Editor: **Philippe Kruchten**
University of British Columbia
pbk@ece.ubc.ca

How Best to Teach Global Software Engineering?

Educators Are Divided

Sarah Beecham, Tony Clear, Daniela Damian, John Barr, John Noll, and Walt Scacchi

WITH GLOBAL SOFTWARE engineering (GSE) becoming standard practice, today's software engineering students will be tomorrow's global software engineers. So, the education systems underpinning the profession will need to change accordingly. However, current approaches to teaching software engineering are outdated and lack authenticity, as Florian Matthes and his colleagues noted:

> *When considering the personal requirement today's software engineers are facing in their daily work life, it is surprising to see that teaching GSE at universities is still in its infancy.*[1]

GSE is an established field, and nearly all practitioners and academics agree that graduating students must have experience in it. A report from the 20th Annual Conference on Innovation and Technology in Computer Science Education reviewed the GSE education literature, exposed the challenges to teaching GSE, and provided a framework for meeting these challenges in a university setting.[2]

To stimulate debate on how to change current approaches to teaching software engineering to reflect the global workplace, Sarah Beecham asked Tony Clear, Daniela Damian, John Barr, John Noll, and Walt Scacchi to discuss how they inject realism into their courses. (This Oxford-style debate took place at the GSE education workshop at UC Irvine in August 2016; for workshop details, visit gse.sivrex.com.) Although they all agreed that changes are necessary, their approaches differed considerably. Clear and Damian argued that the best way to emulate the workplace is to engage in cross-university, multisite courses. In contrast, Barr suggested that having students contribute to open source projects gives them real-world experience without the overhead involved in cross-university courses. Finally, Noll and Scacchi argued for using online simulations and games to provide students a range of experiences that wouldn't be possible within the constraints of a university term.

The following provides an overview of the approaches they discussed, in their own words.

The Multisite, Cross-University View (Clear and Damian)

In our work with students, we seek to conduct authentic global virtual collaborations in cross-university courses. Our key goals are to develop global collaborative capabilities, develop cross-cultural understanding, and demonstrate the challenges and complexities of working

0740-7459/17/$33.00 © 2017 IEEE

in global virtual teams, thereby fostering international understanding, peace, and global sustainability.

In doing so, we've also challenged ourselves to work with global colleagues in complex, sophisticated ventures. We've taken what has often been pioneering work, in which the course becomes a living laboratory, as an opportunity to engage students in research-based teaching. Through this teaching, we model and encourage inquiry-based learning. This form of teaching or learning isn't easy or comfortable, and mistakes and frustrations abound. It's true that we don't have all the answers, but why should we shield our students from that? This is how students develop the insight and skills to work effectively and sensitively as tomorrow's global practitioners.

Over two decades of diverse collaborations, we've inquired deeply into GSE and GSE education jointly with our students, created course models and instances, and generated new knowledge. We've developed global friendships with colleagues and students and seen these courses open doors for graduates. We argue that students learn best about GSE by doing GSE. Optimally, that occurs in a structured, conscious global learning experience—although at times we must eat some of our own dog food!

Can we really teach GSE competencies in the classroom? How do we expose students to the reality of complex working relationships specific to globally distributed software development? How do we mentor them through the sustained effort of finding strategies for successful global-software-development projects?

Not only can GSE learning be achieved in the classroom, but also a university course offers a safe environment for students to experiment and work through different strategies when facing new challenges. We believe that students can learn GSE competencies in the classroom but that this requires an educational environment that offers realistic challenges and more questions than answers. Our experience designing and evaluating GSE teaching frameworks in which students engaged in hands-on GSE projects indicates that these courses resembled, as much as possible, the reality of the software industry.

The outcome is convincing. For example, we recently tried out distributed-Scrum practices in a project led by a real client and involving distributed university environments. The students experienced realistic challenges of working with remote peers and the client across organizational, cultural, and temporal boundaries. In keeping with agile processes, the students engaged in ongoing reflection about their challenges, their response to these challenges, and GSE learning. The empirical evidence we collected on the students' learning of GSE competencies shows clearly that they learned, for example, to minimize cross-boundary communication when allocating work to respond to the client's feature requests.[3]

However, a note of caution: Despite this success, setting up, teaching, and evaluating student work in such educational environments requires more effort, strategy, and instructor resilience than in traditional courses. Enabling students' learning of global software development has many facets. However, multisite, multi-university courses let educators design an experience to best relate to the current processes, supporting tools, and realities of GSE. Despite the challenges we experienced running these courses, we should continue to do so.

A Pragmatic Approach to GSE Education (Barr)

Education in an academic setting must always be a facsimile of reality. In particular, any education must take place in an artificial framework—namely, the academic institution, which places constraints on the pedagogy. Challenges include these issues:

- *Course design and organization*. Universities and instructors, even within a country, have different philosophical beliefs about how to structure and assess courses.
- *Students*. Students from different universities and cultures have different work ethics, skill levels (grad versus undergrad), expectations from instructors, and expectations about class requirements.

> A university course offers a safe environment to experiment with GSE strategies.

- *Tools*. Universities, instructors, and students have experience with different communication and development tools. Sometimes, these tools are determined at the department or university level, which makes changing them difficult.
- *Class management*. Course instructors generally have set responsibilities (lectures, grading, office hours, and so on), which become more problematic when spread across multiple universities and geographic locations. Which instructor is responsible for what? How do instructors support students in remote locations? How are student teams managed across several locations? How does the instructor manage student relationships with remote clients?
- *Sustainability, scalability, and reusability*. How do you scale a GSE education class? Such scaling greatly magnifies the GSE challenges, and instructors normally receive few if any new resources to deal with them.

