Roundtable

The Future of Software Engineering for Internet Computing

Jian Lü, Nanjing University

David S. Rosenblum, National University of Singapore

Tevfik Bultan, University of California, Santa Barbara

Valerie Issarny, Inria Paris-Rocquencourt

Schahram Dustdar, Vienna University of Technology

Margaret-Anne Storey, University of Victoria

Dongmei Zhang, Microsoft Research, China

FOR THIS SPECIAL ISSUE, seven research leaders in software engineering for Internet computing discuss important issues that will shape this field's future. The essays cover opportunities and challenges for the shifting software paradigm (Jian Lü); stepping outside the comfort zone to revisit issues such as software correctness (David S. Rosenblum); improving Internet software dependability and programmability (Tevfik Bultan); addressing software engineering issues for the Internet of Things (Valerie Issarny); exploring the relationships among the Internet of Things, people, and software services (Schahram Dustdar); supporting a participatory culture of software development (Margaret-Anne Storey); and rethinking logging in online services (Dongmei Zhang). Enjoy! —Antonia Bertolino, M. Brian Blake, Pankaj Mehra, Hong Mei, and Tao Xie, guest editors

Internetware: Shifting the Software Paradigm toward the Internet

Jian Lü

The Internet, not only of computers but also of things and human users, has been rapidly and profoundly changing how we construct, deploy, and use software applications. To achieve their application goals, software systems on this Internet platform need to coordinate autonomous third-party services and resources, adapt to constant changes in their environment and the requirements they must satisfy, and continuously maintain a quality of service (QoS) that satisfies users. So, Internet computing poses the significant challenge of how to help software engineers manage these new dimensions of complexity.

Conventional software paradigms, such as structured, object-oriented, and component-based methods, are inadequate here. We need a paradigm shift to comprehensively support Internet-computing applications that are autonomous, cooperative, situational, evolvable, emergent, and trustworthy.[1]

To this end, Chinese software

0740-7459/15/$31.00 © 2015 IEEE

engineering researchers have conceptualized and developed the *Internetware* paradigm, supported by the National Basic Research Program of China. Internetware follows two principles:[1,2]

- *Coordination-centric architecture*. Programmable connectors and run-time software architecture models facilitate flexible but disciplined coordination of autonomous entities.
- *Environment-driven adaptation*. Run-time environment models, built as integral parts of Internetware systems, direct the probing and interpretation of the real environment and drive system adaptation when necessary.

Internetware also aims to provide comprehensive assurance of system quality. First, besides traditional quality factors such as correctness, performance, and reliability, it emphasizes quality factors related to the user experience, such as energy efficiency, privacy, and user-friendliness. Second, instead of setting a universal, fixed QoS target, it considers user-specific, dynamically tunable targets by taking into account user preferences and feedback. Finally, to ensure trustworthiness in the open Internet, it not only analyzes software artifacts' quality but also manages the trust relationships between the subjects owning the artifacts.

Internetware researchers have made progress in architecture models, middleware frameworks, and development tools, along with system prototypes and case studies.[1,2] However, much research remains to realize the full vision of Internetware. We need more research on systematic software engineering methodologies and enabling techniques that will eventually shift the software paradigm toward Internet computing.

References

1. J. Lü et al., "Internetware: A Shift of Software Paradigm," *Proc. 1st Asia-Pacific Symp. Internetware* (Internetware 09), 2009, article 7.
2. H. Mei, G. Huang, and T. Xie, "Internetware: A Software Paradigm for Internet Computing," *Computer*, vol. 45, no. 6, 2012, pp. 26–31.

JIAN LÜ is a professor of computer science in Nanjing University's State Key Laboratory for Novel Software Technology. Contact him at lj@nju.edu.cn.

Stepping outside Our Comfort Zone

David S. Rosenblum

The Internet poses many challenges to software engineering; these challenges are becoming particularly acute with the emergence of computing paradigms that exploit the Internet in new ways. The emergence—and convergence—of mobile and ubiquitous computing, sensor networks, cyber-physical systems, data analytics, and the Internet of Things is pulling software engineering further and further from the comfort zone of principles and techniques that have prevailed for many decades.

Looking through the prism of software engineering research, you get little sense of the enormous changes taking place and their accompanying challenges. Software engineering researchers largely still view software and its engineering precepts and solutions much as they always have. Yet even something as basic as the concept of correct behavior is being undermined by the changing nature of software.

