Distributed Development – an Education Perspective on the Global Studio Project

Ita Richardson
Dept. of CSIS and Irish Software Engineering Research Centre
University of Limerick, Ireland
353.61.202765
ita.richardson@ul.ie

Allen E. Milewski
Dept. of Software Engineering
Monmouth University
West Long Branch, NJ, USA
1.732.571.7578
amilewsk@monmouth.edu

Patrick Keil
Institut für Informatik – I4
Technische Universität München
Garching, Germany
49.89.289.17386
keilp@in.tum.de

Neel Mullick
Siemens Corporate Research
755 College Road East
Princeton, NJ, USA
1.609.734.3668
neel.mullick@siemens.com

ABSTRACT

The Global Studio Project integrated the work of Software Engineering students spread across four countries into a single project and represented, for most of the students, their first major "real-world" development experience. Interviews indicated that the major areas of learning were informal skills that included learning to establish and work effectively within a team, learning how to react quickly to frequent changes in requirements, architecture and organization, and learning to manage and optimize communications. Since all these skills require rapid reaction to unpredictable factors, we view them as improvisation and discuss the role of experiential education in facilitating improvisation.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *curriculum.*

General Terms

Management, Design, Experimentation, Human Factors.

Keywords

Software engineering education, Curriculum, Global Software Development

1. INTRODUCTION

The growth of Global Software Development (GSD) internationally has meant that distributed software teams are now being used widely. The increase in volume and scope of globally distributed development will persist [7], and will have an impact on development processes, project management practices, architecture, maintenance, quality and so forth [16].

With a presence in 190 countries and over 30,000 software developers, it is inevitable that Siemens executes GSD projects. With an expanding global marketplace, a trend towards developing software in low cost countries, and the growing complexity and size of software systems, the percentage of Siemens projects that are globally distributed has been steadily increasing. Siemens Corporate Research, Inc. (SCR) has been doing research aimed at developing a better understanding of the issues and impact of various practices with respect to GSD.

In this paper, we describe part of this research, the Global Studio Project (GSP), which has organized the work of Software Engineering and Computer Science student teams from five Universities in four countries into a single global project. We present and analyze experiences made by teams from three Universities. Since our key focus here is on the educational aspects of GSP, we begin by summarizing previous literature on experiential learning in Software Engineering, especially distributed educational contexts in Software Engineering. Then we describe the GSP process and what it meant for the existing University coursework. Finally, we summarize our insights and those of our students to assess the role that experiences like the GSP can play in professional development.

1.1 Teaching the "Real World"

When we teach students, we try to give them an insight into what the 'real-world' is about. We present material in courses such as software processes, components, architecture design, requirements engineering and human-computer interaction.

These academic experiences alone are not sufficient for preparing Software Engineering students for productive careers [19]. Project managers require "a combination of technical, human resources, and management skills coupled with on-the-job experience" [22]. Thus, Universities need to educate professionals to have more than a technical understanding of the personnel and management issues surrounding successful project in real contexts. But Surendran, et al. [24] note that a difficult thing to achieve in a curriculum is realism—real products signifying tangible, relevant achievements and real people signifying collaborative effort. These researchers utilized well known taxonomies of both learning mastery levels and Software Engineering topics [17] to develop a model of professional mastery appropriate for the real world. The model depends on a complex interaction of subject matter and method of instruction. For example, basic factual information about core topics like requirements, design and testing are well suited to traditional classroom instruction because they require only recall and comprehension. However, deeper levels of mastery, such as application of this knowledge, as well as its analysis and synthesis, require extra-classroom experiences such as practicum, apprenticeships and internships. In addition, some critical categories of knowledge are best conveyed in these extra-classroom experiences. For example, how to work efficiently in collaborative teams is a skill that is difficult to teach in the classroom, but can only be practiced effectively in a real-world team-oriented project [1, 23].

In fact, we believe that there is a substantial set of important "informal" skills that fall into this category. They include the ability to:

- quickly learn about a domain or technology to begin project planning
- figure out an organizational structure well enough to seek needed expertise efficiently
- assess others' abilities to support accurate team planning and status assessment and to know when to take the lead
- react swiftly to project changes.

