Zhang X, Zhang D, Traon YL *et al.* Roundtable: Research opportunities and challenges for emerging software systems. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(5): 935–941 Sept. 2015. DOI 10.1007/s11390-015-1572-8

Roundtable: Research Opportunities and Challenges for Emerging Software Systems

Xiangyu Zhang [1] (张翔宇), Dongmei Zhang [2] (张冬梅), Yves Le Traon [3], Qing Wang [4] (王 青), and Lu Zhang [5] (张 路)

[1] *Department of Computer Science, Purdue University, West Lafayette, IN 47907, U.S.A.*

[2] *Microsoft Research, Beijing 100080, China*

[3] *Faculty of Science, Technology and Communication, University of Luxembourg, Luxembourg*

[4] *Institute of Software, Chinese Academy of Sciences, Beijing 100190, China*

[5] *School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China*

E-mail: xyzhang@cs.purdue.edu; dongmeiz@microsoft.com; yves.letraon@uni.lu; wq@itechs.iscas.ac.cn zhanglu@sei.pku.eu.cn

Received August 9, 2015; revised August 10, 2015.

Abstract For this special section on software systems, several research leaders in software systems, as guest editors for this special section, discuss important issues that will shape this field's future directions. The essays included in this roundtable article cover research opportunities and challenges for emerging software systems such as data processing programs (Xiangyu Zhang) and online services (Dongmei Zhang), with new directions of technologies such as unifications in software testing (Yves Le Traon), data-driven and evidence-based software engineering (Qing Wang), and dynamic analysis of multiple traces (Lu Zhang). — Tao Xie, Leading Editor of Special Section on Software System.

Keywords data processing program, software analytics, online service, software testing, data-driven software engineering, evidence-based software engineering

1 Engineering Stable Data Processing Programs (Xiangyu Zhang)

In this Big Data era, data processing is becoming one of the most prevalent computing tasks. In the mean time, errors pose a serious threat to output validity for modern data processing, which is often performed by computer programs. Raw inputs may be acquired by physical instruments that have precision limitations, leading to input errors. Parameters used in data processing may be provided by human scientists based on their experience, leading to uncertainty. Data may not be precisely represented due to the limited precision of the machine used, leading to representation errors. Once these errors get into computation, they may get propagated and magnified by the operations conducted, producing unreliable output. It is called the instability problem. Instability problems have substantial impact in various aspects. For instance, it could lead to bogus scientific findings that are costly down the road, inaccurate long-term weather forecast resulting in substantial lost, and inaccurate perception of battle field situation causing casualties. Traditionally, error analysis is conducted on mathematical models. However, modern data processing uses more complex models and relies on computers and programs, rendering mathematical analysis difficult. As such, developing stable data processing programs is becoming a prominent challenge for software engineers and software engineering researchers.

Instability problems have unique characteristics, compared to the traditional functional bugs. They cannot be completely evaded as such problems are usually

Regular Paper
Special Section on Software Systems
©2015 Springer Science + Business Media, LLC & Science Press, China

rooted at the underlying data processing algorithms or even the requirements. Using higher precision in computing or changing implementation may mitigate the problems, but cannot completely evade them. Instabilities are highly input dependent. They only occur at specific input ranges which are usually very very small, despite that the consequence of such problems may be devastating. In practice, people are willing to live with these unstable problems and the entailed risks. Some recent study has shown that even the widely used data processing programs in the SPEC-FP benchmark set have instability problems while people still feel comfortable using those programs due to the low probabilities of the instability problems.

In order to help engineers to build stable data processing programs, I foresee that our research community needs to address the following challenges.

• Develop cost-effective runtime predictors that can effectively predict if a specific execution is stable. In most cases, the predictor will report no instability has occurred and the outcome can be trusted. In the very rare scenarios where instability does occur, the predictor shall be able to report the output variations in the presence of errors.

• Develop testing tools that can identify the input ranges in which instabilities occurs. This would help users/developers to avoid such ranges or use a higher precision for such inputs.

• Develop techniques that can statically choose the right precision for a given data processing problem or use compiler to inject support for multiple precisions in the program such that the selection of the right decision can be performed at runtime.

