altstadthotel	++49 / 851 / 33 70	Bräugasse 23
Hotel Am Paulusbogen	++49 / 851 / 93 10 60	Rindermarkt 2
Hotel Garni Herdegen	++49 / 851 / 95 51 60	Bahnhofstraße 5
Hotel König	++49 / 851 / 38 50	Untere Donaulände 1
Hotel Passauer Wolf	++49 / 851 / 93 15 10	Untere Donaulände 4
Hotel Residenz	++49 / 851 / 989 020	Fritz-Schäffer-Promenade 6
Hotel Spitzberg	++49 / 851 / 95 54 80	Neuburger Str. 9
Hotel Weisser Hase	++49 / 851 / 922 10	Heiliggeistgasse 1
Hotel Wilder Mann	++49 / 851 / 350 71	Am Rathausplatz 1
Hotel Schloss Ort	++49 / 851 / 340 72 73	Ort 11
IBB Hotel Passau	++49 / 851 / 988 30 00	Bahnhofstraße 24
Rotel Inn	++49 / 851 / 951 60	Donauufer

Local Transportation

- Taxi: ++49 / 851 / 573 73


Domplatz
Bayerischer Löwe
Nikolakloster
Passau-Obernzell
Windorf-Passau
Schanzlbrücke
Regensburger Straße
Ludwigsplatz
St2125
Nikolastraße
Innstraße
Gottfried-Schäffer-Straße
Mariahilfstraße
Saint Stephan's Cathedral
Große Messergasse
Residenzplatz
Untere Donaulände
Obere Donaulände
Fritz-Schäffer-Promenade
Theresienstraße
Heiliggeistgasse
Ludwigstraße
Grabengasse
Steinweg
Innsteg
Klostergarten
Kleiner Exerzierplatz
Augustinergasse
Innsteg-Aula
Grünaustraße
Heuwieserstraße
Bahnhofstraße
Dr.-Hans-Kapfinger-Straße
Franz-Stockbauer-Weg
Schmiedgasse
Neutorgraben
Lederergasse
Map data ©2012 GeoBasis-DE