Report of the 4th International Symposium on Empirical Software Engineering and Measurement ESEM 2010

Emiliano Lutteri, Barbara Russo, Giancarlo Succi
Faculty of Computer Science, Free University of Bolzano, Piazza Università,1, Bolzano, Italy
{emiliano.lutteri, barbara.russo, giancarlo.succi}@unibz.it

Abstract

This report summarizes the research works, in particular the full and short papers, presented at the 4th International Symposium on Empirical Software Engineering and Measurement (ESEM 2010), held the 16th and 17th of September in Bolzano-Bozen, Italy. The program provided thirty full papers, twenty six short papers and three invited talks held by Bertrand Meyer, Steve Fraser and Carlo Ghezzi.

Keywords: ESEM, software engineering, empirical software engineering, software measurement

Introduction

This paper reports of the 4th International Symposium on Empirical Software Engineering and Measurement ESEM 2010 organized by the Free University of Bolzano, Italy, on September 16-17, 2010. The call for research papers attracted one hundred two full papers and forty-nine short papers and posters submissions from thirty-nine different countries and one hundred ninety eight authors. Eighteen percent of the submissions were from industry. The review was intensive with three hundred nineteen reviews and eleven paper discussions. Thirty papers were accepted as full papers (less than 30% acceptance rate) and twenty-six as short papers. The conference was co-located with ISERN (International Software Engineering Research Network), IDoESE (International Doctoral Symposium on Empirical Software Engineering), IASESE (International Advanced School on Empirical Software Engineering), and METRISEC (International Workshop on Security Measurements and Metrics). The conference (ESEM2010) was attended by about one hundred thirty participants.

The major mission of the conference is presented in the introductory welcome of the co-chairs, Nachi Nagappan and Maurizio Morisio: "The objective of the International Symposium on empirical Software Engineering and Measurement (ESEM) is to provide a forum where researchers and practitioners can report and discuss recent research results in the area of empirical software engineering and metric. The conference focuses on the processes, design and structure of empirical studies and on the results of specific studies." As such, the conference aims at promoting research in empirical software engineering, specifically in generalization and replication of experiments in software engineering, software reliability, and methods of development processes such as agile, distributed or open source.

Keynotes

One welcome keynote and two opening keynotes opened the conference works. The conference developed into three parallel tracks each of three sessions. Two tracks were for full and one for short papers. Bertrand Meyer challenged the audience at the welcome keynote with his presentation *Empirical Research: Questions from Software Engineering*. Meyer stated, "The premise of empirical software engineering is that methods and tools should undergo objective assessment. Many empirical studies, however, fail to give conclusive evidence on the questions of fundamental interest in software engineering. If empirical software engineering cannot answer to such questions, its benefit is debatable." He claimed to be in fact an outsider of the empirical software engineering research and, as such, in a good position to step back and look at this research under a different angle. He presented a number of open questions, pertaining to modern trends in software methods, and outlined ways to answer them through empirical research. Steven Fraser opened the first day of the conference with his keynote *Software Best Practices: Tales of Adoption and Agility through Iteration*. Fraser observed that modern software practices have been embracing a mix of enthusiastic passion and practical need. Agile practices are the example. Unlike the past, agile practices need to scale up to more mission-critical, distributed, and large software developments. Fraser presented a variety of best practices that challenged the audience. Carlo Ghezzi was the speaker of the second day opening keynote *The Disappearing Boundary Between Development-time and Run-time*: "Modern software systems are increasingly embedded in an open world that is constantly evolving, because of changes in the requirements, in the surrounding environment, and in the way people interact with them. The platform itself, on which software runs, may change over time, as we move towards cloud computing. These changes are difficult to be predicted and anticipated, and their occurrence is out of control of the application developers. Because of these changes, the applications themselves need to change. Increasingly, changes in the applications can't be handled by re-playing the development process off-line in the maintenance phase, but require the software to self-react by adapting its behavior dynamically, to continue to ensure the desired quality of service. The big challenge in front of us is how to achieve the necessary degrees of flexibility and dynamism required by software without compromising the necessary dependability."