• Develop verification techniques that prove small numerical functions are free from instability.

Errors in computing could cause severe problems in scientific discovery, economy, society, and military operations. However, people are lacking effective and efficient automated tools to help them address these problems. More importantly, due to the nature of these problems, I foresee many of them can be addressed by building upon existing software engineering techniques such as program analysis, testing, and verification.

2 Software Analytics for Online Services (Dongmei Zhang)

Software services are now widely available impacting various aspects of people's lives. Due to the enormous user base and highly frequent use, a huge wealth of different types of data are generated at every moment throughout the lifecycle of these services. Hidden in the data is information about the service quality, user experience, as well as the development dynamics. With various analytical and computing technologies, such as machine learning, data mining, information visualization, program analysis, and large-scale data computing, Software Analytics[1-2] enables software practitioners to perform effective and efficient data exploration and analysis, in order to obtain insightful and actionable information for data-driven tasks in engineering online services.

Data sources are vital to ensuring service quality and user experience. Service monitoring systems usually collect huge amount of runtime data including system states and service logs. User requests record how users use the services. In addition, the information on deployment and configuration as well as service incidents are also recorded. From the user perspective, customer profiles and their support history along with commerce data further enrich the data sources. The common data sources in online services are summarized in Table 1.

Table 1. Common Data Sources in Online Services

Category	Data Sources
System runtime	Performance counters, event logs, web logs, service component logs
Usage	User requests, user interaction logs
Deployment	Network topology, service topology, service configuration
Customer	Tenant profiles, user profiles, customer support
Commerce	Subscription information

In general, two types of data analysis pipelines, realtime and offline, are built up to serve different purposes of service quality management. The following challenges are often observed in building such analysis capabilities.

1) Service monitoring mainly relies on active monitoring that often has limited coverage;

2) Huge amounts of telemetry data from real users are under-utilized;

3) Diagnosis is mostly manual;

4) Knowledge about service, diagnosis, past incidents, etc. is scattered and not well organized and accumulated;

5) Customer support is not well connected to service quality management, resulting in unsatisfactory customer experience and high support cost;

6) The understanding on users and user behaviors is limited, which in turn limits the contribution to service quality, customer experience, and business success.

Two pillars need to be firmly established in order to address the aforementioned challenges. One is integration and the other is intelligence.

The integration pillar refers to storing different data sources in a well-designed infrastructure, in order to enable easy and well-controlled access as well as efficient computing. More importantly, the data sources should be properly linked to enable in-depth and cross-source analysis. Unified instrumentation, consistent schema, and data models should be used across different services to reduce friction in data collection, access, and analysis.

The intelligence pillar refers to the rich and deep analyses conducted on the data source. Moreover, knowledge bases should also be established and evolved in order to break the expert bottlenecks and speed up the data-to-insight process.

Based on the scenarios of quality management in different services, a common set of analysis problems can be identified that fall into three categories: anomaly detection, problem localization and diagnosis, and problem categorization. For each category of problems, different data analysis techniques can be developed and applied.

Anomaly detection is the problem of finding patterns in data that do not conform to a model representing the normal behavior of the data. It is in great need in service monitoring to detect unexpected behaviors, e.g., sudden change of KPIs, and rarely seen usage patterns.

Given the symptom of a service issue and a set of service monitoring data, problem localization helps scope down the service issue, e.g., to server nodes, service components, and certain log patterns. Problem diagnosis provides useful information for identifying the root cause of the underlying service issue. Both techniques are important for reducing MTTR (Mean-Time-To-Recovery) in incident management, as well as making root-cause analysis more effective in problem management.

In quality management of online services, it is common that many telemetry signals reflect the same or similar service issues. In order to correctly understand and prioritize the actual service issues, problem categorization techniques need to be used in practice to group the service issues properly. Categorization not only provides a basis for reporting and prioritization, but also helps make the diagnosis effort manageable. It enables product teams to answer questions like "how many service issues are found" and "how many of them are not resolved yet".