As a result of these intrinsic challenges, I propose an alternative to the multi-university approach: introduce students to GSE through participation in a global open source project. Open source projects have many advantages. If the project is indeed global, students are introduced to GSE challenges and can participate in the GSE process—GSE education's major goals. Open source projects use specific tools and processes that students will need to master; this provides essential experience in using communication and development tools. As students participate in development, they communicate globally with a team that's experienced in GSE and in integrating new developers.

This approach also ameliorates many of GSE education's constraints. The GSE infrastructure already exists, the development tools are set and typically are universally known and used (for example, git, Internet Relay Chat, wikis, and blogs), the languages are usually well known (for example, Python, Java, and C++), and many resources are available to new developers. An open source project is also highly motivating; it's a real project for a solution that's often in widespread use, and the open source community generally is accepting and encouraging. Because there's such a large selection of open source projects to choose from, instructors can choose a project (or a piece of a project) that fits their course objectives and their students' abilities.

Finally, many of the institutional challenges are vastly reduced. Instructors don't have to deal with differing philosophical approaches, manage remote students, negotiate schedules, or deal with many of the constraints of a multi-university approach.

Of course, an open source project isn't the same as a GSE project in a commercial company. However, it meets GSE education's general learning objectives and reduces the challenges significantly. I believe that, for most universities, it's the most cost-effective and practical approach.

> For most universities, open source projects are the most cost-effective and practical approach.

Why Not Use Simulations? (Noll and Scacchi)

Regardless of how much effort educators devote to making a distributed-project course realistic, it will always have constraints impeding its fidelity. University terms rarely last longer than 16 weeks, students have other classes and obligations competing for their time, and they don't earn a salary.

Nevertheless, a GSE course is still a simulation of a real industrial software development project. So why not go the full distance and train students using an online simulation or game? After all, airline pilots train extensively in flight simulators, which let them experience critical situations without putting themselves at risk. A "GSE simulator" could have the same advantage, letting students experience problems and make mistakes without putting their projects (and grades) at risk. A simulator can also compress time, simulating an entire project in an hour, letting students run many trials involving different project scenarios or strategies. And, simulations and games can be fun.

Educators can use simulations and games focusing on GSE processes in many interesting ways, providing educational and research

affordances that are too difficult, costly, or lengthy to realize in live GSE course projects. Such affordances can include the capability to

- progressively guide students through GSE processes across difficulty levels, through short, self-paced training scenarios;
- monitor and incrementally assess student progress during simulated GSE process enactments;
- control the triggering or emergence of situated or contextual problems, conflicts, misalignments, time-zone asymmetries, and so on that can arise in GSE projects;
- capture and replay simulated GSE process enactments, allowing for close-up analysis and retrospective diagnosis of the actions taken and their consequences;
- incorporate live or simulated GSE tools and repositories;
- simulate benign to problematic GSE project circumstances such as mid-project staff termination and budget and schedule reductions; and
- accommodate student errors, mistakes, and process enactment failures as learning experiences that are safe, don't require remote confederates, and can simulate differences or gaps in cultural practices and diversity.

Simulations and games for GSE are no panacea; they're only as good as their developers have allowed. Poor design or implementation, or ill-conceived and poorly matched ontologies underlying the simulation, can render a simulator or game ineffective. But well-designed simulations and games offer capabilities that are scalable and open to experimentation and replicability in ways that are much easier to adopt and implement than with live role-playing GSE projects.

The approaches described here are, of course, simulations of the real workplace. The key message is that educators are changing how they teach, to try to reflect industry needs. Industry is calling for more and better-skilled software engineers; perhaps this is one way to address the skills shortage.

Acknowledgments

Science Foundation Ireland grant 13/RC/2094 partly supported this research.

> Airplane pilots train extensively in flight simulators. A "GSE simulator" could have the same advantage.

References

1. F. Matthes et al., "Teaching Global Software Engineering and International Project Management—Experiences and Lessons Learned from Four Academic Projects," *Proc. 3rd Int'l Conf. Computer Supported Education* (CSEDU 11), 2011, pp. 5–15.
2. T. Clear et al, "Challenges and Recommendations for the Design and Conduct of Global Software Engineering Courses: A Systematic Review," *Proc. 2015 ITiCSE on Working Group Reports* (ITiCSE-WGR 15), 2015, pp. 1–39; http://dx.doi.org/10.1145/2858796.2858797.
3. M. Paasivaara et al., "Learning Global Agile Software Engineering Using Same-Site and Cross-Site Teams," *Proc. 37th Int'l Conf. Software Eng.* (ICSE 15), vol. 2, 2015, pp. 285–294.

SARAH BEECHAM is a senior research fellow at Lero—The Irish Software Research Centre. Contact her at sarah.beecham@lero.ie.

TONY CLEAR is an associate professor in the School of Engineering, Computer and Mathematical Sciences at the Faculty of Design & Creative Technologies at the Auckland University of Technology. Contact him at tony.clear@aut.ac.nz.

DANIELA DAMIAN is a professor of software engineering at the University of Victoria. Contact her at danielad@uvic.ca.

JOHN BARR is an associate professor in Ithaca College's Department of Computer Science. Contact him at barr@ithaca.edu.

JOHN NOLL is a research fellow at Lero—The Irish Software Research Centre. Contact him at john.noll@lero.ie.

WALT SCACCHI is a senior research scientist and member of the research faculty at the Institute for Software Research at the University of California, Irvine. Contact him at wscacchi@ics.uci.edu.