The three keynotes have drawn a clear path of the future: empirical software engineering research needs to assist and drive practices that enable real-time adaptation of large, distributed and dynamic systems.

Parallel full paper Sessions

The full papers presentations are organized into two parallel tracks of three sessions per day. Each session regards a specific topic in empirical software engineering.

DOI: 10.1145/1943371.1943393 http://doi.acm.org/10.1145/1943371.1943393

Day 1 - Session 1A Generalization and Replications

This section concerns theories of generalization and replications of experiments and case studies in Software Engineering (SE). Authors debate the quality of the research and present methods to guarantee it in experiments as well as in their replications.

Cruzes and Dyba, in *Synthesizing Evidence in Software Engineering Research*, believe that the "Research synthesis is, a way of making sense of what a collection of studies is saying." The authors have analyzed papers on systematic reviews on SE published between 2005 and 2010, collected from various sources. They have found thirty-one different works on systematic review, half of which did not follow any synthesis method. "The results show that, despite of the focus on systematic reviews, there is limited attention to research synthesis in software engineering." The rest of the works use several methods of synthesis and different types of presentation of the synthesized findings. To assess the quality of software engineering experiments, Kitchenham et al. in *Can We Evaluate the Quality of Software Engineering Experiments?* analyze the usability of tools - a checklist - for evaluating the quality of software engineering experiments. In their approach, the authors compared individual and collaborative review on a sample of selected papers. The results indicate that "the inter-rater reliability was poor for individual assessments but much better for joint evaluations." Four reviewers working in pairs were more reliable than eight independent reviewers. Gómez et al. in *Replications Types in Experimental Discipline* discuss how replications are performed in other scientific disciplines like social science, economy or business. By analyzing papers on empirical SE from the most common databases, the authors have identified eighteen different replication methods, containing seventy-nine replication types that are not yet standardized. Based on this study, replications seem to fall into three groups:

1) Replications that vary little or not at all with respect to the reference experiment,

2) Replications that do vary but still follow the same method as the reference experiment,

3) Replications that use different methods to verify the reference.

Day 1 - Session 1B Defects, Faults, Issues

Practitioners and managers are always looking for ways to reduce costs of maintenance. Detecting, localizing and fixing bugs (or issues) are typical activities of software maintenance that aim at estimating, preventing, and limiting those costs. In this session, authors debate this issue at various degrees.

Shihab et al. in *Understanding the Impact of Code and Process Metrics on Post-release Defects: A Case Study on the Eclipse Project*, conducted a case study on the Eclipse Project. They propose to use both code and process metrics to predict post-release defects. They found that that only four of the thirty four code and process metrics they considered show a significant likelihood of finding a post-release defect. In addition, the logistic model they use achieved comparable performance over more complex Principal Component Analysis(CPA)-based models. Kim and Baik in *An Effective Fault Aware Test Case Prioritization by Incorporating a Fault Localization Technique* claim that techniques of prioritization of test cases, defined by ordering test cases according to some coverage criteria, increase fault detection. In *An Effective Fault Aware Test Case Prioritization by Incorporating a Fault Localization Technique,* the authors introduce a new technique of test case prioritization that considers both coverage and historical fault information and use an automated debugging tool to isolate the location of faults. This new technique is called Fault Aware Test Case Prioritization (FATCP). The authors have performed an experiment to evaluate the performance of this new technique: "[..]FATCP, performs better than the existing low level coverage-based prioritization techniques in terms of rate of the fault detection." To simplify the system maintenance activities, Murgia et al. have analyzed the commit messages of Concurrent Version Systems (CVS). They identify and categorize statements bug fixing reports. In *A machine learning approach for text categorization of fixing-issue commits on CVS* they analyze commits in the CVSs of Eclipse and Netbeans projects. With their categorization, they were able to identify the characteristics of issue reports related to bug fixing activities. They compare the performance of various machine learning classifiers finding that these techniques can be successfully adopted for text mining and classification.