Consider the use of machine learning in software engineering. Machine-learning research has produced a rich, diverse collection of approaches and algorithms for performing automated statistical classification of data processed by software at run time. Machine learning has matured to the point at which it's being commoditized in powerful tools such as Weka (Waikato Environment for Knowledge Analysis) and contributing powerful functionality to a wide variety of real-world applications.

At the algorithmic level, it's challenging enough to determine whether some implementation of a machine-learning algorithm is producing the outputs it should, given the inputs it receives. But even if we're willing to accept at face value the correctness of the algorithm's implementation, we must still deal with the consequences of the statistical nature of the algorithm itself, whose classification ability typically is less than 100 percent precise. A rose by any other name is still a rose, but a machine-learning classifier might occasionally say it's a daisy. That might be the best (the algorithm of) the classifier can do. So, its output occasionally might be incorrect, but that doesn't mean the classifier itself is incorrect in the traditional software engineering sense. If we embed this classifier within a larger application, and the application occasionally produces incorrect output, how can we tell whether this is due to the classifier's imprecision or some fixable bug?

This problem, which Sebastian Elbaum and I are studying,[1] is but one of the many ways the Internet threatens some of the bedrock notions of software engineering. We

as software engineering researchers have only just begun to understand and appreciate these challenges' implications. It's time for us to step outside our comfort zone and address them.

Reference

1. S. Elbaum and D.S. Rosenblum, "Known Unknowns: Testing in the Presence of Uncertainty," *Proc. 22nd ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, 2014; pp. 833–836.

DAVID S. ROSENBLUM is a professor of computer science in the National University of Singapore's School of Computing. Contact him at david@comp.nus.edu.sg.

Internet Software's Dependability and Programmability

Tevfik Bultan

Internet computing has been evolving rapidly as we keep finding new ways to build and use network-connected computing devices (NCCDs). Programmable phones and tablets have overtaken PCs. In a few years, programmable glasses, watches, or cars might be the most common NCCDs.

In concert with NCCDs' increasing diversity, we see a uniform approach to software application development based on the software-as-a-service paradigm and the multitiered architecture. Nowadays, most applications are software services hosted on clouds and accessed by thin clients that execute on NCCDs.

In this modern computing landscape, software engineering research for Internet computing faces two related challenges: improving dependability and programmability.[1]

The necessity to address the first challenge was made painfully clear last year by the problems that HealthCare.gov encountered. Modern software applications are distributed systems comprising multiple components executing on multiple machines and interacting with each other by exchanging messages over the network. Given these systems' complexity, you could argue that it isn't surprising to have many bugs and security vulnerabilities. However, if we want to continue making software systems a critical part of every human endeavor, we need to develop software engineering techniques that are better at establishing dependability.

> We need to develop software engineering techniques that are better at establishing dependability.

The second significant challenge is, how can we make software development easier, so that even end users can participate in basic application development or customization? This challenge is critical because the explosive increase in software applications will likely continue as novel NCCDs are introduced and people find new uses for them. I don't think we'll be able to produce enough computer scientists to meet the demand for application development. We'll need to provide mechanisms that let end users create applications or customize existing ones.

The following research directions can help us address these two fundamental challenges.

Current software development practices based on general-purpose programming languages rely on the talents of individual developers. Success in state-of-the-art software development isn't easily repeatable and scalable. We must develop higher-level abstractions for modern software application development. We need to think about what's the best way to specify a software service's behavior rather than focus on pieces of the problem such as client-side versus server-side programming. We should develop high-level modeling languages that can be automatically compiled to executable code. Let the compiler decide where to run the code (on the client or server side). We should be thinking about what's the best abstraction for specifying the application's behavior.

In addition, we should devise modular software development techniques that let us write an application as a combination of multiple policies, each specifying a certain aspect of the application behavior. For example, you could think of a modern Web application as a combination of several policies, such as

- a navigation policy identifying the flow between different views of the application,[2]
- a data update policy specifying how the data in the persistent storage is updated,[3]
- a UI policy identifying how to present the views to the user,
- an access control policy identifying which data a user has access to,[4]

- an authentication policy specifying how users are identified, and
- an input validation and sanitization policy specifying how the user input is accepted.[5]

Existing Web application development frameworks somewhat support modularization along these lines. However, these policies are written using general-purpose programming languages, which make their analysis and verification difficult. We should develop high-level domain-specific languages that facilitate modular application development with formal guarantees for separation of concerns. (For example, an access control policy can't be violated by the navigation policy.)

