Report from Dagstuhl Seminar 13362

Cloud-based Software Crowdsourcing

Edited by

Michael N. Huhns[1], Wei Li[2], and Wei-Tek Tsai[3]

1 **University of South Carolina, US,** huhns@sc.edu
2 **Beihang University – Beijing, CN,** liwei@nlsde.buaa.edu.cn
3 **ASU – Tempe, US,** wtsai@asu.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 13362 "Cloud-based Software Crowdsourcing".

In addition to providing enormous resources and utility-based computing, clouds also enable a new software development methodology by crowdsourcing, where participants either collaborate or compete with each other to develop software. Seminar topics included crowd platforms, modeling, social issues, development processes, and verification.

Seminar 01.–04. September, 2013 – www.dagstuhl.de/13362
1998 ACM Subject Classification K.6.3 Software Management
Keywords and phrases Crowdsourcing, Software Development, Cloud Computing
Digital Object Identifier 10.4230/DagRep.3.9.34
Edited in cooperation with Wenjun Wu

1 Executive Summary

Michael N. Huhns
Wei-Tek Tsai
Wenjun Wu

Crowdsourcing software development or software crowdsourcing is an emerging software engineering approach. Software development has been outsourced for a long time, but the use of Internet with a cloud to outsource software development to the crowd is new. Most if not all software development tasks can be crowdsourced including requirements, design, coding, testing, evolution, and documentation. Software crowdsourcing practices blur the distinction between end users and developers, and allow the co-creation principle, i.e., a regular end-user becomes a co-designer, co-developer, and co-maintainer. This is a paradigm shift from conventional industrial software development to a crowdsourcing-based peer-production software development. This seminar focused on the notion of cloud-based software crowdsourcing, with the following goals:

1. to establish a theoretical framework for applying software crowdsourcing, and identify the important design patterns and highly interactive and iterative processes in a cloud-based infrastructure.
2. to propose and design a reference architecture for software crowdsourcing
3. to develop and finalize the research roadmap for software crowdsourcing for the next five years

Cloud-based Software Crowdsourcing, *Dagstuhl Reports*, Vol. 3, Issue 9, pp. 34–58
Editors: Michael N. Huhns, Wei Li, and Wei-Tek Tsai

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The grand research challenge in cloud-based software crowdsourcing is how to embrace elements from the two aspects: cloud infrastructure and software crowsourcing. Metaphorically, it can be regarded as synergy between two clouds – machine cloud and human cloud, towards the ultimate goal of developing high-quality and low cost software products. This seminar intended to bring together scientists from both fields to tackle the major research problems in this emerging research area.

More than twenty researchers, who work on different domains such as crowsourcing, human-computer interaction, cloud computing, service oriented computing, software engineering and business management attended the seminar. In addition to regular 5-minute talks from every participant in the seminar, the organizer arranged a keynote speech delivered by Prof Schahram Dustdar, which summarizes large-scale collective problems solving research enabling software crowsourcing. The topics covered by their presentations can be roughly categorized into three groups: software crowdsourcing process and models, crowdsourcing cloud infrastructure and human crowd management. To promote in-depth discussion among these topics, we also divided people into five discussion groups including:

Crowd Source Software Engineering Design-Group: This group identified the three main areas in the design of software crowdsourcing: processes, models, and techniques. It highlighted the importance of standardized generic models of software crowdsourcing study, and explored multiple crowdsurcing techniques, especially virtual team formation and quality assessment.

Worker-centric design for software crowdsourcing: This group focused on the crowd management in software crowdsourcing and aimed to answering the question about how to make a sustainable software crowdsourcing industry. Discussion in the group covered the major issues such as careers and reputation development of workers, trust among workers and "employers" (task solicitors) on crowdsourcing markets, virtual team selection and team building.

Cloud-based Software Crowdsourcig Architecture: This group discussed the possible common architectures of crowd-sourcing applications and explored two complementary architectural approaches.

Experimentation Design for Software Crowdsourcing: The central topic of this group is about how to design a valid and reproducible experiment for software crowdsourcing research. The group had extensive discussion on software crowdsourcing experiment approaches and the major crowdsourcing infrastructures.

Infrastructure and Platform: This group reviewed the motivations to construct the crowdsourcing platform, analyzed architecture design issues, and proposed a educational platform for software crowdsourcing.

During the session of our seminar, Dagstuhl also set up a parallel seminar named "Crowdsourcing: From Theory to Practice and Long-Term Perspectives", which mostly focused on general crowdsourcing research and service platforms. Software crowdsourcing can be regarded as one of the most complex crowdsourcing activities that often need intense dedication from workers with high-level skills of software engineering. Thus, there are some interesting overlapping areas such as worker incentive and quality assurance, between our seminar and the parallel seminar. To foster collaboration among the two groups, we hold a joint discussion session for introducing and sharing findings from each group, followed by an evening session with two presentations from the general crowdsourcing group.

We believe this seminar is a good start for software crowdsourcing research. Finding and consensus generated from the seminar have been formalized in the wiki page of software crowdsourcing (http://en.wikipedia.org/wiki/Crowdsourcing_software_development) to give a clear definition and initial reference architecture of cloud-based crowdsourcing software development. More efforts will be put into the growth of the research community and production of joint research publications.

2 Table of Contents

3 Overview of Talks

3.1 Large-Scale Performance Testing by Cloud and Crowd

Xiaoying Bai (Tsinghua University – Beijing, CN)

Joint work of Bai, Xiaoying; Tsai, Wei-Tek; Wu, Wenjun

Software scale and complexity increase tremendously in recent years. Performance testing of Internet scale software systems is usually expensive and difficult. The challenges include: (1) to simulate diversified usage scenarios; (2) to generate various workload distributions; and (3) to measure and evaluate performance from different aspects. Cloud computing and Crowdsourcing are two emerging techniques in recent years. They promote new testing architectures that are promising to address the challenges. Cloud-based testing, such as TaaS (Testing-as-a- Service), aims to support on-demand testing resources and services in/on/over clouds for testers at any time and all time. Testing by crowdsourcing, such as uTest and Mob4hire, follows the Web 2.0 principle of harnessing collective intelligence and various testing tasks can be crowdsourced including test case design, script development, script debugging, test execution, and test result evaluation.