While these skills undoubtedly benefit from classroom training indirectly, many of them have a strong improvisational element and depend on the context at hand. Dawson [9] has vividly summarized several of these skills and has discussed their simulation in software engineering coursework. Reif and Mitri [22] also draw the distinction between traditional classroom training and practical experience with "informal" knowledge. They postulate that senior managers "construct mental models for the project, problems, and opportunities under investigation. Next, they scan their mental experience repository for adequate actions to be taken against problems or to explore opportunities".

1.2 Teaching Distributed Development

Since GSD is being used increasingly, it is critical that graduates develop relevant skills. Many of the informal skills and knowledge areas described above are complicated significantly in a global context [6, 2]. Bruegge et al. [5] found that even with a rich set of collaboration tools and some face-to-face meetings between certain team members, actual collaboration and information sharing between geographically remote teams was difficult and infrequent. Edwards & Sridhar [11] studied virtual teams of students distributed across different countries and found that trust between the teams and the clarity of task-structure definition were both key factors in the teams' efficiency, effectiveness and level of satisfaction. Although these factors play an important role in any project, the global nature of this study likely increased their importance.

The role of less formal factors has also been made clear in the extensive body of research on remote and global collaboration outside of the educational context. Hersleb and Grinter [14] found that while organizations attempt many mechanisms to coordinate cross-site work this is vulnerable to imperfect foresight and unexpected events. Solving these requires workers skilled at informal communication and negotiation. Moreover, GSD makes all these activities more difficult; the lack of subtle modes of communication often results in decreased perception that remote coworkers are helpful, decreased ability to react to changes quickly and increased project delays [15]. GSD has stronger needs for planned communication patterns and – paradoxically – for a higher flexibility and adaptivity of processes.

2. Experiment within Siemens Corporate Research

Led by SCR and assisted by Harvard Business School, five Universities internationally set up seven distributed student development teams in order to study the problems of GSD. At the same time, the project allowed the students involved to develop an experiential understanding of GSD.

To get results as close as possible to problems appearing in real projects, the GSP was set up to shadow an actual GSD project in Siemens. The project organization, management, and the system being developed largely mirrored that of the real project. This experiment allowed investigation of particular questions without risk to current Siemens business. The SCR central team was responsible for the high-level requirements, software architecture, system test and integration, and project management. Distributed student teams were responsible for design, development and unit test for defined work packages (core modules / sub-systems). The central team included two M.Sc. students from TUM who completed research dissertations.

The central team modified the requirements from the real project to accommodate student resources available and to allow for development and testing without specialized environments. They defined the architecture keeping in mind both quality requirements and the composition of the student teams. The central team included a build meister and supplier manager. The build meister was responsible for system testing, integration, configuration management, and delivery of artifacts from the teams. Supplier managers managed particular student teams. They were responsible for monitoring progress of the team, being the first point of contact and coordinating the communication between the teams and other project members. Teams, as a rule, were not permitted to communicate directly with one another but spoke directly to their supplier manager.

The project plan reflected the iterative nature of the intended development process. Because of the communication issues typically experienced in GSD projects, additional controls were put in place to track progress against the plan and to identify issues early. One of the approaches taken to accomplish this was to define four to six week milestones with clearly defined deliverables by all the teams and an integration task at each of the milestones (engineering releases). The first engineering release was essentially a "hello world" with a minimal delivery due from each team. This ensured that all of the infrastructure was in place, identified any major issues in the development and delivery process and served as a first test for the integration process. Subsequent engineering releases were more closely related to the releases expected from a real-world distributed project.