Day 1 - Session 2A Maintainability and dependability

This session is dedicated to maintainability of software product. In particular, the authors propose methods and activities that help maintaining software products. Three major contributions have been presented:

1) Practitioners' opinion on good practices of testing to identify improvements

2) Analysis of code artifacts to identify developers behavior, and

3) Developers' opinion of good practices of development.

Kasurinen et al. in *Test Case Selection and Prioritization: Risk-Based or Design-Based?* study how and why real-world software companies determine their approach to test case selection and prioritization. Findings indicate that the basic approaches to test case selection are:

1) The risk-based selection, where the aim is to focus testing on those parts that are too expensive to fix after launch;

2) The design-based selection, where the focus is on ensuring that software is capable of completing the core operations it was designed to do.

The authors have found that the risk based approach is selected in those organizations where testing resources are limited and where is possible to change the product design during the process. When projects need more testing resources and software design is made with plan-driven methods, testing focuses on test case coverage and, therefore, the design-based approach is preferred. Grechanik et al. in *An Empirical Investigation into a Large-Scale Java Open Source Code Repository* describe an infrastructure to perform empirical analysis of source code artifacts. The authors

DOI: 10.1145/1943371.1943393 http://doi.acm.org/10.1145/1943371.1943393

posed thirty-two research questions to understand how programmers write their source code. To answer these questions, they analyzed two thousand eighty Java applications randomly chosen from the public repository Sourceforge. Analyzing their results they found that

1) The majority of the methods has one or zero parameters or do not return any values,

2) Few methods are overridden,

3) Most inheritance hierarchies have the depth one; close to 50% of the classes are not explicitly inherited from any classes, and

4) The number of methods in a class is strongly correlated with the number of fields in the same class.

These findings may have severe implications on maintenance activities. The paper *A Survey of Scientific Software Development* presents results form a survey on development of software for scientific research purposes (scientific software) administrated to the community of scientific software developers. Nguyen-Hoan et al. identify where improvements in scientific software practice can be made. Six areas were covered in the survey: programming languages, tools, development teams, documentation, testing and verification, and non-functional requirements. The survey shows that development practices in scientific software have improved in recent years. In particular, software has improved in reliability and functionality. There is still space for improvement, though. "Although version control and IDEs are used by around 50% of developers, further improvements in the uptake of these and other tools are possible." Documentation in scientific software can be improved and testing and verification activities - like peer review and integration testing - could be more widely used.

Day 1 - Session 2B Large-scale analysis

In this section, authors present ways to streamline and speed up the maintenance of large size software. Three strategies have been discussed: evolution of design patterns, code smell for re-factoring, and developers' contribution on software vulnerability.

Schanz and Izurieta in *Object Oriented Design Pattern Decay: A Taxonomy*, introduce grime as measure of decay of the design patterns. Grime measures the deviation of a pattern from its original conception. This deviation indicates degeneration of the original pattern. Authors define a modular grime taxonomy that uses three criteria to characterize grime: strength, scope and direction. Then, they applied their taxonomy of design pattern grime to some large projects. They report that "Grime originating from external classes is harder to remove because of the heightened responsibility of the pattern. Grime originating inside the pattern causes the testability of the pattern to increase because the dependencies of said pattern are higher." Schumacher et al. in *Building Empirical Support for Automated Code Smell Detection* discuss of re-factoring techniques in agile methods – e.g. code smells detection - that helps developers identifying design flaws in their software. They have compared human ability of detecting the so called *god classes* with the performance of automated classifiers. ".. the results for the automatic detection of god classes show how Marinescu's detection strategies [1] recall successfully for the two projects. A computer assisted strategy in which automatically detected god classes undergo a second human review can reduce the required effort. The results of the study increase the overall confidence in the results of automatic code smell detection." Meneely and Williams in *Strengthening the Empirical Analysis of the Relationship between Linus' Law and Software Security* studied the effect of the distributed development in open source – in particular, the structure of open source developer collaboration - on security vulnerabilities. In this paper, the authors use network analysis to quantify how developers collaborate on projects, a binary classification to identify if the file is vulnerable or neutral, and the analysis of version control logs to characterize development activities. They perform an empirical analysis on three software projects: the Linux kernel, the PHP programming language, and the Wireshark network protocol analyzer finding the following observations:

1) source code files changed by multiple, otherwise-separated clusters of developers are more likely to be vulnerable than changed by a single cluster;

2) files are likely to be vulnerable when changed by many developers who have made many changes to other files;

Day 1- Session 3A Agile Methods

Agile methods claim to have some distinguishing advantages over conventional software development methods: the goal of this session is to provide a better understanding of how agile processes affect software quality, and whether and how any advantage is perceived.

Li, et. al. in *Transition from a Plan-Driven Process to Scrum – A Longitudinal Case Study on Software Quality* report on a longitudinal study in which they have followed a project for more than three years. Authors analyse the software quality assurance process and the software defects detected during two phases: during seventeen months in which development followed a plan-driven approach and a second twenty month phase of Scrum. The results of the study does not show a significant reduction of defect densities or changes of defect characteristics after the use of Scrum. However, the iterative development in Scrum has contributed to manage software defects better and to fix them earlier. resulting more efficient in controlling software quality and release plans. Zazworka, et al. in *Are Developers Complying with the Process: An XP Study* investigate process conformance during developers' training after the introduction of new software processes and practices in organizational and academic environments. The authors have developed an approach to detect process conformance violations - i.e. situations not conforming to the process' definition. Their approach is based on non-intrusive data collection form a versioning control system. They refer to non-intrusive data collection as a method that keeps minimal costs and overhead of data collection activities used to investigate process conformance. The authors performed an experiment with students. They selected three different practices related to eXtreme Programming : Test-Driven Development, Collective Code Ownership, and Continuous Refactoring. Findings show that it is possible to determine conformance violations using minimally intrusive methods. Moreover, issues of non-conformance are prevented by understanding better the applicability and effectiveness of the agile practices. In order to identify potential

DOI: 10.1145/1943371.1943393 http://doi.acm.org/10.1145/1943371.1943393

threats to productivity in large agile development projects, Hannay and Benestad in *Perceived Productivity Threats in Large Agile Development Projects* have conducted repertory grid interviews with thirteen members of an agile project on their perceptions of threats to productivity. The project was large as one hundred seventy six people were involved. "The repertory grid sessions produced one hundred issues, analyzed into ten main problem areas:

1) Restraints on collaboration due to contracts, ownership, and culture,

2) To architectural and technical qualities are given low priority,

3) Conflicts between organizational control and flexibility,

4) Volatile and late requirements from external parties,

5) Lack of a shared vision for the end product,

6) Limited dissemination of functional knowledge,

7) Excessive dependencies within the system,

8) Overloading of key personnel,

9) Difficulties in maintaining well functioning technical environments,

10) Difficulties in coordinating test and deployment with external parties."

Day 1- Session 3B Requirements analysis and engineering

This session concerns methods, techniques and tools that aim at improving, at simplifying and making the collection and the analysis of software requirements more efficient. This helps increasing productivity and shortens development time.