This paper first investigates the framework to facilitate large-scale performance testing by integrating cloud infrastructure and crowd wisdom. A Cloud-based testing platform, Vee@Cloud was built to provide a cross- cloud virtual test lab to support on-demand test scripts provisioning and resource allocation, scalable workload simulation, and continuous performance monitoring. Following the crowdsourcing approach, participants can join the platform for different purposes, such as biding testing tasks, contributing test cases and scripts, renting test resources, and carrying on test executions. The process of establishing test collaboration can be formulated as a multi-criteria decision making problem, using qualitative evaluation models for different factors like task allocation and scheduling, quality control, and cost control.

3.2 Finding Experts

Xavier Blanc (Univ. Bordeaux, LaBRI – Talence, FR)

Heavy usage of third-party libraries is almost mandatory in modern software systems. The knowledge of these libraries is generally scattered across the development team. When a development or a maintenance task involving specific libraries arises, finding the relevant experts would simplify its completion. However there is no automatic approach to identify these experts. In this talk we propose LIBTIC, a search engine of library experts automatically populated by mining software repositories. We show that LIBTIC finds relevant experts of common Java libraries among the GitHub developers. We also illustrate its usefulness through a case study on the Apache HBase project where several maintenance and development use-cases are carried out.

3.3 Crowdsourcing Cloud Infrastructure using Social Networks

Kyle Chard (University of Chicago, US))

The increasing pervasiveness of online social networks is not only changing the way that people communicate and interact but it is also allowing us to represent, document and explore inter-personal relationships digitally. Social networking platforms have gone beyond a platform for communication and are now a viable platform in their own right on which to implement unique socially oriented services with access to an increasingly complex social graph modeling every aspect of an individual life. Building upon this social fabric services can leverage real world relationships, inferring a level of trust between users, exploiting intrinsic social motivations, and developing socially aware algorithms. This talk describes the experiences and lessons learned developing a Social Cloud, a platform that enables the construction of a crowdsourced cloud infrastructure to facilitate resource and capability sharing within a social network. Social Clouds are motivated by the need of individuals or groups for specific resources or capabilities that can be made available by connected peers. Such resources are not necessarily only computational or software resources, but can be any electronically consumable service, including human skills and capabilities. Social Clouds leverage lessons learned through volunteer computing and crowdsourcing such as the willingness of individuals to make their resources available and offer their expertise altruistically for good causes. This talk explores aspects such as inter-personal trust, platform implementation, social incentives and use cases, by leveraging methodologies from computer science, economics and sociology. It looks specifically at these aspects in the context of social network based crowdsourcing and attempts to generalize the approaches used in Social Clouds to other forms of software crowdsourcing.

3.4 Cloud based Crowdsourcing Software Development – Keynote

Schahram Dustdar (TU Wien, AT)

In this talk I begin with analyzing the historical roots of large-scale collective problems solving research in Computer Science. this is followed by a detailed analysis of the mechanisms, algorithms, models, and software architectures and deployment models of software solutions helping to enable crowdsourcing software development on a large scale. This is discussed in relationship with an analysis of currently missing aspects of solutions aiming at supporting cloud based crowdsourcing for software development.

3.5 Hyperscale Development of Software

Michael N. Huhns (University of South Carolina, USA)

The benefits of the open-source (Bazaar? approach to software development have not been fully realized, because the number of software developers is still relatively small and orders of magnitude smaller than the number of users. Developers typically are experts in computing, whereas users typically have domain expertise: this produces a disparity in viewpoints, causing a mismatch between the developed software and its desired use. Moreover, the proposition, given enough eyeballs, all bugs are shallow,would take on much greater significance, if a larger fraction of the users could also be developers. A solution to this problem has so far been impractical: end-users often do not have sufficient expertise to contribute software, nor the time to learn how to do so, and there was no way to meld it with existing software until it was proven correct or from a trusted developer.

In this paper I describe an approach for broadening dramatically the number of people who contribute to the development of software. If successful, the approach will result in a more effective software-development process, greatly improved software reliability, and increased end-user satisfaction. The approach is ambitious – in that it affects all stages of software development and several levels of the software execution process and transformative – in that the software will be developed and executed in new ways.

The approach makes use of take advantage of-all contributed code, components, and designs until they have proven to be of no value. The talk will describe how agent-based wrappers can manage the necessary collaboration and competition, allowing the contributions to be used alongside their existing counterparts until their behavior and features can be assessed. The solution exploits concepts from N-version programming, multicore processors, model- driven architectures, test-driven development, autonomic computing, negotiation in multiagent systems, group decision-making, and consensus.

The two main threads of the proposed research are broader participation in developing software and multiagent systems for the use and execution of the software. For each we are developing prototypes and realistic evaluations demonstrating greater robustness and usefulness. The research can help solve the problem of incorrect software by improving robustness, while enabling a wider cross-section of society to develop and personalize software. This can lead to greater understanding, satisfaction, and utility of software that behaves as people generally want it to.

3.6 "Microtask" vs. freelancer platforms – how crowdsourcing can complement software development

Robert Kern (IBM Deutschland – Böblingen, DE)

In the era of cloud computing, mobile computing, collaboration and big data, software development requirements are significantly changing. Users and organizations are asking for shorter development cycles, improved ease of use, better integration and lower administration and operation overhead. This results in a need for flexible software development and operation

processes combined with a new assignment of roles in order to hide manual efforts from the user.

Crowdsourcing has the potential to address these challenges in several ways. On the one hand, platforms like TopCoder or oDesk enable for flexible outsourcing of design and implementation efforts to freelancers. On the other hand, "microtask" platforms like Crowdflower or Amazon MTurk provide human workforce as a scalable service in order to fulfill formalized tasks which are difficult to automate. Such services can either be integrated into SW services to deliver complex services like search engine optimization, data cleansing and social media analysis, or they can complement SW operation and development, e.g. by providing testing services.

After contrasting the two types of platforms, this presentation focuses on the latter type. By elaborating on the analogy of cloud computing and crowdsourcing, a "cloud labor" stack is introduced that seamlessly integrates with the concepts of cloud computing. The capabilities provided by the different layers of the stack are identified and implementation options are outlined. The considerations are then validated by discussing a crowd-based medical coding service offered by IBM. Finally, a bow is drawn back to the freelancer platforms and initial ideas are provided on how the two complementary concepts could be merged in the future.

3.7 Software Development Crowdsourcing Issues: Organization Design and Incentive Design

Donghui Lin (Kyoto University, JP)

In the talk, we discuss two important issues in Software Development Crowdsourcing: Organization Design and Incentive Design. We share the experience of problem-based learning course of crowdsourcing in university.