The make-up and background of the University teams varied. Monmouth University, New Jersey, U.S.A. had three teams comprised of 5 Masters Students each. The other Universities - University of Limerick, Ireland, Technische Universität München, Germany, Carnegie Mellon University, U.S.A., and Indian Institute of Information Technology, India – each had one team. The team in the University of Limerick was composed of 5 M.Sc. in Software Engineering students and the team in Technische Universität München was composed of 3 Masters students. The teams were engaged in the definition phase of GSP. The schedule, available weekly student hours, and the geographic proximity of the teams to the central team were all considered when defining the original work packages which were defined and distributed to all of the teams. The initial work packages contained a partially defined requirements specification, high level architecture, market intent, specification for the protocol used by the field systems, high level project plan, and a description of the tasks to be completed by the team for the first engineering release.

2.1 Course Summary

The courses participating in the GSP at University of Limerick (UL), Monmouth University (MU) and the Technische Universität München (TUM) have many features in common. First, they are all graduate (M.Sc) classes designed for teams of between three and twelve students with the goal of teaching them "what it means to work in a team and how to go through the whole project life cycle" [13]. Students collect and analyze requirements, create an architecture, detailed software design, and often a user-interface design. Teams also develop the working software, carry out a test plan and provide customer documentation for installation and operation of the product. Typically, projects in these courses develop fairly simple applications. Teams work fairly autonomously; the course instructor serves as an advisor. Students generally enter the courses with some experience, having completed a relevant undergraduate degree. Some of them have recommenced formal education after having spent time in industry. All GSP participants in these courses were volunteers.

There were also some differences between participating institutions. For example, at UL, students had additional work, to complete their M.Sc. dissertations. They had to compare an aspect of the GSD process, such as project management, with that local software development, providing insights into differences. At MU, the graduate practicum is a capstone course that also requires a series of team documents, such as project plan, requirements, design and test plans, etc. Many of the GSP participants were employed together as government SE interns, but they were strongly encouraged to interact only within their teams, not across team boundaries. At TUM, after the completion of the project, a report has to be prepared and the results are presented to other students and the chair staff. The team was asked to define, adopt and evaluate its own processes based on agile methods. As only the global process was defined, the students decided on using Extreme Programming and SCRUM as a base for their process.

2.2 Research Project

In addition to GSP being an educational experience, research activities focused on global collaboration and engineering process. During the year, students completed interviews and questionnaires designed and conducted by Harvard Business School with input from the research partners. Students completed a weekly on-line questionnaire, which focused on communication patterns and on their perception of the current project status (products, understanding of requirement, etc.). Each group participated in a monthly telephone interview. Additionally, each team member participated either in a semi-structured individual interview or in an open-ended email questionnaire, focusing on the development process within which the students were involved and the educational outcomes from this process. Content analysis to identify overarching themes invoked by the participant themselves was carried out by the authors. A broad analysis of the frequency of different concepts was conducted based on an interrogation of the interview transcripts.

2.3 Educational Value and Insights

Motivation for Participating: The main reason GSP students volunteered was that they were interested in getting some 'real' development experience. Generally this was driven by a curiosity about environments outside their own academic and employment contexts. For example, one student was eager to work "*with a firm that has a great reputation in the software world. I wanted to learn how industries other than government produce software*." Another mentioned that "*working with a large-scale company and the possibility to have our work actually be used by the customer in the future was intriguing.*" Yet another "*liked the 'realness' of the project as compared to a made-up one*". Several mentioned that it would look good on their CVs. UL tied the GSP to a dissertation and some students participated as it gave them an interesting dissertation topic. These students had to become reflective practitioners on the project studying one chosen process.

Teamwork: Some of the students found that working on a team in a real-world environment was much different than what they had previously experienced in class projects. Comments included: "*There can be a lot of problems with team dynamics"; "How difficult it is to work in groups, ...even organizing times to suit everyone"; "Working in a team – working through problems faced and task allocation"*. Depending on previous experiences, some teams had less difficulty with this aspect. "*Team had common concept of how well we wanted to get the project done". "We were able to work together, come up with creative solutions. Some were good at development, some at documentation. One would make a breakthrough, then the others put their shoulders to the wheel"*.