Finally, improving dependability and programmability require increasing the level of automation in application development. We can achieve such automation by increasing the level of abstraction and modularity. So, contributions in the two research directions I just outlined would facilitate better automated techniques for code analysis, synthesis, verification, and repair.

References

1. T. Bultan, "Software for Everyone by Everyone," *Proc. FSE/SDP Workshop Future of Software Eng. Research* (FoSER 10), 2010, pp. 69–74.
2. S. Hallé et al., "Eliminating Navigation Errors in Web Applications via Model Checking and Runtime Enforcement of Navigation State Machines," *Proc. 25th IEEE/ACM Int'l Conf. Automated Software Eng.* (ASE 10), 2010, pp. 235–244.
3. I. Bocić and T. Bultan, "Inductive Verification of Data Model Invariants for Web Applications," *Proc. 36th Int'l Conf. Software Eng.* (ICSE 14), 2014, pp. 620–631.
4. F. Sun, L. Xu, and Z. Su, "Static Detection of Access Control Vulnerabilities in Web Applications," *Proc. 20th USENIX Conf. Security* (SEC 11), 2011, p. 11.
5. M. Alkhalaf et al., "ViewPoints: Differential String Analysis for Discovering Client- and Server-Side Input Validation Inconsistencies," *Proc. 2012 Int'l Symp. Software Testing and Analysis* (ISSTA 12), 2012, pp. 56–66.

TEVFIK BULTAN is a professor in the University of California, Santa Barbara's Department of Computer Science. Contact him at bultan@cs.ucsb.edu.

Software Engineering for the Internet of Things

Valerie Issarny

The vision of pervasive computing, since its introduction by Mark Weiser in the early '90s and throughout its redefinition along the years, hasn't evolved much. Furthermore, this is clearly no longer a vision; nevertheless, in many of its aspects, it still challenges the computer science communities at large. The software engineering community is no exception, and the challenge is even more so with the advent of the Internet of Things (IoT). The IoT promises to blend the physical and virtual worlds, hence introducing a vast amount of knowledge into our now largely pervasive, distributed software systems.

With the IoT, sensing and actuation are called on to become a utility. To make the IoT a reality, research must solve these challenges:[1]

- enable massive scaling, considering the foreseen trillions of things;
- devise system architectures that cope with the networking environment's high heterogeneity and dynamics;
- extract knowledge from the sensed raw data;
- support an open environment, whereas sensor-based systems so far have been mostly closed domain-specific systems;
- guarantee robustness of the enacted systems despite the mostly unknown networking environment;
- enforce security and privacy; and
- allow the synergistic operation of humans and things.

Addressing these challenges will affect the development of the supporting software systems.

To meet these challenges, our group at Inria Paris-Rocquencourt has been studying extensively how to leverage but also revisit the traditional service-oriented-architecture paradigms.[2] Indeed, service orientation combined with semantic technologies allows dealing with the IoT's dynamics and heterogeneity. Still, the massive scale of the network of things calls for completely new protocols to discover, access, and coordinate things, including mobile things.[3] In addition, development environments for applications to be deployed over the IoT remain pretty much an open issue in light of the requirements for openness and robustness. Similarly, we need software tools that enable reasoning about applications' security and privacy. Finally, software tools to process and analyze the big data made available by the IoT have yet to be devised.

A key issue underlying these challenges is the traditional centralized architecture versus a distributed architecture. Distribution is crucial to meeting these challenges. Moreover, solutions must be probabilistic, given the uncertainty of the target networking environments.

References

1. J.A. Stankovic, "Research Directions for the Internet of Things," *IEEE Internet of Things J*, vol. 1, no 1, 2014, pp. 3–9.
2. T. Teixeira et al., "Service Oriented Middleware for the Internet of Things: A Perspective," *Towards a Service-Based Internet*, LNCS 6994, Elsevier, 2011, pp. 220–229.
3. S. Hachem, A. Pathak, and V. Issarny, "Service-Oriented Middleware for the Mobile Internet of Things: A Scalable Solution," to be published in *Proc. 2014 IEEE Global Communications Conf.* (GLOBECOMM 14), 2014.

VALERIE ISSARNY is a senior research scientist at Inria Paris-Rocquencourt. Contact her at valerie.issarny@inria.fr.