3.8 Crowdsourcing with Expertise

Greg Little (ODesk Corp. – Redwood City, US)

Crowdsourcing often focusses on small tasks that require no special skills other than being human. I'm interested in tasks and workflows that do require special skills, like programming or art skills. For instance, one could imagine a crowdsourced logo that involves several phases. The first phase might pay 20 sketch artists to sketch logo ideas, and then pay a traditional crowd to vote for the best 5. The next phase might then pay 5 designers to flesh out the best 5 logos from the previous phase, followed by another round of voting. The final phase might involve hiring an expert to finalize the top design.

Crowdsourcing with expertise is difficult today because it is difficult to programmatically hire many experts – hiring experts typically involves looking at resumes and portfolios. My research has focussed on finding scalable ways to reliably identify experts. Two promising ideas include: (1) having people play simple subjective games based on their expertise, and rating each others results; (2) having public work histories which include both the task and

the results, as opposed to portfolios today which typically only display the results, but it is hard to know what this worker contributed, or whether their result met their client's requirements.

3.9 Multi-Agent System Models and Approach of Crowdsourcing Software Development

Xinjun Mao (National University of Defense Technology – Changsha, China)

The advent and successful practices of software crowdsourcing needs to investigate its in-depth essence and seek effective technologies to support its activities and satisfy increasing requirements. We highlight crowdsourcing participants consist of a multi-agent system and software crowdsourcing is a multi-agent problem-solving process. This paper discusses the characteristics and potential challenges of software crowdsourcing in contrast to traditional software development, and present a general analysis framework based on multi-agent system to examine the organization and behaviors of software crowdsourcing. Several software crowdsourcing models performed on typical platforms like Topcode, uTest are established and their organization and coordination are discussed. We have developed a service-based multi-agent system platform called AutoService that provides some fundamental capabilities like autonomy, monitoring, flexible interaction and organization, and can serve as an infrastructure to support software crowdsourcing models and tackle its challenges. A software crowdsourcing prototype is developed and some scenarios are exemplified to illustrate our approach.

3.10 Crowds, Clouds, Agents and Coordination

Dave Murray-Rust (University of Edinburgh, GB)

Joint work of Murray-Rust, Dave; Robertson, Dave

The term "social machines" describes a class of systems where humans and machines interact, and the mechanical infrastructure supports human creativity. As well as software crowdsourcing projects such as TopCoder and oDesk, this includes distributed development platforms such at GitHub and Bitbucket. In this paper, we describe a formalism for social machines, consisting of i) a community of humans and their "social software" interacting with ii) a collection of computational resources and their associated state, protocols and ability to analyse data and make inferences. These social machines are increasingly the target of software development, and as such, they represent an interesting problem, as the community must be "programmed" as well as the machines. This leads to evolving and unknown requirements, and having to deal with much softer concepts than formal systems designers usually work with. Our model hence uses two coupled social machines. There is the target machine, and the machine which is used to create it—much as GitHub and associated resources might be used to form a social machine to create "the next Facebook". We draw on the ideas of 'desire lines' and 'play-in' to argue that top down design of social machines is impossible, that we hence need to leverage computational support in creating complex systems in an iterative, dynamic and emergent manner, and that our formalism provides a possible blueprint for how to do this.

3.11 Collaborative Majority Vote: Improving Result Quality in Crowdsourcing Marketplaces

Khrystyna Nordheimer (Universität Mannheim, DE)

Joint work of Nordheimer, Dennis; Nordheimer, Khrystyna; Schader, Martin; Korthaus, Axel

Crowdsourcing markets, such as Amazon Mechanical Turk, are designed for easy distribution of micro-tasks to an on-demand scalable workforce. Improving the quality of the submitted results is still one of the main challenges for quality control management in these markets. Although beneficial effects of synchronous collaboration on the quality of work are well-established in other domains, interaction and collaboration mechanisms are not yet supported by most crowdsourcing platforms, and thus, not considered as a means of ensuring high-quality processing of tasks. In this paper, we address this challenge and present a new method that extends majority vote, one of the most widely used quality assurance mechanisms, enabling workers to interact and communicate during task execution. We illustrate how to apply this method to the basic scenarios of task execution and present the enabling technology for the proposed real-time collaborative extension. We summarize its positive impacts on the quality of results and discuss its limitations.

3.12 The Open Source Volunteering Process

Dirk Riehle (Universität Erlangen-Nürnberg, DE)

Today's software systems build on open source software. Thus, we need to understand how to successfully create, nurture, and mature the software development communities of these open source projects. In this article, we review and discuss best practices of the open source volunteering and recruitment process that successful project leaders are using to lead their projects to success. We combine the perspective of the volunteer, looking at a project, with the perspective of a project leader, looking to find additional volunteers for the project. We identify a five-stage process consisting of a connecting, understanding, engaging, performing, and leading stage. The underlying best practices, when applied, significantly increase the chances for a successful open source project.

3.13 Artifact-centric Incentive Mechanisms for Socio-technical Systems

Ognjen Scekic (TU Wien, AT)

Joint work of Scekic, Ognjen; Truong, Hong-Linh; Dustdar, Schahram
Main reference O. Scekic, H.-L. Truong, S. Dustdar, "Incentives and rewarding in social computing," Communications of the ACM, 56(6):72–82, 2013.
URL http://dx.doi.org/10.1145/2461256.2461275

Crowdsourcing systems of the future (e.g., social compute Units – SCUs, collective adaptive systems) promise to support processing of richer and more complex tasks, such as software development. This presupposes deploying ad-hoc assembled teams of human and machine

services that actively collaborate and communicate among each other, exchanging different artifacts and jointly processing them. Major challenges in such environments include team formation and adaptation, task splitting and aggregation, and runtime management of data flow and dependencies, collaboration patterns and coordination mechanisms. These challenges can be somewhat alleviated by delegating the responsibility and the know-how needed for these duties to the participating crowd members, while indirectly controlling and stimulating them through appropriate incentive mechanisms.

In this paper we present a novel, artifact-centric approach for modeling and deploying incentives in rich crowdsourcing environments. Artifact's lifecycle model is augmented with incentive mechanisms to create encapsulated units that can be offered to the crowd for processing. The incentive mechanisms are adaptive and constantly advertised to the crowd to drive the processing in the envisioned direction and tackle the aforementioned challenges. They are designed to promote teamwork, and to support time and data dependencies.