Involvement in a life-cycle: This project gave students the opportunity to be involved in a software project's complete life-cycle, from where they had to define the partial requirements given by SCR to providing a tested product. In the classroom, it is very difficult to give a project which includes a full life-cycle. Students found that "*It was good to figure out how work packages should be delivered". "How do you go about doing a requirements document? We all knew it needed to be done but how to do it was a problem"*.

Commencing the Project: One of the most difficult things for the student teams was figuring out how to begin the project. *"The* *only difficulties I experienced was getting started."* Some of these difficulties even preceded requirements gathering. Among others, students were provided with a very large document describing an industry standard protocol that they were to use, but most students did not read the details. SCR referred them to this document, and eventually, teams found ways to gain expertise on the protocol. For example, one team assigned one team member to read the document as his sole responsibility.

Changing requirements: A key insight made by many students concerned the chaotic and frequently-changing nature of real projects and the importance of reacting quickly to these changes. This is a point made often in the classroom, but the GSP experience made it real. "*Our customer did not know what he or she wanted- a very common thing in the software world." "The industry benefit was that we saw how a real project is run- delays, changing requirements, shifting of personnel, etc"*. The changes during the project resulted in coordination problems since it got *"very hard to resolve open questions via email, especially when you don't know what you don't know"* and because *"you don't know where to start asking questions"* because so many new things came up.

Communication: Because of the nature of the research, nearly all extra-team communications were between each student team and the SCR central team, rather than between two or more student teams. Compared with traditional classes and practicum, the GSP was viewed positively because it exercised skills in communicating with other technical people (on the central team). "*I got experience in interacting with other software organizations. A very common thing to do in the work environment". "In this project the customer knew more about the technologies so it was more demanding. It was not easy to get around or fluff at the end." "The fact that the Siemens representatives themselves were very technical provided some benefit in the fact that they required us to develop things ... we were not familiar with. So, we had the opportunity to learn some new concepts/tools."* Communicating with the SCR team, however, was not always smooth. Along with difficulties collecting clear requirements, the most common difficulty reported by participants was communicating with the central team. In some cases this was because of personnel changes midway through the project. In other cases it was because the central team did not have clear answers, because it took substantial time to ramp the project up to a smooth state. In many cases, however, the problem was that the student teams were not proactive in asking questions of the central team or were not adept at integrating answers into their own cognitive frameworks. *"We were inexperienced. We didn't know when to go to SCR." "At times we did not speak the same terminology and it was confusing. The central team would mention something in talking about design issues and we would not understand because our terminology for that same issue might be different."*

In the context of GSD, face-to-face contact with central team members proved to be extremely important, even though it was infrequent. One student wrote that *"it seems that the lack of face-to-face talk can lead to apparent similar perceptions while both parties still have differing "images" of the system. So being precise and verifying even aspects taken for granted is important, which can only be accomplished by intensive communication."* At one point, one MU team requested a visit from their supplier-manager in order to clear up what seemed like endless confusion over their software designs. This was recognized to be a useful tactic and was quickly followed by similar requests from all MU teams. In all cases, these visits resulted in a marked improvement in understanding and progress continued after the meetings. Communication may have been especially difficult for the UL team because they did not have a supplier manager responsible for them. All of their communication with central team personnel was on a more ad hoc basis. This caused communication problems - *"We felt it came from – we felt isolated – that was part of the project". "There should be more phone conferences". "Lots of problems were tactical – like how to communicate with SCR"* as the team did not know them personally. Again, face-to-face meetings were critical. Until they had an opportunity to meet face to face, both formally at meetings and socially with SCR personnel, did they relax about communicating regularly with SCR. The TUM team had no supplier manager, but the advantage that a German student joined the SCR team for the second half of the project. Even if their English skills were very high, this eased communication *"because some things are easier to explain in German.*" At the beginning, though, *"communications with the central team was not effective."* After meeting one representative of SCR face-to-face, the students had *"a better idea who to ask questions to."* All this shows why – independent of team size and location – face-to-face communication was rated *"most effective"*.

