Distributed Teams, Developer Participation, and More

Jeffrey C. Carver, Henry Muccini, and Aiko Yamashita

IN THIS ISSUE, we report on two talks from the First International Workshop on Collaborative Modeling in MDE (model-driven engineering) and three papers from the 23rd International Conference on Software Analysis, Evolution, and Reengineering. Feedback and suggestions are welcome. In addition, if you try or adopt any of the practices mentioned in the column, please send Jeffrey Carver and the paper authors a note about your experiences.

Model-Driven Engineering for Distributed Teams

In the talk “Scaling Up MDE to Support Large Geographically Distributed Teams—an Experience Report,” Vinay Kulkarni related his experiences applying MDE principles at Tata for more than 20 years. These experiences covered more than 70 very large applications, each with several million LOC, developed by large geodistributed and heterogeneous teams of more than 100 members.

Tata needed to ensure consistency between the various submodels of the global system model. To allow multiple stakeholders, who have different concerns, to work on a common model, Tata developed a centralized multiuser repository. It also developed a novel component abstraction to support the notions of private and public workspaces. To scale up MDE to support large and geographically distributed teams, Tata replicated the repository at each site and realized a protocol for synchronizing them.

Kulkarni discussed different project aspects (for example, large apps, continuous evolution, and large heterogeneous and distributed teams) and goals (for example, better coordination, agility, and economics). He explained how to achieve those goals through mechanisms such as workspace and access control, versioning and configuration management, pattern-based diff-n-merge, change models, and variability modeling. You can access the presentation slides at bit.ly/PD_May17_1.

A Collaborative Modeling Tool

In the talk “MDE Collaboration: Temporality and Ergonomy in the Cloud, the GenMyModel Solution,” Vincent Aranega presented the principles and characteristics of GenMyModel (www.genmymodel.com), a collaborative modeling-support tool that focuses on conflict management and change awareness. The collaboration model stresses concurrency, especially related to the interaction between many online model changes. Using this concurrency concept, Aranega explained how collaborative modeling is a matter of conducting the present while always keeping track
of the past and, somehow, dealing with the future.

GenMyModel supports collaborative modeling through a mutual-exclusion mechanism over a star topology on which the server conducts each change. The tool uses the Eclipse Modeling Framework (EMF) and GWT on the client side and a Tomcat Server and EMF on the server side. The mechanism implements atomic modifications that are used for building complex modifications.

Aranega also covered issues such as secure access and modifications, performance due to network latencies and server loads, and costs. You can access the presentation slides at bit.ly/PD_May17_2.

**Forking and Developer Participation**

"Forking and the Sustainability of the Developer Community Participation—an Empirical Investigation on Outcomes and Reasons," by Ayushi Rastogi and Nachiappan Nagappan, tells of a study in which forking, in which a new project emerges from an existing one, had mixed effects on the developer community’s participation.\(^3\)

For the original project, forking could be positive—indicating a healthy project—or negative—by drawing developers away.

Of the 2,217 GitHub projects that Rastogi and Nagappan studied, 18 percent experienced decreased developer community participation, 29 percent experienced increased participation, and 53 percent experienced unchanged participation. This result was mediated by the existing and new projects’ characteristics. Existing projects’ maturity increased forking while reducing the chances of decreased participation. More popular projects were less likely to be forked and less likely to see decreased developer community participation. When a large popular project forked into a popular new project, participation decreased. A large community resulted in decreased participation for small projects but increased participation for large projects. You can access this paper at bit.ly/PD_May17_3.