The Internet of Things, People, and Software Services

Schahram Dustdar

The Internet has undergone an essential transformation and has been a stunning success. It changed from being a network of networks enabling access to remote machines to a network of content, applications, people, and (software) services, thereby weaving itself into the fabric of today's global and interconnected society. We can safely claim that today's use of the Internet constantly transforms how people, businesses, and society as a whole operate.

The interactions we witness are such that some claim they defy the laws of behavioral physics because they're built by autonomously interacting people who are often unpaid and intrinsically motivated. Assumptions about interaction models and patterns (between humans, systems, processes, and organizations) are seriously challenged. Novel foundational technologies and methods need proper attention from science, particularly computer science and information systems.

Future Internet (FI) research's goal therefore must be to provide the infrastructure (networks and services) and means to deal with the changing novel requirements of today's society. Doing this will pave the way to the convergence of application-specific networks, supporting the Internet of Services (IoS), Internet of Things (IoT), and Internet of Content (IoC) in a homogeneous network.

> With the Internet of Things, sensing and actuation are called on to become a utility.

Research has to provide the modern algorithmic, engineering, and methodological foundations for dynamic aspects, which are essential for building FI computing systems. In many scientific disciplines, as in most political matters today, we face grand challenges that increasingly require deep collaboration among scientists, government agencies, political decision makers, and the general public. Such challenges require dynamic infrastructures supporting novel ways of collaboration, coordination, and communication between social and technical subsystems.

One hope is that the FI enables the construction and execution of such highly dynamic novel infrastructures based on the IoS, IoT, and IoC. These main building blocks of FI systems will thus include both executable software-based services and human-provided services. In the latter case, the provider of some functionality won't be a machine (in the form of a software executable) but a human being. This person will (through machine-processable interfaces) provide services and (automated) interactions, called *service ensembles*. The building blocks of service ensembles are active, which renders their composition a challenge.

A service ensemble consists of humans and software services. Interaction in service ensembles includes people communicating and coordinating with other people, people using (human- or software-based) services, and services invoking other services. Service ensembles aren't a purely social system because services greatly affect how people interact. Services determine how people can coordinate, communicate, and carry out their joint work. Neither is a service ensemble a purely technical system. The social structure greatly influences the required service capabilities. Groups that exhibit great trust among members want to collaborate more freely and with less structure than groups that follow a rigid organizational structure.

We must work on the scientific foundations for building and testing FI systems that allow construction and deployment of service ensembles.

SCHAHRAM DUSTDAR is a professor in the Vienna University of Technology's Faculty of Informatics. Contact him at dustdar@dsg.tuwien.ac.at.

Supporting a Participatory Culture of Software Development

Margaret-Anne Storey

Over the past few decades, software development has transitioned from a predominantly solo activity of developing standalone programs to a highly distributed, collaborative approach that depends on or contributes to large, complex software ecosystems. The distributed and collaborative nature of software development continues to grow as new Web-based tools are proposed and adopted. These tools offer social networking features (for example, watching and following other developers or projects) and lightweight, transparent channels for knowledge sharing. Such features facilitate the emergence of a participatory development culture,[1] in which developers are keen to learn from and cocreate with others.[2]

This participatory culture emerged not just because of social tools but also owing to the advancement of remote systems and the wider availability of the Internet in the early '80s. Development communities naturally formed around free and open source projects, with communication tools such as email, version control, and Usenet playing an important role in community formation. This participatory culture isn't isolated to open source projects; it's also clearly evident in industrial distributed and global software development projects. Nowadays, we see social coding, microblogging, social news sites, cloud-based development tools, and question-and-answer websites playing an essential role across many development contexts. The participatory nature of software development is embedded in and facilitated by an ecosystem of tools, developers, and content. The continued adoption of social systems and online development tools leads to three major trends.

First, we see the emergence of *social developers* who are passionate about contributing to community resources and who care deeply about what others think of their contributions. They nurture and use their social networks to stay up to date with technological changes, to broaden their skills and manage their own identity. Furthermore, their development tasks shift from writing new code to reusing or mashing up existing solutions. So, their success and effectiveness rely not only on their technical skills but also on how they can use their social networks to find, curate, and share important information.

> The use of the Internet to host community projects and tools across diverse domains broadens participation.