2.4 Improvements and Expansions

Scheduling: One of the difficulties the students faced was that they were not a full-time development team, but also had to schedule coursework and other projects while working on the Global Studio Project. This caused difficulties - "*Sometimes they (SCR) weren't happy with us – we didn't always get back to SCR. We didn't want to give the time to be available everyday to get back to them*". It is important that we, as project supervisors ensure that the students maintain a balance between what they are doing for SCR and what they are doing for their other coursework: "*How to fit this into the course – balancing between getting a project in for 30% and an SCR deliverable*".

Development process: SCR provided a development process called MD.RAD (Model Driven Rapid Application Development) that is specifically designed for global projects and which was planned to be applied as a reference process for GSP. In contrast, local teams needed to establish their own processes. There is huge potential for future GSP phases to gather more and deeper insights in the strengths and weaknesses of MD.RAD as well as of local methods. One option could be to ask one or two teams to choose different methods and implement them. For the students, this would result in a deeper understanding of such processes. For the researchers and the instructors, such experiments would lead to improvements in the overall process and would add new experiences on how to plan and control distributed projects. The work package to define, implement and evaluate local methods was planned for the first phase, but its priority was displaced downward when the software implementation phase began.

Project planning: Future GSP phases need to provide more robust methods for information sharing and for alerting teams to project changes. "*SCR had nothing set up to keep the team informed that things were happening – change to requirements,* *change to project plan, change in personnel*". *"What's taking up our time is finding out what we are supposed to do."*

Provision of Resources within University: A requirement from SCR was that the students' work was under Non-disclosure agreement. This did not cause a problem in general, but it did require that students were located in a laboratory which gave them some privacy. This is not always an easy task to achieve within a class of people who are also working on projects.

Furthermore, at all the Universities, the students were slow to discuss problems with their dissertation supervisor, as this could be seen as a weakness on their part, and might affect their final grade. They need a teaching support person who would be able to help them when their technical skills were weak so that they would not have to call on their supervisor when problems arose.

Inter-Team Communication: Some of the communications difficulties resulted from teams being restricted to interactions with the central team; there was a large load on the central team and the level of detail that could be conveyed by this "middleman" was sometimes limited. At one point in the project, a single MU team was allowed to communicate directly with one of the teams of Carnegie Mellon University over the details of a shared interface. In this case, direct communication seemed to simplify communication. Of course, with globally distributed teams, language, distance and cultural differences make such communication more complex. Nonetheless, in the next phase of GSP, inter-team communication will be encouraged and studied and in this sense will be more of a traditionally global collaboration. This will also give students the opportunity to directly perceive cultural differences and their effects and to learn how they can be handled.

Tools: An important side effect of a real-world project like the one presented is "to familiarize students with some of the CASE tools used in the industry" [23]. Therefore, without focusing the project on this, some of the teams could be asked to evaluate different tools (e.g. for configuration management, testing etc.) and configure the ones chosen. This time, the central team took these decisions.

3. Conclusion

Instructors' involvement in this project centered on both research and educational perspectives. From the research perspective, this project affords the opportunity to study the effects of GSD and can help answer questions such as: how should project management be different in a local / global environment? How can the software be divided out among teams? How can the software be integrated successfully? What information-seeking strategies are used in global teams? To answer these questions, a huge amount of data was collected and many experiences have been documented for evaluation.

From an educational standpoint, instructors' interest in the GSP centered on its opportunity for real-world experiences. Students had to set up their own project team, appoint a project manager, manage the work coming in from SCR, deliver output, work with modifications required by SCR and communicate with people they had not met. This gives them experiential learning in a global environment. Similarly, most students volunteered for the project in an active attempt to get real experience outside their own academic and employment contexts.

A key concern of this investigation has been to understand in detail what can be learned in such a real-world project. Student interviews and questionnaires revealed that the major areas of learning were informal skills that included learning to establish and work effectively within a team, learning how to react quickly to frequent changes in requirements, architecture and organization, and learning to manage and optimize communications in the GSD environment. This list of key areas learned may not be surprising in that they are all well-known and critical areas of practice required to be an effective software engineer. What is notable is that these areas are all those that are traditionally thought to be difficult to teach in a standard classroom setting. Their centrality in the GSP is consistent with the framework proposed by Surendren et al, [24]. Students struggled with these areas during their GSP experience but learned something about the behaviors required to react successfully.