Qattous et al. in *An Empirical Study of Specification by Example in a Software Engineering Tool,* present an empirical study on the use a technique of requirement elicitation called Specification by Example (SBE) as a user-computer interactive technique for constraint specification. They compare constraint definition by example with a traditional wizard specification method. The technique has been inspired by Programming By Example (PBE), a practice in development that uses examples of data and values to create the whole program. The objective of PBE is to make programming an easier task also accessible for non-programmers. Findings prove that SBE increases the correctness of the code by defining a high percentage of constraints. In general, participants required less time to define constraints using SBE than the wizard. The results also demonstrate that using SBE for constraint definition reduces the complexity of the constraint definition task. This indicates that SBE can add value to the meta-modeling process and, consequently, to the specification process of software engineering tools in general. Ricca et al in *On the Effectiveness of Screen Mockups in Requirements Engineering: Results from an Internal Replication* illustrate a replication of a controlled experiment to assess the effectiveness of including *screen mockups* when adopting Use Cases. For the authors, the technique is a simple way to capture and define requirements from the end user point of view. The results of the original experiment showed that subjects found useful the inclusion of screen mockups when adopting Use Cases. Findings showed a clear improvement of the understandability of functional requirements. The experiment replication was conducted at the University of Genoa (Italy) with fifty-one third-year students of the Bachelor program in Computer Science. The analysis of the replication has confirmed the results of the original experiment and .indicated "a clear improvement in the comprehension of software requirements when screen mockups are present with no significant impact on effort." Falessi et al. in *A Comprehensive Characterization of NLP Techniques for Identifying Equivalent Requirements,* characterize a set of Natural Language Processing (NLP) techniques to detect equivalent requirements. The study has been run in an Italian company working for the defense department. Finding show that simple NLP techniques are more precise than the complex ones.

Day 2- Session 1A Management, business and research in industry

A major role of empirical software engineering research is to provide decision makers with evidence about best technologies that they could adopt in software development.

Jedlitschka in *Evaluating a Model of Software Managers' Information Needs – An Experiment* proposes the evaluation of a model that improves the delivery of relevant information in Empirical Software Engineering (ESE) research to software managers. His approach suggests to enriching reports of ESE research with information relevant for managers. The effectiveness of this model was evaluated during an experiment with twenty-two software managers from industry who have read two versions of the same report: one of these versions contains information as required by the model. Analyzing the results, Jedlitschka concludes that the model of information needs provides a means for supporting the identification of alternative viable solutions. Alternative solutions provide managers with a decision making tool. In particular, he uses Business Value Analysis (BVA) to quantify solutions in terms of value and cost risk. This approach is an alternative to the well- known GQM+ Strategies method that aims at defining business goals and strategies of a software company. The GQM+ strategy method is deeply discussed in *Utilizing GQM+ Strategies for Business Value Analysis. An Approach for Evaluating Business Goals* of Mandic et al. The authors describe how to perform business value analysis (BVA) using the GQM+ Strategies approach. The integration of these two methods couples value goals, like cost-benefit and risk analysis, with measurable business goals. To illustrate the feasibility of their method and its application, the authors present an example inspired by a real world case. Robinson and Francis in *Improving Industrial Adoption of Software Engineering Research: A Comparison of Open and Closed Source Software* present a large metrics-based study comparing the most common open source programs to a set of industrial applications. They have calculated several Source metrics and compared them between twenty-four open source and twenty-one industrial programs. The results show that Open source software (OSS) is similar to Industrial Software (IS). The finding could be useful to guide managers in selecting software with the largest relevance for industrial adoption.

DOI: 10.1145/1943371.1943393 http://doi.acm.org/10.1145/1943371.1943393

Day 2- Session 1B Agile, collaborative and distributed development

Distributed developing teams, stakeholders from different national and organizational cultures, geographic locations, and time zones characterize the modern ways of software development. These conditions have consequences on communication, coordination, and control within software projects. Since software development depends on human interactions, addressing these challenges is critical for successful distributed development methods. This session debate this issue.

Salleh et al. in *The Effects of Neuroticism on Pair programming: An Empirical Study in the Higher Education Context* present the results of an empirical study that investigates the effects of the personality trait of neuroticism on the academic performance of two hundred seventy students, who practiced pair programming during one academic semester. The study is focused on the neuroticism factor - one personality factor of the Five-Factor Personality Model (FFM)). In literature, this factor has a prominent role in learning in educational context. The aim of the study is to investigate whether neuroticism plays a role in the performance of students who works as pair programmer. Findings show that the success of distributed development depends on achieving and maintaining trust within teams with different cultures. Trust in distributed development is the theme of another study of this session. Jalali et al., in *Trust Dynamics in Global Software Engineering*, explore the dynamics of trust in software organizations using distributed software development. The authors present a model of trust dynamics, built up analyzing the results of semi-structured interviews conducted in six different distributed software development organizations. In particular, the work presents best managerial practices for building or maintaining trust in distributed development .With the paper *Investigating the Use of Tags in Collaborative Development Environments: A Replicated Study,* Calefato et al., present an independent replication of an empirical study that investigates how tags are used in large software projects. They analyzed two different projects, Jazz Foundation and WebLion, developed ,respectively, with two Collaborative Development Environments (CDE): Jazz and Trac. The authors stated, "The findings from our replicated study extend the initial contribution of the original study by