3.14 Engineering Multi-Cloud Service-Oriented Applications

Lionel Seinturier (Lille I University, FR)

Joint work of Paraiso, Fawaz; Merle, Philippe; Seinturier, Lionel

Cloud platforms are increasingly being used for hosting a broad diversity of services from traditional e-commerce applications to interactive web-based IDEs and crowdsourcing systems. However, the proliferation of offers by cloud providers raises several challenges. Developers will not only have to deploy applications for a specific cloud, but will also have to consider migrating services from one cloud to another, and to manage distributed applications spanning multiple clouds. In order to address these challenges, we present a federated multi-cloud PaaS infrastructure that is based on three foundations: i) an open service model used to design and implement both our multi-cloud PaaS and the SaaS applications running on top of it, ii) a configurable architecture of the federated PaaS, and iii) some infrastructure services for managing both our multi-cloud PaaS and the SaaS applications. We report on the deployment of this cloud-based infrastructure on top of 10 existing IaaS/PaaS.

3.15 On Assuring Quality of Results in Hybrid Compute Units in the Cloud

Hong-Linh Truong (TU Wien, AT)

Joint work of Truong, Hong-Linh; Dustdar, Schahram

Recently there is an increasing trend of utilizing hybrid compute units consisting of software-based and human-based services to solve complex problems. When both software-based and human-based services are provisioned based on pay-per-use, it is a great challenge to assure the quality of results, which center on the cost, response time and the quality of data. We discuss a range of techniques, from composition to monitoring phases, to support the assurance of quality of results in cloud-based crowdsourcing software development.

3.16 Software Crowdsourcing Maturity Models

Wei-Tek Tsai (ASU, Tempe, AZ , US)

Some speculated that crowdsourcing can be used for ultra large systems. But examinations of existing crowdsourcing websites such as TopCoders, uTest, and AppStori indicates that it may take a long time to reach that levels of maturity. What will be the roadmap for crowdsourcing growth? What will be the platform architecture to support future crowdsourcing? How about conventional issues such as testing (including regression), configuration, scalability, software architecture, fault-tolerant issues, concurrent software development processes?

Here we define a Crowdsourcing Maturity Model:

Level 1: Single persons, well-defined modules, small size, limited time span (less than few months), quality products, current development processes such as the one by AppStori, TopCoder, and uTest. At this level, coders are ranked, websites contains online repository crowdsourcing materials, software can be ranked by participants, crowdsourcing platforms have communication tools such as wiki, blogs, comments as well as software development tools such as IDE, testing, compilers, simulation, modeling, and program analysis.

Level 2: Teams of people (< 10), well-defined systems, medium size, medium time span (multiple months to less than one year), and adaptive development processes with intelligent feedback in a common cloud platform where people can freely share thoughts. At this level, a crowdsourcing platform supports adaptive development process that allow concurrent development processes with feedback from fellow participants; intelligent analysis of coders, software products, and comments; multi-phase software testing and evaluation; Big Data analytics, automated wrapping software services into SaaS (Software-as-a-Service), annotate with ontology, cross reference to DBpedia, and Wikipedia; automated analysis and classification of software services; ontology annotation and reasoning such as linking those service with compatible input/output.

Level 3: Teams of people (< 100 and > 10), well-defined system, large systems, long time span (< 2 years), automated cross verification and cross comparison among contributions. A crowdsourcing platform at this level contains automated matching of requirements to existing components including matching of specification, services, and tests; automated regression testing.

Level 4: Multinational collaboration of large and adaptive systems. A crowdsourcing platform at this level may contain domain-oriented crowdsourcing with ontology, reasoning, and annotation; automated cross verification and test generation processes; automated configuration of crowdsourcing platform; and may restructure the platform as SaaS with tenant customization.

3.17 Trustie: a Platform for Software Development Ecosystem incorporating Engineering Methods and Crowd Wisdom

Huaimin Wang (NUDT – Hunan, CN)

Software production activity includes the process of software creation, which relies on developers' inspiration and talent, and the process of software manufacture, which is executed under strict engineering code. Traditional Software Engineering stresses software manufacture, while the Open Source Software development focuses on software creation. These two software production models each has its strengths and weaknesses, but are complementary to each other. Literature research and engineering practices have shown that traditional software engineering approaches have encountered obstacles in several creation activities such as requirement elicitation. By exploiting the crowd wisdom, open source development provides a better environment for software creation. However, it can neither make promise on specific software requirements, nor guarantee the progress and quality of production. In this paper, we suggest to introduce a software development ecosystem which combines the strengths of these two models. First, we propose the general service model of a platform which can support such a dual function development ecosystem. Architecturally, the core of this service model contains a novel software process model and software evidence model. Besides, it integrates collaborative development, resource sharing and analysis into a unified framework. Based on this service model, we designed Trustie: a software development platform, under the support of a Chinese national 863 grand project. Trustie is equipped with built-in collaboration tools and an open and evolving software repository with deep analysis utilities. Trustie bridges software manufacture activities and software creation activities in three featured aspects:

1. It enables crowd-oriented collaboration among developers and stakeholders.
2. It supports massive software resource and knowledge sharing.
3. It provides historical software data analysis and software quality evaluation.

Up till now, Trustie has already been successfully adopted by a series of on-line projects and large-scale industry applications.

3.18 Crowdsourcing for Software Ecosystem

Wenjun Wu (Beihang University – Beijing, CN)

Software development is complex and creative as it involves requirement analysis, design, architecture, coding, testing and evaluation. Recently, software crowdsourcing, or outsource software development to the crowd, has been popular with numerous software coders participated in various software competitions. We first analyze the data collected on popular software crowdsourcing and summarizes major lessons learned. We then examine two popular software crowdsourcing processes including TopCoder and AppStori processes. Specifically, this paper evaluates competition rules used in these processes, and compare with a traditional software development process IBM Cleanroom methodology as it claims of delivering zero-defect software. We identify an important design element in software crowdsourcing for software quality and creativity is the min-max nature among participants. The min-max nature comes from game theory where participants compete to win the game with one party tires to

minimize an objective or the other party tries to maximize the same object. However, in software crowdsourcing, the min-max is done by one party tries to maximize the finding of bugs in a set of artifacts, and the other parties try to minimize the potential bugs in the same artifact. The min-max can be practiced without being a zero-sum competition where a gain from one party will be matched by another party. In other words, software crowdsourcing can be a win-win for all parties and they can collaborate while practicing a min-max game. By using this approach, lots of aspects of software development can be crowdsourced from initial project concepts, to specification, design, algorithms, coding, and testing, with the crowd can contribute their creativity to each aspect.