A useful model for the needed behaviors is that of improvisation, being a complex cognitive process of adapting already-learned material and high-level rules and heuristics to apply to unanticipated events in new situations. Successful improvisation requires that the improviser has a large repertoire of previously-learned solutions and techniques to draw on and then creatively apply them in reaction to a current, dynamic situation [3]. While improvisation is often associated with artistic endeavors, its value has more recently been realized in domains as diverse as organizational management [3] and emergency-response [20]. Finally, Dybå, [10] has concluded that improvisation is a critical skill for the success of professional software organizations-especially those that work in small teams. According to Dybå, as software environments become increasingly unpredictable, successful software engineering requires a careful balance between disciplined exploitation of already-learned processes and experimentation with new behaviors. The correct balance depends on the context of the specific project- both team size and project turbulence.

We believe that the most significant educational value of the GSP was in the area of improvisation. When students joined the project, they had already experienced most or all of the traditional class work associated with software engineering programs. This gave most of them at least a basic a repertoire of project process models, team organizations, architectural and software design patterns and testing strategies. While stringing these together in a single, complex project provided some educational value in itself, the real discoveries came from the frustrations and anxiety associated with the need to research, plan, communicate and respond to frequent changes quickly in the GSD environment.

Besides the challenges to transfer theoretical knowledge into a realistic project setting, which requires informal skills and improvisation talent, students also learn to integrate and combine their knowledge on different subjects. In a project environment like the one for GSP, students are forced to regard testing not as a problem of models or even tools, but also as a managerial, organizational and technical challenge.

From the instructors' standpoint, it is tempting to try to teach students how to improvise within a software project even before their experience with it. However, this may not be feasible. Improvisation is often believed to be a "learnable" skill; there are educationally-oriented treatments of it in artistic contexts [18].

Mirvis [21] describes the benefits of practicing improvisation as part of a business education. However, virtually all of these treatments avoid direct instruction in favor of exercises and scenarios where students can practice and gain insight into the skills needed. Drawing on these exercises, students become aware of what needs to be improvised, develop a vocabulary of improvisation to facilitate planning and also become desensitized toward the anxiety associated with the dynamic environments where improvisation is required [21]. We feel these insights are as important for software engineering students as they are for other fields, and that the GSP has provided an useful environment to facilitate their learning. We are also sure that this project increased – ceteris paribus – our students' 'market value' by experiencing a large-scale project and getting insights into the 'real world' of globally distributed software development.

4. Acknowledgement

We would like to thank the students who participated in the project. Participation by UL was funded by Science Foundation Ireland through the GSD for SME project.

5. REFERENCES

[1] Alfonso, M. I. and Mora, F. Learning Software Engineering with Group Work. *In Proceedings of the 16th Conference on Software Engineering Education and Training* (March 20 - 22, 2003). CSEET. IEEE Computer Society, Washington, DC, 309.

[2] Azadegan, S. and Lu, C. An international common project: implementation phase. *In Proceedings of the 6th Annual Conference on innovation and Technology in Computer Science Education* (Canterbury, United Kingdom). ITiCSE '01. ACM Press, New York, NY, 125-128.

[3] Barrett, F. J. Creativity and Improvisation in Jazz and Organizations: Implications for Organizational Learning. *Organization Science, 9, 5* (Sep/Oct, 1998), 605.

[4] Brereton, P., Gumbley, M. and Lees, S. Distributed Student Projects in Software Engineering. *In Proceedings of the 11th Conference on Software Engineering Education and Training* (February 22 - 25, 1998). CSEET. IEEE Computer Society, Washington, DC, 0004.