Second, the Internet as a hosting platform and environment for software development increases the emphasis on data over code in software engineering processes. Continuous release cycles, large-scale testing in the wild, user feedback through social media, and operational data from distributed development are just some of the data resources that can be analyzed and visualized to improve software quality, the user experience, and developer productivity.[3] The ability to continuously analyze and visualize real-time information plays an important feedback role in the participatory-development culture.

Third, the use of the Internet to host community projects and tools across diverse domains broadens participation (for example, to scientists and other end-user programmers). Such participation will likely expand further owing to the ubiquitous nature of computation that's visible across the Internet of Things. The Internet together with social tools supports community-authored and community-curated resources that help attract and retain those participants. However, the transparency these environments afford might also lead to participation barriers that shouldn't be ignored.

Finally, software developers are sometimes called the "prototype knowledge workers of tomorrow."[4] Developers are the creators or early adopters of new technologies that knowledge workers (for example, in healthcare, the sciences, or journalism) might rely on in the future. So, the impact of understanding the challenges of and opportunities from adopting and using social tools could reach across many knowledge domains.

References

1. H. Jenkins et al., *Confronting the Challenges of Participatory Culture: Media Education for the 21st Century*, MIT Press, 2006.
2. M.-A. Storey et al., "The (R) Evolution of Social Media in Software Engineering," *Proc. the Future of Software Eng.* (FOSE 14), 2014, pp. 100–116.

3. D. Zhang et al., "Software Analytics in Practice," *IEEE Software*, vol. 30, no. 5, 2013, pp. 30–37.
4. A. Kelly, *Changing Software Development: Learning to Become Agile*, John Wiley & Sons, 2008.

MARGARET-ANNE STOREY is a professor in the University of Victoria's Department of Computer Science. Contact her at mstorey@uvic.ca.

Rethinking Logging in Online Services

Dongmei Zhang

Logging is important for recording program execution process and debugging problems that occur during execution. It's widely used in in-house software development and in telemetry that collects program run-time information in the large. In the Internet computing era, logging has become even more critical in the quality management of online services because it's almost the only feasible diagnosis mechanism for large-scale distributed service systems. Although logging's value is indisputable, it faces new challenges in the context of online services.

The first challenge is cost. Owing to the massive user base and enormous volume of transactions served by online services, the volume of logs will increase significantly, resulting in huge storage and processing costs. Additionally, a huge number of logs often poses serious challenges to problem diagnosis. For example, many logs are often irrelevant to the problem under investigation, thus making diagnosis like finding a needle in a haystack.

The second challenge is how to control logging quality. Logging quality deals mainly with two issues. One is to detect and avoid logging incorrect information—data bugs. Data bugs are more likely to occur in a fast-paced dynamic development environment with many engineers. Flexible logging schemas, frequent code changes, code maintenance, and so on can cause data bugs.

The other logging-quality issue relates to effectiveness and efficiency. Insufficient logging might impact the logs' effectiveness because it might miss run-time information needed for postmortem analysis. Excessive logging might impact the logs' efficiency because the extra logs might incur a prohibitive cost at run time and in offline storage and processing.

Finally, the huge quantity of logs demands scalable, effective analysis techniques and tools to help engineers gain insights into their service systems' quality. On one hand, this lets engineers diagnose and resolve service problems as quickly as possible to reduce the mean time to recovery. On the other hand, engineers can use the insights to proactively detect and fix hidden problems in the systems and to enhance the monitoring mechanism accordingly.

It's great to see the research conducted and published on logging over the past few years.[1–3] I hope to see more researchers and practitioners rethink logging in the context of online services, and tackle the aforementioned challenges with breakthroughs in both research and practice.

References

1. W. Xu et al., "Large-Scale System Problem Detection by Mining Console Logs," *Proc. 22nd ACM Symp. Operating Systems Principles* (SOSP 09), 2009, pp. 117–132.
2. D. Yuan et al. "Be Conservative: Enhancing Failure Diagnosis with Proactive Logging," *Proc. 10th USENIX Conf. Operating Systems Design and Implementation* (OSDI 12), 2012, pp. 293–306.
3. Q. Fu et al., "Where Do Developers Log? An Empirical Study on Logging Practices in Industry," *Companion Proc. 36th Int'l Conf. Software Eng.*, 2014, pp. 24–33.

DONGMEI ZHANG is a principal researcher at Microsoft Research, China. Contact her at dongmeiz@microsoft.com.