**Casual Contributors**

"More Common Than You Think: An In-Depth Study of Casual Contributors," by Gustavo Pinto and his colleagues, describes the characteristics of casual contributors—those who performed at most one commit to a FLOSS (free/libre and open source software) project—and their contributions.\(^4\) Analysis of 275 of the most popular projects in GitHub (determined on the basis of the stars awarded them) revealed that although almost half of the contributors were casual, they accounted for less than 2 percent of the total contributions. However, those contributions were far from trivial; 30 percent were bug fixes, 29 percent were documentation fixes, 19 percent were new-feature proposals, and 9 percent were refactoring code. The casual contributors’ most common motivation was to “scratch their own itch”—fix a problem that directly affected them. Others were motivated by giving back to the community, gaining a reputation, and improving a project.

Members of the FLOSS projects felt that casual contributors made valuable contributions. As one interviewee stated, “[Casual contributors] are great. They fix minor issues that all add up, and it takes me just seconds to evaluate, approve, and merge (thanks to GitHub).” FLOSS projects can benefit from casual contributors by

• specifically identifying tasks well-suited for them,
• creating specific project roles for them,
• creating testing or documentation infrastructure to help them, and
• providing contribution guidelines to reduce the overhead on the project team.

You can access this paper at bit.ly/PD_May17_4.

**Mobile-App Release Practices**

"Release Practices for Mobile Apps—What Do Users and Developers Think?,” by Maleknaz Nayebi and her colleagues, shares observations from a survey of 36 developers and 654 users of mobile apps that aimed to ascertain the perceived value and impact of different release practices.\(^5\) The most popular release strategies were based on time (for example, weekly or yearly) or market considerations (for example, targeting

Most developers who used a time-based release strategy were willing to bend it to accommodate user feedback.
specific countries). Some developers followed more than one strategy.

Developers tended to choose a release strategy at the beginning and continue with it throughout an app’s life. However, most developers who used a time-based strategy were willing to bend it to accommodate user feedback, especially when they were convinced that the change would impact the feedback. The results were mixed on whether the release strategy impacted development practices.

Although the users were aware of mobile-app updates, only half of them automatically updated their apps. Most users preferred to install updates themselves, and the release date wasn’t a deal breaker.

More than half of the users were hesitant to update apps. Much of the hesitancy resulted from problems encountered after an update (for example, increased memory consumption, an app or a device crashing, a device slowdown, or the loss of functionality).

Finally, users had mixed perceptions of frequent updates. Two-thirds of the users enjoyed them. Conversely, one-third of them had uninstalled an app because updates were too frequent. You can access this paper at bit.ly/PD_May17_5.

References

JEFFREY C. CARVER is an associate professor in the University of Alabama’s Department of Computer Science. Contact him at carver@cs.ua.edu.

HENRY MUCCINI is an associate professor of software engineering in the University of L’Aquila’s Information Engineering and Computer Science and Mathematics Department. Contact him at henry.muccini@univaq.it.

AIKO YAMASHITA is a data analyst and entrepreneur at Yamashita Research and an adjunct associate professor at Oslo and Akershus University College of Applied Sciences. Contact her at aiko.fallas@gmail.com.

2017 B. Ramakrishna Rau Award Call for Nominations

Honoring contributions to the computer microarchitecture field

New Deadline: 1 May 2017

Established in memory of Dr. B. (Bob) Ramakrishna Rau, the award recognizes his distinguished career in promoting and expanding the use of innovative computer microarchitecture techniques, including his innovation in compiler technology, his leadership in academic and industrial computer architecture, and his extremely high personal and ethical standards.

WHO IS ELIGIBLE? The candidate will have made an outstanding innovative contribution or contributions to microarchitecture and use of novel microarchitectural techniques or compiler/architecture interfacing. It is hoped, but not required, that the winner will have also contributed to the computer microarchitecture community through teaching, mentoring, or community service.

AWARD: Certificate and a $2,000 honorarium

PRESENTATION: Annually presented at the ACM/IEEE International Symposium on Microarchitecture

NOMINATION SUBMISSION: This award requires 3 endorsements. Nominations are being accepted electronically: www.computer.org/web/awards/rau.

CONTACT US: Send any award-related questions to awards@computer.org.

www.computer.org/web/awards/rau