[5] Bruegge, B., Dutoit, A. H., Kobylinski, R. and Teubner, G. Transatlantic project courses in a university environment. *In Proceedings of the Seventh Asia-Pacific Software Engineering Conference* (December 05 - 08, 2000). APSEC. IEEE Computer Society, Washington, DC, 30

[6] Carmel, E. *Global Software Teams*. Prentice-Hall, Upper Saddle River, NJ, 1999.

[7] Casey, V. and Richardson, I. Virtual Software Teams: Overcoming the Obstacles, *3rd World Congress on Software Quality,* (September 26th-30th, 2005). Munich, to appear.

[8] Davison, D. Offshore Outsourcing: Market Overview. *METAspectrum*. October 2004.

[9] Dawson, R. Twenty dirty tricks to train software engineers. *In Proceedings of the 22nd international Conference on Software Engineering* (June 04 - 11, 2000). ICSE '00. ACM Press, New York, NY, 209-218.

[10] Dybå, T. Improvisation in Small Software Organizations. *IEEE Software. 17, 5* (Sep. 2000), 82-87.

[11] Edwards, H. K. and Sridhar, V. Analysis of the Effectiveness of Global Virtual Teams in Software Engineering Projects. *In Proceedings of the 36th Annual Hawaii international Conference on System Sciences* (January 06 - 09, 2003). HICSS. IEEE Computer Society, Washington, DC, 19.2.

[12] Favela, J. and Peña-Mora, F. An Experience in Collaborative Software Engineering Education. *IEEE Software. 18, 2* (Mar. 2001), 47-53.

[13] Gnatz, M., Kof, L., Prilmeier, F. and Seifert, T. A Practical Approach of Teaching Software Engineering. *In Proceedings of the 16th Conference on Software Engineering Education and Training* (March 20 - 22, 2003). CSEET. IEEE Computer Society, Washington, DC, 120.

[14] Herbsleb, J. D. and Grinter, R. E. Splitting the organization and integrating the code: Conway's law revisited. *In Proceedings of the 21st international Conference on Software Engineering* (May 16 - 22, 1999). ICSE '99. IEEE Computer Society Press, Los Alamitos, CA, 85-95.

[15] Herbsleb, J. D., Mockus, A., Finholt, T. A. and Grinter, R. E. Distance, dependencies, and delay in a global collaboration. *In Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work* (Philadelphia, PA, United States). CSCW '00. ACM Press, New York, NY, 319-328.

[16] Herbsleb, J. D., Paulish, D. J. and Bass, M. 2005. Global software development at Siemens: experience from nine projects. *In Proceedings of the 27th international Conference on Software Engineering* (May 15 - 21, 2005). ICSE '05. ACM Press, New York, NY, 524-533.

[17] Hilburn, T., Hirmanpour, I., Khajenoori, S., Turner, R. and Qasem, A. A Software Engineering Body of Knowledge, Version 1.0, tech. report CMU/SEI-99-TR-004, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1999.

[18] Johnstone, K. *Impro: Improvisation and the Theater*, Theatre Arts Books, 1979.

[19] McCracken, W. M. Counterpoint-SE Education: What Academia Can Do. *IEEE Software. 14, 6* (Nov. 1997), 27.

[20] Mendonca, D. and Wallace, W. A. Studying Organizationally-situated Improvisation in Response to Extreme Events. *International Journal of Mass Emergencies and Disasters, 22, 2* (August 2004), 5-29.

[21] Mirvis, P. H. Practice Improvisation. *Organization Science, 9, 5* (Sep/Oct, 1998), 586.

[22] Reif, H. L. and Mitri, M. How university professors teach project management for information systems. *Commun. ACM 48, 8* (Aug. 2005), 134-136.

[23] Robillard, P. N. Teaching Software Engineering through a Project-Oriented Course. *In Proceedings of the 9th Conference on Software Engineering Education* (April 21 - 24, 1996). CSEE. IEEE Computer Society, Washington, DC, 85.

[24] Surendran, K., Hays, H. and Macfarlane, A. Simulating a Software Engineering Apprenticeship. *IEEE Softw. 19, 5* (Sep. 2002), 49-56.