In recent years, increasing amount of research was published on data-driven quality management for online services, including data analysis techniques as well as experience reports on applying such techniques in real practice[3-8]. In the near future, software analytics for online services will continue to be an important focus for software engineering research and technology transfer.

3 Unifications in Software Testing (Yves Le Traon)

Fundamentally, software testing is about detecting a maximum of defects contained in a software product. Although historical data in the field has shown that software testing is a continuous battle, it also reveals that a substantial amount of knowledge has been accumulated since the early days of computer science. The question of interest today is: do we fully take benefit from this knowledge to build cheaper, safer and better software?

As a generic research domain, software testing appears to be doomed by the variety of options and constraints in the tested subjects. Indeed, software testing directions may be influenced by the nature and complexity of the software, from embedded or mobile to cloud software, from desktop or centralized to web or distributed software. Software testing is also applied differently according to the different stages in the software life cycle, the ever evolving technologies involved, the design settings and the programming languages. Considering these moving targets, it is quite remarkable to note the amount of achievements that have been produced in the last three decades, for instance, the definition and application of test coverage and adequacy criteria, automated test generation and execution environments, code-based test generation (e.g., concolic approaches), model-based testing (MBT), test qualification techniques such as mutation and so many other

relevant contributions. Among the other areas of computer science and engineering, software testing research is especially interrelated and interdependent with industry, some companies and their developers having contributed to major achievements in software testing. Unfortunately, all these positive remarks are counterbalanced by observing that a vast majority of the contributions have a focus limited to functional testing, with no definitive progress concerning the oracle question, which determines whether a result is correct, and the continuity from fault detection to repair.

The last decade has seen an interest in investigating more in-depth some of those directions, such as non- or extra-functional testing (security, performances, consumption), new design/programming paradigms (aspect-oriented programs, model-driven engineering, software product lines, distributed systems), and metrics and instruments for qualifying testing techniques (e.g., mutation analysis). All these research works still require a lot of efforts: security and privacy are at stake while scalability and performance remain paramount.

Building on this quick (and probably incomplete) summary of past research as a baseline, let me list four longer-term unifications that currently deserve the attention of researchers in the continuation of this past research.

• *Unification of the Process from Requirements to Testing.* The oracle problem is related to the identification of what we expect the software to do and not to do, according to diverse requirements. The translation of requirements and user-needs into executable test cases is key, and MBT as well as model-driven traceability are potential bootstrapping steps in that translation. The point is to be able to better analyze requirements and user needs for testing what must be tested according to those requirements. The entry point can be either explicit requirements, usually involving natural language, or implicit requirements. Implicit requirements may be defined as what the user really needs (and does not need) but that he/she has not expressed. For implicit requirements, analytics could be of great help to learn about software usability and actual usages. Why test parts that will never be used by anyone? How to test only what is required? The requirements are multiple since many diverse aspects are becoming mandatory to have a valid software: preserving security and privacy is important, being compliant with regulation is another requirement, scaling to stress conditions is becoming key and software not ensuring this requirement is a failing software that must be fixed.

• *Unification of the Process from Defect Detection to Repair, and from Repair to Suggestions for Improvement.* The point here is to consider software testing as a unified process from detection to fault localization and repair, and even to recommending improvements. Many studies are about data mining of real-program repositories, and as such are not directly related to the detection of defects, but instead to their localization, and repairing/fixing. The outcomes of these studies are not yet assembled with software testing approaches, enabling a continuous identification of a potential defect, its execution to exhibit it as an actual defect, and then the recommendation for a fix. A defect not only is the non-satisfaction of a requirement but also may be seen as an under-satisfaction of a set of requirements. Code smells, information leakages, and under-performance code, all these are defects that may be hunted and then lead to code modification for improving the final software. Along this line comes all the research related to multi-objective optimization of a software, multi-dimensional fixing (e.g., blocking an access to a resource while keeping the functionality unchanged).