3.19 An Evolutionary and Automated Virtual Team Making Approach for Crowdsourcing Platforms

Tao Yue (Simula Research Laboratory – Lysaker, NO)

Crowdsourcing has demonstrated its capability of supporting various software development activities including development and testing as it can be seen by several successful crowdsourcing platforms such as TopCoder and uTest. However, to crowd source large-scale and complex software development and testing tasks, there are several optimization challenges to be addressed such as division of tasks, searching a set of registrants, and assignment of tasks to registrants. Since in crowdsourcing a task can be assigned to registrants geographically distributed with various background, the quality of final task deliverables is a key issue. As the first step to improve the quality, we propose a systematic and automated approach to optimize the assignment of registrants in a crowdsourcing platform to a crowdsourcing task. The objective is to find the best fit of a group of registrants to the defined task. A few examples of factors forming the optimization problem include budget defined by the task submitter and pay expectation from a registrant, skills required by a task, skills of a registrant, task delivering deadline, and availability of a registrant.

4 Working Groups

4.1 Cloud Infrastructure for Software Crowdsourcing

Wei-Tek Tsai

The group discussed about this topic from two point of views:

1. Understand the current cloud infrastructures such as IaaS (infrastructure-as-a-service), PaaS (platform-as-a-service), and SaaS (software-as-a-service), and how these infrastructures can support software crowdsourcing.
2. Understanding the needs of software crowdsourcing including its processes, organization structure, collaboration patterns, user support, user interface, and payment features. Cloud Features to Support Software Crowdsourcing

In general, the group felt that most of cloud infrastructure including all the related features such as metadata-based design, scalability architecture, multi-tenancy architec-

ture, automated migration, automated redundancy management, automated recovery, new database design, runtime system composition, execution, and monitoring are useful to support software crowdsourcing. These features may be made as menu selection for software crowdsourcing organizers. For example, an organizer may like triple redundancy for collaboration-based crowdsourcing, but additional redundancy for competition-based crowdsourcing. The additional redundancy is needed to ensure that competition data will not be lost.

Software Crowdsourcing Process and Organization Needs

In general, the group felt that while different software crowdsourcing processes may have different needs, but among the diversity of software crowdsourcing processes, they actually share much commonality. Common themes for software crowdsourcing processes include:

1. Collaboration and communication tools such as a distributed blackboard system where each party can participate in discussions,
2. Participant ranking and recommendation tools;
3. Software development tools such as modeling tools, simulation tools, programming language tools such as compilers and intelligent editors, design notations, and testing tools;
4. Cloud payment and credit management tools;
5. Cloud service management dashboard for system administrators and software crowdsourcing organizers;
6. Board where user can register and upload their profiles information; and
7. Repository of software development assets such as modules, specifications, architecture and design patterns.

Major variations will come from different software crowdsourcing processes, for example, competition-based processes such as those done by TopCoder will be different from collaboration-based processes, such as by AppStori. Competitions require enforcement of game rules including time management and rigorous evaluation, collaborations require communication, publishing, sharing, alerts, automated text and index processing.

The group came out with a draft reference architecture as in Fig. 1.

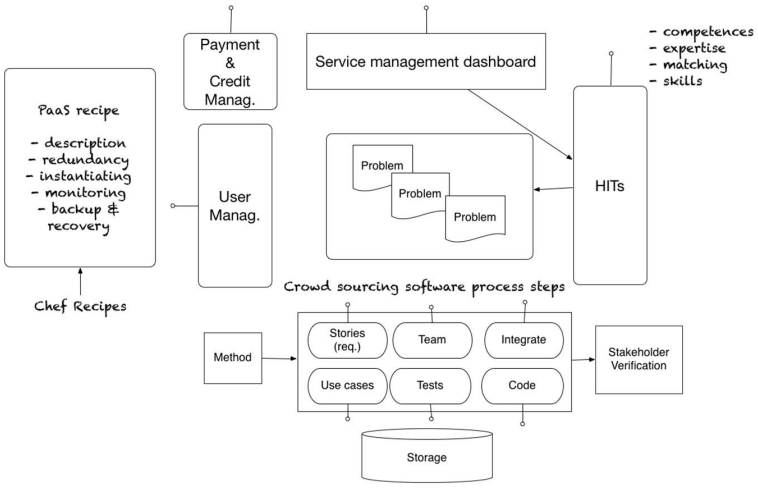

Figure 1 Reference architecture of cloud-based software crowdsourcing.

4.2 Crowd Source Software Engineering Design

Shaukat Ali

Joint work of Shaukat Ali, Michael Huhns, Khrystyna Nordheimer, Yue Tao, Hong-Linh Truong, Wenjun Wu

The discussion in this group was focused on Crowd Source Software Engineering design in the three main areas, processes, models, and techniques.

4.2.1 Processes

Software engineering development projects based on crowd sourcing can be developed using any software development process models such as Water Fall and Agile development. Such information may be requested by a project submitted or alternatively can be suggested by a cloud platform. Which approach is better is still an open issue and requires further investigation.

4.2.2 Models

There is a need for standardized generic models (ontologies/domain specific language) for capturing information about tasks (such as task description, micro-tasks and atomic tasks), individual in crowds (such as expertise, payment history and successful project history) and project information (such as cost, time, and required expertise). There are several benefits to create such models:

1. A standardized way of capturing various concepts in crowd sourcing and their relationships to promote a common and unified understanding in the community. These can further be pushed for standardization in the future.
2. A systematic way of specializing the models for specific applications of software development such as implementation and testing.
3. Promote automation of various activities in crowd sourcing such as team formation and ranking.

An important issue is how to divide a project into a set of atomic tasks that can be performed by different individuals in the crowd independently. The following potential solutions were discussed:

1. A project submitted by a project owner may be divided into atomic tasks by the project owner. However, this may not always be optimal if not impossible. Such process may be improved by restricting the project owner to submit the project in a specific format that can facilitate division of the project into atomic tasks. However, this raises several potential open issues such as which kind of restricted format can be used and if there are several alternatives, which one is the best one for what kind of tasks.
2. An semi/automated way of introducing a platform independent broker, which could somehow divides the project into independent tasks, which can then be sent to crowd sourcing platforms. Such broker may not be visible to the project submitted.
3. A common practice in the current crowd sourcing platforms is the use of man-in-the-loop approach, where a project moderator coordinates with the bidders and help dividing the project into different atomic tasks.