(1) showing evidence of differences in tag usage between the two collaborative development environments, and

(2) providing a clear understanding that tags used in such environments significantly differs from those used in traditional collaborative tagging systems."

Day 2 - Session 2A Measurement and Estimation

In software industry, Functional Size Measurement (FSM) is the most popular way of measuring software requirements for cost estimation purposes, but these measures do not consider the amount and complexity of the required elaboration.

Lavazza and Robiolo in *Introducing the Evaluation of Complexity in Functional Size Measurement: a UML-based Approach* show that a measurement-oriented UML modeling can support the measurement of both functional size and functional complexity. The authors show, by means of a case study, that it is possible to derive simply from UML models, different types of functional size measures, as well as complexity measures. Moreover, they claim that "the development effort appears very well correlated to the size of the requirements expressed in unadjusted function points and the complexity of the required system functions, expressed as the mean number of paths per transaction." Another paper presented in the same session, deals with Code Size estimation using COSMIC Function Points (CFP) measures. Lind and Heldal in *Categorization of Real-Time Software Components for Code Size Estimation*, aim at identifying factors affecting the linear relationship between CFP and Code Size, in order to categorize new requirements to be measured and selecting the proper linear relationship to convert CFP into Code Size of real time software component. They have conducted two experiments from the automotive industry domain, studying 46 software components of three functionality types: each experiment showed strong correlation, but a different linear relationship. The authors have identified the four factors affecting the linear relationship: functionality type, quality constraints, development method and tools, and information regarding hardware interfaces missing in the requirement specification. "COSMIC, the FSM method used, can produce accurate Code Size Estimates provided that sub-sets of cohesive and uniform requirements can be identified." Software organizations have increased their interest on software process improvement (SPI). In high maturity levels, SPI involves implementing statistical process control (SPC), which requires measures and data that are suitable for this context. In the paper *Evaluating the Suitability of a Measurement Repository for Statistical Process Control*, Barcellos et al., describe an instrument for evaluating the suitability of measurement repositories that support software organizations in implementing SPC. The tool, Instrument for Evaluating the Suitability of a Measurement Repository (IESMR), developed by the authors, currently composed mainly by spreadsheets, was used to evaluate the measurement repositories of three organizations. The results show that the evaluation and adjustment of the measurement repository before performing SPC, help applying statistical techniques.

Day 2 - Session 2B Human and User Studies

A modern concept in software development is pluralism—that is, the design of software artifacts that can "resist any single, totalizing, or universal point of view" [2]. Pluralism requires a specific attention to human factors and issues, so that it could improve usability of the software products. This section sheds some light on the use of software, analyzing use from different points of view.

Burnett, et al. in *Gender Differences and Programming Environments: Across Programming Populations* investigate if there are gender differences in using programming tools and how such phenomena are widespread in industrial programming environments. Considering three different programming populations involving almost three thousand people using three different development platforms in industry, the authors analyzed five industry studies from a gender perspective. In particular, they investigate gender differences in features' usage and exploration and show significant differences across populations and platforms. The usability of software as teaching tool, is studied by Dahotre et