• *Unification of Search-Based Software Engineering (SBSE) Techniques and Deterministic Approaches for Handling Quasi-Infinite Input Spaces.* Software test selection/generation is about finding the best input data to reveal defects within a very large, quasi-infinite input space. Exploring such quasi-infinite input domains is challenging, especially with respect to multiple and potentially conflicting testing objectives, functional and non-functional ones. An example of such conflicting objectives comes with the emergence of highly configurable systems. Software development is increasingly moving from the production of a single, yet configurable, software to the development of families of software products (Software Product Lines). The configuration space is growing with the number of software features that can be combined to build tailored software products. For such systems, the challenge has been addressed by combining search-based techniques and constraint solving with very promising results. In this respect, this unification of SBSE with formal techniques, when applied efficiently, may lead to devise hyper-heuristics that generalize to any system with well-specified input domains.

• *Unification of Dynamic and Static Techniques for Program Analysis.* Program static and dynamic analyses are complementary tools for detecting, localizing and fixing defects. Powerful tools are already available to perform complex analyses, for instance, bytecode security analysis of Android apps. However, both static and dynamic analysis face intrinsic limitations, static analysis may lead to over-approximations while dynamic analysis (dynamic testing) may miss some cases and lead to under-approximations in program analysis (e.g., detecting leakages in an app). It would be a great tool, the one that would smartly combine code static analysis with software testing.

All these unification trends are related to the consolidation and continuation of well-established research directions. Looking ahead to the new technological challenges, it is hard to foresee what may and will be achieved, and thus to what extent this may modify software testing. Among many ones, let me mention in bulk: the analytics tools and recommendation systems, the emergence of new non ACID databases, the trivialization of very-large scale distributed systems (IoT, Cyber-Physical Systems), software dematerialization and dynamic relocation in the "cloud", and the development of more and more volatile markets with the acceleration of time-to-market-to-trash for services and software. The next years will be overwhelming, by forcing the testing processes to evolve, the testing techniques to scale and adapt to fast validation, and maybe weakening the notion of correctness to deal with uncertainty and approximation. Even basic notions such as defect/fault/error may be completely redefined with these new perspectives. There will be no rest for the testing community, but a huge domain to continue exploring, and consequently many lessons to learn from new unforeseen errors.

4 Data-Driven and Evidence-Based Software Engineering (Qing Wang)

Nowadays, software systems are being developed and applied in an increasingly wide scope. There are various emerging types of software systems such as Social Software Service Systems and Software Ecosystems. On the other hand, open and autonomy software environments have changed the software development methodologies and processes. With these recent trends, there are three main important challenges to be addressed by researchers and practitioners.

1) How to process, understand, and leverage the data produced in the software development process? There are many data, such as software documents, process management data, user feedback and comments, bug reports, issue and feature requests. These data exist for some time but most of these data have not been leveraged well. Especially in open development environments, massive data are delivered by various stakeholders voluntarily with various quality levels. It is very important to find valuable information out of these data to assist productivity and quality improvement.

2) How to manage the quality of software? Traditional software project management based on contracts has been increasingly replaced by community-based project management, such as project management for open source and crowdsourcing. Most of the developers and testers of a project join the project based on their interest and willingness, and contribute their time and effort without being constrained. In addition, massive users give feedback of their feelings and preferences freely. It has been increasingly challenging to manage the quality of software such as collecting, elaborating, and formalizing the evidence for the quality of software.

3) How to stimulate collective intelligence of the software community? In the increasingly open, autonomy software community, the cooperation and the competition coexist. The data and knowledge of group wisdom should spring collective intelligence in some way. It is important to cultivate and encourage positive collective intelligence to build an active, booming, and healthy community and to evolve software effectively and efficiently.

5 Dynamic Analysis of Multiple Traces (Lu Zhang)

Traditionally, dynamic analysis techniques focus on analyzing a single execution trace to obtain various properties specific to the execution trace. Since one single execution trace may not sufficiently demonstrate some important properties of the software under analysis, there have been increasing demands in software engineering to analyze multiple execution traces. There are two paradigms for dynamically analyzing multiple traces.

• *Batch Analysis.* In this paradigm, the analyzer faces a set of traces in the first place. The aim of the analyzer is to summarize or synthesize properties on

the basis of analyzing each trace. One typical example is dynamic specification mining[9].