4.2.3 Techniques

Crowd sourcing activities may be implemented by various different techniques. In the discussion, first we talked about various techniques for ranking individuals in a crowd. Several machine learning techniques have been applied by various researchers to rank individuals. Secondly, we talked about team formation techniques. Similar to ranking techniques, different approaches such as based on search algorithms and machine learning have been applied for team formation. A fundamental question to answer for both of the above activities is which approaches are cost/effective in which situations. Doing so requires thorough empirical evaluations based on focused case studies to study strengths and weakness of the proposed approaches.

4.2.4 Other Open Issues

In addition, we also discussed some of the open issues related to the quality of crowd sourcing from various perspectives:

- How to assess the quality of a crowd?
- How to assess the quality of an individual in a crowd?
- How to assess the quality of crowd software?
- How to assess the quality of crowd sourcing infrastructure?

4.3 Worker-centric design for software crowdsourcing

Dave Murray-Rust (University of Edinburgh, GB)

Joint work of Murray-Rust, Dave; Yin, Gang; Scekic, Ognjen; Wang, Huaimin; Lin, Donghui; Mao, Xinjun; Kern, Robert; Little, Greg; Riehle, Dirk

Crowdsourcing is emerging as a compelling technique for the cost-effective creation of software, with tools such as ODesk and TopCoder supporting large scale distributed development. From the point of view of the commissioners of software, there are many advantages to crowdsourcing work – as well as cost, it can be a more scalable process, as there is the possibility of selecting from a large pool of expertise. From the point of view of workers, there is a different set of benefits, including choice of when and how to work, providing a means to build a portfolio, and a lower level of commitment to any particular employer.

Most analyses of crowdsourcing take the point of view of the commissioners of work: how is it possible to get work done better, more cheaply, more robustly etc. When considered as an "outsider" technology, this need to prove value to commissioners of work is completely understandable. Crowdsourcing is no longer a niche activity, however. From 2000–2009, cloudworkers had been paid up to $2Bn; the number of participants has grown by over 100% per year, and there are now over 6 million cloudworkers worldwide.

In this group, we engaged with the question of what it would take to make software crowdsourcing a sustainable industry. This means being able to attract intelligent, motivated individuals, who can make enough money to satisfy themselves. Essentially, we asked the question "What would we want from a crowdsourcing marketplace if we were going to work in it".

Software crowdsourcing is markedly different to Turking and other crowdlabour projects, for a number of reasons. If crowdlabour is divided into micro tasks, macro tasks, small projects and complex projects, then while much of the crowdsourcing industry focusses on

Mechanical Turk style microtasks, software creation tends to fall into the small- or complex-project brackets, requiring workers to bring in existing skills, and some degree of coordination or direct worker contact. However, if crowdsourcing were to replace traditional employment for a significant proportion of software developers, the reduced levels of commitment between workers and commissioners could prove problematic for workers over time. In this paper, we identify several areas of interest, and discuss what the issues are, and how current and future solutions could address them: careers and reputation development; trust; team selection and team building; and contextualisation of the work carried out.

4.3.1 Trust and Reputation

Arguably, co-workers are one of the most important factors contributing to a pleasant and productive working environment . In traditional companies workers usually cannot directly select their co-workers. However, since the nature of the employment relationship is a long-lasting one, it gives them time to get to know their colleagues and forge working relationships. The management will actively monitor these relationships in order to achieve a more harmonic, and thus more productive or creative environment.

In crowdsourcing environments, the relationship of workers with the platform and co-workers are irregular and short-lived. This leaves no time to get to know and other workers. Crowdsourced teams are often unique, both time- and composition-wise. Co-workers are often hidden behind digital profiles, creating an atmosphere of distrust and uncomfort. Furthermore, such settings make for an attractive environment for attempting fraudulent activities, such as multitasking, rent-seeking or tragedy of the commons

The proliferation of different metrics and trust models indicate that agreeing on a uniform, context-independent trust and reputation model is practically unfeasible. A new approach and some out-of-the-box thinking will be needed take to address this problem.

4.3.2 Team Selection

In the crowdsourcing environment, large complex projects always need to be conducted by cooperation of a number of crowd workers. As well as the issues of trust and reputation discussed previously, the composition of the team can have an effect on performance, and also the satisfaction of the workers who constitute the teams.

Existing crowdsourcing platforms do not have much support for coordination and interaction among crowd workers. In some platforms like Amazon Mechanical Turk, tasks are separated in an atomic manner so that crowd workers do not need to collaborate with each other. Other platforms support offline collaboration among crowd workers with the guidance of the work requester for complex projects. However, creative work like software development requires a large degree of knowledge integration, coordinated effort and interaction among workers.

The task allocation problem has been discussed for decades in artificial intelligence and distributed computing circles. In crowdsoucing environments, recent researches focus on task decomposition for modeling appropriate workflow with iterative tasks for the purpose of quality assurance. However, task matching for crowd workers is also important for creative complex task in crowdsourcing, where two main factors should be considered: the skills posessed by crowd-workers, and the incentives needed to motivate them.

It will become increasingly necessary to provide mechanisms by which software crowd-workers can collaborate with people they know and trust; where they can organise themselves

effectively as situtations and contexts evolve; and where they are able to utilise—and improve—their skils on a variety of non-monotonous tasks.

4.3.3 Contextualisation

The context and purpose of software development can be a large motivating factor for workers; there is a need to reduce the anonymity on both working and comissioning sides, to provide task context; decontextualized tasks remove the ability of workers to understand the moral valence of their labour, and decide whether the task they are carrying out is morally acceptable to them. Examples range from spammers attempting to break Captchas to governments outsourcing recognition of persons of interest in photographs.

When discussing general collective intelligence situations, Malone describes the reward for taking part as being based on "Money, Love or Glory". The complement of this (leaving aside the pecuniary aspects) is that one should be engaged in a task that one does not hate, and is not ashamed of. Additionally, Malone suggests that commissioners of collective intelligence should engage with the design questions: What is being done? Who is doing it? Why are they doing it? How is it being done? These questions can be reversed to create a list of questions which crowdworkers should be able to ask, both for their own peace of mind and as a way for commissioning entities to engage with the Love and Glory motivations:

- "What is the overall project?"
- "Who is comissioning the work?"
- "Why are they comissioning it?"
- "How is it being carried out?"