DOI: 10.1145/1943371.1943393 http://doi.acm.org/10.1145/1943371.1943393

al. In *A Qualitative Study of Animation Programming in the Wild* They analyze Scratch, a development and animation tool that was introduced in the after-school programming clubs to teach programming. Scratch's purpose is to achieve three goals in teaching skills: technical, collaborative and for reuse. The authors have performed an empirical study of animations and users' comments - taken from the online animation repository - in order to assess the strengths and weaknesses of Scratch. They found that Scratch represents a progress, but they have also identified several opportunities for its improvement. Finally, Aquino et al. analyzed the usability of the user interfaces in terms of satisfaction, effectiveness and efficiency. Their paper *Usability Evaluation of Multi-Device/Platform User Interfaces Generated by Model-Driven Engineering* introduces to the use of Model Driven Engineering (MDE) to generate multi-platform graphical user interfaces (e.g., desktop, web, etc.). The authors investigated the usability of the interfaces generated on multiple display devices with several screen sizes. From the results of questionnaires and interviews of the thirty one participants, the authors state: "Efficiency was significantly better in large screens than in small ones as well as in the desktop platform rather than in the web platform [...] The experiment also suggests that satisfaction tends to be better in standard size screens than in small ones."

Short paper Sessions

Day 1 short Sessions

The short papers sessions are organized into two tracks of six sessions per day. Each session regards a specific topics in empirical software engineering. The first short papers session was dedicated to research methods and generalizations. As for full papers, this session concerns meta-studies across replications or large number of samples. These studies provide recommendations and trends in the empirical software engineering research.

One of the typical methods in generalization studies is literature review. Q.B. da Silva et al. have analyzed fifty-three literature reviews of software engineering articles. The authors found that there is a need of standardization of terms and methods to perform literature reviews in software engineering. A second approach is to use simulated data and empirical environments. Simulations allow replications of the study in identical environments, but they need a clear definition of the experimental settings. For this reason, Pereira and Travassos introduced activities workflows for software engineering experiments in simulated environments (called *in silico* experiments). Simulations might be one approach that solves the scarcity of software engineering data. Another one can be crowd outsourcing as Stolee and Elbaum proposed as a mechanism to recruit subjects for empirical studies. Finally, the session includes two focused applications of generalizations theories. One concerns the relevance of time measures in defect predictions (Stephen MacDonell and Martin Shepperd) and the second introduced stakeholder's perceptions as a measure of success of software processes (McLeod and MacDonnel).

The second short paper session focuses on topics related with agile, collaborative and distributed development. As in the full paper session, issues concern collaboration, distributed development, product quality, and project success. Collaboration between customers and developers is the topic discussed by Racheva and Daneva whereas simplification in the collaboration among researchers groups using Virtual Machines, is proposed by Augustine and Robinson in their work. Collaboration and problems within developers' teams is analyzed by Ricca and Marchetto. The authors define a measure of spread of knowledge within a team of developers to determine the risk to have heroes within the team that might increase the risk of failure of the project. Also França et al. discuss of the success / failure of projects that use agile management methods like SCRUM. Finally , Lavazza et al. propose how to predict trustworthiness for open source software (OSS) by means of static code measures using Elementary Code Assessment (ECA). ECA is a pragmatic approach for providing a quick estimate of the internal quality of software products with little effort. The last track of the first day debates qualitative, social and alternative perspectives. Empirical Software engineering has to deal with different sort of people, developers, customers, and software producers. This is a large interacting community. Kakarla and Namin focus on technology. In their exploratory study, they identify environmental factors, which influence the runtime behavior of paralleled applications, specifically multicore platforms. Social aspects are also critical in software projects' management - as for example, balancing project staffing. To support managers in this activity, Rahman et al. propose a comparison under different optimization algorithms of staffing optimization in the activities of feature development and bugs fixing. The motivation of software practitioners is a key factor in system quality. To understand, for example, the motivation for people to work as software engineer, Sach et al. present an analysis of data collected from twenty three practitioners at a workshop on motivation, while Daneva and Ahituv discuss their results of the Focus Group Study on Inter-organizational ERP Requirements Engineering Practices that evaluated twelve practices for engineering coordination requirements, presented in a previous work of the authors. Finally, Sarcià in his paper reports that it is possible, with specific modifications, to adapt the GQM+ Strategies approach to those domains that have the same nature as software development (i.e., human-intensive domains).