• *One-By-One Analysis*. In this paradigm, the analyzer starts with one single trace. Based on analyzing this trace, the analyzer determines which trace should be analyzed next (typically by determining the input to generate the next trace). One typical example is dynamic symbolic execution[10].

Dynamic analysis of multiple traces typically requires combining program analysis with intelligent data processing (e.g., data mining and artificial intelligence). In the first paradigm, the summarization process mainly relies on some intelligent mining technique. In the second paradigm, the determination of the next trace to analyze is actually a decision making process.

Obviously, how to combine such two types of techniques is important. In existing research, it is common to use coarse combinations. That is to say, it is typical to use program analysis and intelligent data processing as distinct passes, although there may be several iterations, each containing both passes.

An interesting new way of combining the two kinds of techniques is search-based dynamic analysis, which adopts search-based optimization as both the intelligent data processing technique and the combination framework. With a meta-heuristic search technique, the execution of the software under analysis can be treated as an integral step of the search process. This way of combination is finer because the intelligent data processing here is no longer a pass independent from the passes of analyzing one trace. Below, I list two examples, each belonging to one paradigm: the first piece of research[11] uses search-based optimization to infer a special kind of specifications (i.e., metamorphic relations); while the second piece of research[12] uses search-based optimization to find inaccuracies in numerical programs.

References

[1] Zhang D, Dang Y, Lou J G, Han S, Zhang H, Xie T. Software analytics as a learning case in practice: Approaches and experiences. In *Proc. International Workshop on Machine Learning Technologies in Software Engineering* (*MALETS 2011*), Nov. 2011, pp.55-58.

[2] Zhang D, Han S, Dang Y, Lou J, Zhang H, Xie T. Software analytics in practice. *IEEE Software*, 2013, 30(5): 30-37.

[3] Fu Q, Lou J G, Wang Y, Li J. Execution anomaly detection in distributed systems through unstructured log analysis. In *Proc. the 9th IEEE International Conference on Data Mining* (*ICDM 2009*), Dec. 2009, pp.149-158.

[4] Lou J G, Lin Q, Ding R, Fu Q, Zhang D, Xie T. Software analytics for incident management of online services: An experience report. In *Proc. the 28th IEEE/ACM International Conference on Automated Software Engineering* (*ASE 2013*), *Experience Papers*, Nov. 2013, pp.475-485.

[5] Ding R, Fu Q, Lou J, Lin Q, Zhang D, Xie T. Mining historical issue repositories to heal large-scale online service systems. In *Proc. the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (*DSN 2014*), June 2014, pp.311-322.

[6] Fu Q, Zhu J, Hu W, Lou J, Ding R, Lin Q, Zhang D, Xie T. Where do developers log? An empirical study on logging practices in industry. In *Companion Proc. the 36th International Conference on Software Engineering* (*ICSE 2014*), May 31-June 7, 2014, pp.24-33.

[7] Shang W, Jiang Z M, Hemmati H, Adams B, Hassan A E, Marin P. Assisting developers of big data analytics applications when deploying on Hadoop clouds. In *Proc. the 35th International Conference on Software Engineering*, May 2013, pp.402-411.

[8] Malik H, Hemmati H, Hassan A E. Automatic detection of performance deviations in the load testing of Large Scale Systems. In *Proc. the 35th International Conference on Software Engineering*, May 2013, pp.1012-1021.

[9] Yang J, Evans D, Bhardwaj D, Bhat T, Das M. Perracotta: Mining temporal API rules from imperfect traces. In *Proc. International Conference on Software Engineering*, May 2006, pp.282-291.

[10] Godefroid P, Klarlund N, Sen K. DART: Directed automated random testing. In *Proc. International Conference on Programming Language Design and Implementation*, June 2005, pp.213-223.

[11] Zhang J, Chen J, Hao D, Xiong Y, Xie B, Zhang L, Mei H. Search-based inference of polynomial metamorphic relations. In *Proc. International Conference on Automated Software Engineering*, Sept. 2014, pp.701–712.

[12] Zou D, Wang R, Xiong Y, Zhang L, Su Z, Mei H. A genetic algorithm for detecting significant floating-point inaccuracies. In *Proc. International Conference on Software Engineering*, May 2015, pp.529-539.