We have discussed the need for trust and repuation between crowdworkers; there is also the need for accountability for comissioners of work. Requesters on Mechanical Turk currently are not bound by reputation systems, allowing them to act with impunity when designing tasks.

Traditional workers have the benefit of many organisational structures that support them. Labour laws ensure a safe and healthy environment; employers are tasked with managing the physical space that they inhabit in working hours; they will meet other people in they workplace; advocacy groups and unions may exist to represent the needs of workers. For a crowd working career, something providing some of the properties of these structures would need to be created.

4.4 Architecture for Cloud-based Software CrowdSourcing

Michael Maximilien (IBM Almaden Center – San José, USA)

Joint work of Michael Maximilien, Xiaoying Bai, Kyle Chard, Schahram Dustar, Wei-Tei Tsai, Lionel Seinturier

Crowd-sourcing platforms have been mostly proprietary for the most parts. Unlike trendy cloud platforms, there are no dominant platforms for building crowd-sourcing applications. This could be due to the fact that crowd-sourcing applications are varied and have similar needs as those for cloud applications. However, in our discussions and analysis, it seems that common architectures may actually exist for crowd-sourcing applications, and creating a reference OSS version of such platform might be of interest.

With this aim, we have explored two complementary approaches which we discuss here:

4.4.1 Cloud Applications – Specifically Crowd-Sourcing for PaaS

In this approach we are assuming that a Platform-as-a-Service (PaaS) exists, such as the ones provided by Heroku, Microsoft Azure, or the OSS CloudFoundry as well as others. The crowd-sourcing application is simply installed onto this PaaS and tuned to address its needs. We are assuming that the PaaS platform provides primitives to support the needs of the crowd-sourcing application, such as, a flexible database, e.g., MongoDB, an application server, e.g., Ruby on Rails or Node.js, as well as various connections with services that might also be needed, e.g., LDAP, OAuth, Payment, and others. Since the crowd-sourcing application integrates into the PaaS it can be scaled up and down using the PaaS primitives. This usually involves adding more containers for the crowd-sourcing application deployment, e.g., replicating the application server layer and adding a high-availability (HA) proxy.

4.4.2 Distributed Loosely Coupled Crowd-Sourcing Platform

In this approach, the focus is on the components making up the crowd-sourcing platform and pulling them apart and distributing them. That way each component exposes an API(s) and can be combined to create the crowd-sourcing application. Where and how these components are hosted is not taken into consideration. A PaaS could be used or the component could be hosted directly onto an IaaS or bare metal environment. To illustrate the point, think of a simplified crowd-sourcing application needing to host tasks and have a pool of workers. One API would be to retrieve, update, and create tasks. One API would be to retrieve, update, and create new workers. And one more API would be to match workers to tasks and get the workers paid when tasks are completed. In such a distributed service-oriented approach the crowd-sourcing application becomes essentially like a mashup of these primitive crowd-sourcing APIs. Of course, a challenge there is security and scaling since you have no control into the storage, security measures, and scaling capability of the APIs you are using.

4.5 Infrastructure and Platform

Xiaoying Bai (Tsinghua University, CN)

Joint work of Michael N. Huhns, Michael Maximilien, Xavier Blanc, Greg Little, Huaimin Wang, Gang Yin, Kyle Chard, Robert Kern, Lionel Seinturier

Crowdsourcing promotes software development by open collaborations. Cloud is expected to provide necessary services to support development and collaboration activities. To address the needs, the group discussion reviewed the motivations to construct the platform, identified the key components of Cloud infrastructure and platform, analyzed architecture design issues, and proposed an experiment to build a crowdsourcing platform for educating software development.

4.5.1 Motivation Revisited

Crowdsourcing is a promising development method to reduce cost and enhance quality by contracting experts in open communities. However, it is not easy to organize crowdsourced software development. It requires a lot of effort and experiences to decompose software into well-organized modules and development tasks, to find good candidates for each task, to communicate requirements clearly and precisely, and to integrate submitted code into project framework. Hence, crowdsourcing may be not suitable for software development in

general, but for specific modules, whose requirements and outcome can be explicitly defined, even quantitatively measured. For example, companies may outsource algorithm design by sponsoring a TopCoder algorithm completion. From another perspective, instead of an engineering platform, crowdsourcing could be an incubator for innovations. It seeks ideas and solution options from crowd wisdom. Problems like GUI design can be open to communities to foster innovative ideas.

4.5.2 Architecture Design

The Cloud for crowdsourcing need to provide services for two categories of activities: software development and open collaboration. To support software development, it needs software lifecycle process tracking platform from requirements distribution to software testing and integration, code repositories with version control system and online IDE environment, document management system, and so on. To support community-based collaboration, it needs to support user account management, expert recommendation, group coordination, and so on. Following the open architecture principle, some of the services can be built by reusing with existing open services, for examples, GitHub as implementation repository, Amazon for computing and storage resources, Cloud9 as online IDE, Google App Engine for software hosting and Google Doc for group discussion. It can also integrate with other system such as sharing use accounts from different social network systems. In addition, it has unique requirements to support this new development method, especially for community management and collaborations. For example, reputation and profile management, expert searching, evaluation and recommendation, and so on.

4.5.3 Experiment: Educating Software Engineering by a Crowdsourcing Platform

A platform can be built for educating purpose so that SE courses from different Universities can share their resources and collaborate in an experiment process. Students can join teams and projects across the world to learn by experience global software development methods, techniques, and challenges.

4.6 Experimentation Design for Software Crowdsourcing

Wenjun Wu (Beihang University, CN)

Joint work of Yue Tao, Shuat Ali, Wei-Tek Tsai; Hong-Linh Truong; Ognjen Scekic; Donghu Lin; Xinjun Mao

In this group, we discussed how to design and conduct software crowdsourcing experiments to validate theoretical models and refine crowdsourcing algorithms. We also listed the major crowdsourcing infrastructures that can be used to support experiments. Given the dynamic nature of crowdsourcing activity, it is a challenge to develop a repeatable experimental plan. Researchers must consider both social and technical factors in a software crowdsourcing process. Social factors such as constitutes of the community, skill and knowledge levels of participants, the incentive of participation often have significant impact on possibility of replicating the same experiment. Technical factor such as software development process and cloud software products are also vital to enable in-depth investigation among the synergy between people and software system. A successful software experiment should carefully integrate these major elements in the settings and describe them in a clearly defined model.