Day 2 Short Sessions

In the second day, the first session is dedicated to issues of software products: defects, faults, and failures. As for the full papers track, this session collect papers that give a contribution to prevent defects, improve and simplify bugs fixing in order to improve quality and maintenance of software products and reduce risks. Basili et al. in *Obtaining Valid Safety Data for Software Safety Measurement and Process Improvement*, examine one hundred fifty four reports on risk, created during the preliminary design phase of three major flight hardware systems, within the NASA Constellation program. Their work aims at assisting safety assurance personnel in identifying system components of top software safety risk and identifying potential risks due to incorrect application of the safety process. The defects detection and their analysis is theme of other works in this session: Marín et al. present a study on the usefulness of functional size measures in detecting defects of software development using Model Driven Design, whereas Albayrak and Davenport in analyze the effect of two maintainability defects on software inspection processes.

DOI: 10.1145/1943371.1943393 http://doi.acm.org/10.1145/1943371.1943393

Ramler et al. explore and describe quantitatively the variance of defects created by different developers working at same project. Impact analysis is the topic of the last two papers: Ahsan and Wotawa use Machine Learning classification to understand the use of information of solved Software Change Requests (SCR). The classification predicts which files have to be changed whenever a new SCR is received. Torchiano and Ricca propose a prototype tool used to analyze information gathered from software repositories of source code comments or version control logs. The findings aim at analyzing the impact on development activities in case of bug fixing or feature enhancement. The last short papers session present works on code design and modeling. The goal of this session is to provide research contributions improvement of software quality at the earlier stages of development. One of the major them is design patterns. Gravino et al. evaluate the effort and the efficiency in performing maintenance operations, when design pattern instances are properly documented and provided to the maintainer. Brennan et al. focus on how design patterns impact on the performance cost in terms of execution time, CPU utilization and memory usage. Modeling was a second research area in this session. Scanniello et al. compared two communication media in Use Case modeling. The authors compare the traditional face-to-face with advanced chat communication by modeling functional requirements using use cases. Ricca et al. study software requirements definition using Use Cases with screen mock-ups. Finally, Fernandez et al. propose their Web Usability Evaluation Method (WUEP), which can be integrated into model-driven Web development processes. This paper presents the first steps in the empirical validation of WUEP through a controlled experiment.

Awards

At the closing of the conference, three papers were awarded as best research works among the Full Papers presented:

- Jalali, Gencel and Smite. *Trust Dynamics in Global Software Engineering*
- Li, Moe and Dybå. *Transition from a Plan-Driven Process to Scrum - A Longitudinal Case Study on Software Quality*
- Zazworka, et al. *Are Developers Complying with the Process: An XP Study*

in addition, among the Short Paper were awarded :

- MacDonell and Shepperd. Data Accumulation and Software Effort Prediction

Papers have been awarded according to four factors: originality, extensive related works investigation, rigor of the empirical analysis, large size or multiple data sets. In term of originality and new trends in empirical software engineering, the three papers contributed respectively to:

1) Global Software engineering and the role of trust in distributed development,

2) Product quality in agile software development in industry,

3) Process conformance in eXtreme software development process.

It is worth noticing that two out of the three best full papers concern agile practices in software development.

Reference

[1] Marinescu, R. (2004). Detection Strategies: Metrics-Based Rules for Detecting Design Flaws. *In Proceedings of the 20th IEEE international Conference on Software Maintenance* (September 11 - 14, 2004). ICSM. IEEE Computer Society, Washington, DC, 350-359

[2] Bardzell, S.(2010) Feminist HCI: Taking stock and outlining an agenda for design, In *Proc. ACM Conference on Human Factors in Computing Systems,* ACM (2010), 1301–1310

DOI: 10.1145/1943371.1943393 http://doi.acm.org/10.1145/1943371.1943393