Xiangyu Zhang is an associate professor at Purdue University, West Lafayette. He received his Ph.D. degree in computer science from the University of Arizona, Tucson, in 2006, and his M.S. and B.S. degrees in computer science from University of Science and Technology of China, Hefei. His research interest lies in dynamic and static program analysis and their applications in debugging, forensic analysis, and data processing. He is currently a Purdue University Scholar. He has received the 2006 ACM SIGPLAN Distinguished Doctoral Dissertation Award, NSF Career Award, and a few Best Paper Awards in top conferences.

Dongmei Zhang is a principal researcher and research manager at the Software Analytics group of Microsoft Research Asia (MSRA), Beijing. Her research interests include data-driven software analysis, machine learning, information visualization and large-scale computing platform. She founded the Software Analytics group at MSRA in 2009. Since then, she has been leading the group to research software analytics technologies. Her group collaborates closely with multiple product teams in Microsoft, and has developed and deployed software analytics tools that have created high business impact.

Yves Le Traon is a professor at University of Luxembourg, in the Faculty of Science, Technology and Communication (FSTC). His domains of expertise are related to software engineering and software security, with a focus on software testing and model-driven engineering. He received his engineering degree and his Ph.D. degree in computer science at the "Institut National Polytechnique" in Grenoble, France, in 1997. From 1998 to 2004, he was an associate professor at the University of Rennes, in Brittany, France. During this period, Professor Le Traon studied design for testability techniques, validation and diagnosis of object-oriented programs and component-based systems. From 2004 to 2006, he was an expert in model-driven architecture and validation in the EXA team (Requirements Engineering and Applications) at "France Télécom R&D" company. In 2006, he became professor at Telecom Bretagne (Ecole Nationale des Télécommunications de Bretagne) where he pioneered the application of testing for security assessment of web-applications, P2P systems and the promotion of intrusion detection systems using contract-based techniques. He is currently the head of the Computer Science Research Unit at University of Luxembourg. He is a member of the Interdisciplinary Centre for Security, Reliability and Trust (SnT), where he leads the research group SERVAL (SEcurity Reasoning and VALidation). His research interests include software testing, model-driven engineering, model based testing, evolutionary algorithms, software security, security policies and Android security. The current key-topics he explores are related to Internet of things (IoT), Big Data (stress testing, multi-objective optimization, models@run.time), and mobile security and reliability. He is author of more than 140 publications in international peer-reviewed conferences and journals.

Qing Wang is a professor of the Institute of Software, Chinese Academy of Sciences, Beijing. She is the director of the Lab for Internet Software Technologies. She has intensive research and industry experience in the area of software process, including software process technologies and quality assurance and requirement engineering. She is a member of Cloud Computing Experts Association under Chinese Institute of Electronics and a Lead Appraiser of SEI CMMI SCAMPI. Qing Wang has led and is leading many important domestic and international cooperative projects. She has won various kinds of awards such as the National Award for Science and Technology Progress. She also is a general co-chair of ESEIW 2015, PC member of some academy conferences and program co-chair of SPW/ProSim 2006, ICSP 2007 and ICSP 2008.

Lu Zhang is a professor at the School of Electronics Engineering and Computer Science, Peking University, Beijing. He received his B.S. and Ph.D. degrees in computer science from Peking University in 1995 and 2000 respectively. He was a postdoctoral researcher in Oxford Brookes University and University of Liverpool, UK. He joined the School of Electronics Engineering and Computer Science, Peking University in 2003. He has served on the editorial boards of two international journals: Journal of Software Maintenance and Evolution: Research and Practice and Software Testing, Verification and Reliability. He also served on the program committees of many major conferences, including FSE, ISSTA, ASE, and ICSM. His papers have won the Best Paper Awards of ASE and APSEC, and the ACM SIGSOFT Distinguished Paper Awards of ICSE and ASE. He is supported by the Outstanding Youth Science Fund of the Natural Science Foundation of China. His current research interests include software testing, software analysis, program comprehension, software maintenance, software reuse, and service computing.