With such a model, the research community of crowdsourcing can share their datasets and derive knowledge from them through cast studies and statistics-driven data analytics.

4.6.1 Experimental Approaches

There are three approaches to perform crowdsourcing experiments and analysis: data analysis based on the process of real projects on crowdsourcing platforms, controlled experiments and simulation.

First, one can always gather data from existing projects that are hosted on the major crowdsourcing platforms such Topcoder [1], Amazon Mechnical Turk [2], Github [3] and Sourceforge [4]. Such a community platform keeps track of software development activities of tens of thousands completed projects. Via web crawlers or access APIs of the platforms, we can download all the activity data and make further analysis to to verify software crowdsourcing models. The advantage of this analytic approach is to access real crowdsourcing data with a low monetary cost and time effort. It is especially useful for statistical analysis and case studies.

Second, when a researcher needs to run controlled experiments to compare different strategies and mechanisms with software crowdsourcing design, he can launch real projects on the platforms mentioned above. The positive side of this approach is that researchers can devise their own experimental scenarios and explore completely new schemes that may never be tried by others. But since the experiment is run on commercial platforms, the researcher has to pay for recruiting community workers as his study subjects.

Last, one can always rely on simulation tools to run software crowsourcing experiments. Actually, one of the talks presented in this seminar discussed how to design simulation experiments following the models of search-based software engineering to study software crowdsourcing. Also, agent-based simulation can be adopted to study incentive models and production quality assurance on large-scale crowds. The major benefit of simulation method is flexibility and low-cost. And it empowers researchers to check their models in a much larger setting than what is available in real environments.

Perhaps a hybrid method combining real projects, data analysis and simulations will be the best way to practice software crowdsourcing studies. Imaging we can model a software crowdsourcing process through data analysis of public crowdsourcing platforms. More case studies based on real projects can be done to improve the model. And the model can be further used on a simulation scenario to demonstrate emergent behaviors in a large-scale crowdsourcing system.

4.6.2 Experiment Design

A researcher needs to define his experimental objective, primary factors and study objects before starting his experiments on software crowdsourcing:

Whether does he plan to verify his model or algorithm? How can he model crowdsourcing tasks, workers, and costs in term of time and budget? How to design the mechanism for ranking and expertise matching to facilitate virtual team formation?

In the design of a controlled experiment, a researcher should isolate or control multiple design factors in order to identify primary factors that are most influential. Typical factors in a crowdsourcing experiment include participants, prize setting, and task design and so on. If he wants to test a crowdsourcing task among the same group of people, he may have to design a two-around crowdsourcing mechanism to retain the participants who have involved in the previous around.

Choice of study subjects in an experiment can also significantly affect the outcome. Since the community is often the study subject in a crowdsourcing project, a researcher must decide whether to run the project with IT professionals or college students. Our group believes that experiments with student groups can produce valid results as ones with professional developers. They are suitable in different kinds of experiments. IT professionals, especially software engineers are always good candidates for software engineering experiments that demands more advanced programming skills and expertise for system design. In other kinds of experiments with less restricted requirements for software design and programming, researcher can choose computer science students as his study subject.

The last but not the least concern for researchers is to design an appropriate ontology model to describe software crowdsourcing processes. If everyone organizes his datasets in a standard process model like SPEM [5] in UML specification, this data interoperability enables very convenient data exchange within the community and facilitates peer researchers to reuse the dataset in their experiments.

4.6.3 Experiment infrastructure

Our discussion group listed the major commercial platforms available for researchers. Well-known examples of crowdsourcing platforms are Topcoder, Amazon Mechanical Turk and Odesk [6]. The distinguishing features among these platforms include community and crowdsourcing style. Most players in the Topcoder community are young people with strong enthusiasm for programming. So when a researcher needs his coding project accomplished within a week, Topcoder seems a good place to go. Because most crowdsourcing tasks in Amazon Mechanical Turk are very cheap laboring jobs that often require only a few minutes of effort. So its community is not very suitable for programming tasks demanding dedication and high skills.

In addition to these above commercial platforms, group members highlighted open source crowdsourcing platforms under development. Vienna's team mentioned about a plan to release a crowdsourcing service platform. Beihang team is developing an educational platform to support Massive Open Online Courses (MOOCs), which could be extended to support software crowdsourcing with MOOC learners. National University of Defense Technology introduces two open source platforms namely Trustie (http://www.trustie.net/) and OW2 Open Source community (http://ow2.org/), which combine the traditional software engineering methods with the crowd wisdom, and provide a promising infrastructure to enable software crowdsourcing projects. Further work is needed to explore all the platforms and build up good cases so that we could summarize the best practices and define guidelines to perform software crowdsourcing experiments.

References

1 Topcoder, http://www.topcoder.com
2 Amazon Mechnical Turk, http://www.mturk.com
3 Github, http://www.github.org
4 Sourceforge, http://www.sourceforge.net
5 SPEM, http://www.omg.org/spec/SPEM/2.0/?
6 Odesk, http://www.odesk.com

5 Open Problems

Open problems were described throughout the previous sections, in particular, in the working group summaries.

Participants

- Shaukat Ali
Simula Research Laboratory – Lysaker, NO
- Xiaoying Bai
Tsinghua Univ. – Beijing, CN
- Xavier Blanc
University of Bordeaux, FR
- Kyle Chard
University of Chicago, US
- Schahram Dustdar
TU Wien, AT
- Michael N. Huhns
University of South Carolina – Columbia, US
- Robert Kern
IBM Deutschland – Böblingen, DE
- Donghui Lin
Kyoto University, JP
- Greg Little
ODesk Corp. – Redwood City, US
- Xinjun Mao
National University of Defense Technology – Hunan, CN
- Michael Maximilien
IBM Almaden Center – San José, US
- Dave Murray-Rust
University of Edinburgh, GB
- Khrystyna Nordheimer
Universität Mannheim, DE
- Dirk Riehle
Univ. Erlangen-Nürnberg, DE
- Ognjen Scekic
TU Wien, AT
- Lionel Seinturier
Lille I University, FR
- Hong-Linh Truong
TU Wien, AT
- Wei-Tek Tsai
ASU – Tempe, US
- Huaimin Wang
NUDT – Hunan, CN
- Wenjun Wu
Beihang University – Beijing, CN
- Gang Yin
NUDT – Hunan, CN
- Tao Yue
Simula Research Laboratory – Lysaker, NO