

Transitioning to Distributed Development in Students' Global Software Development Projects: The Role of Agile Methodologies and End-to-End Tooling

Christelle Scharff
Seidenberg School of Computer Science and
Information Systems
Pace University, New York, USA
cscharff@pace.edu

Olly Gotel
Independent Researcher
New York City, USA
olly@gotel.net

Vidya Kulkarni
Computer Science Department
University of Delhi
Delhi, India
vkulkarni@cs.du.ac.in

Abstract—From 2005 to 2008, we explored different models of collaboration in student software development projects. In the past, project roles were distributed across students in the US, Cambodia, India and Thailand. What was common to our previous models was the co-location of developers, the client and quality assurance roles being the ones that were commonly distributed. A loose waterfall software development process was always used and activities were supported by a mashup of technologies. In 2009, we distributed the developers across the US, India and Senegal to form a truly distributed developer role. We also switched to the use of Agile methodologies with Scrum and to an end-to-end tooling solution, specifically the IBM Rational Team Concert environment. This paper describes the new model and reports on the evolution of our process and tooling infrastructure. In particular, it investigates how well Agile and Scrum practices supported our model and how important tooling is to their implementation. Initial guidelines for other educators are provided.

Keywords-Agile Methodologies; Global Software Development; Scrum.

I. INTRODUCTION AND BACKGROUND

In today's software industry, development teams are no longer co-located but distributed over cities, countries and continents. Communication and coordination challenges are amplified by distance, time zones and cultures [10]. Agile methodologies recognize that a close relationship with the customer and face-to-face communications are crucial for success in software development [13][16]. They are based on the repetition of short work cycles involving requirements, design, coding and testing activities, as well as on the production of shippable software at the end of each cycle. Scrum is a generic, iterative and incremental management framework often used in combination with agile methodologies [3]. Despite some conflicting premises, there is a growing interest in using Agile methodologies and Scrum in a distributed context in the industry

[12][14][17][19] and in academia [1][8][11]. Clearly, students need to be prepared to understand the opportunities and challenges in these scenarios. In this paper, we describe how we have built on a mature Global Software Development (GSD) teaching initiative to explore such scenarios. The current research on using Agile methodologies in class settings has mostly concentrated on the use of Extreme Programming and pair programming for co-located teams [9]. One of its practices, pair programming, has been studied in a distributed context in education [1][8][11].

For four years, from 2005 to 2008, we explored different educational models to enable students distributed across three to four different countries to work together on GSD projects [4][5][6][7]. The models involved up to sixty concurrent students, from the US, Cambodia, India and Thailand, as co-located teams of developers with a client outside the US. The involved countries are at different places on the offshore outsourcing equation, but we believe that this diversity is crucial for students to experience as they go through the issues and acquire the skills associated with distance, different time zones and cultures. The emphasis was on different themes each year, ranging from global supply chains, tooling infrastructures, software quality assurance, integration, deployment and maintenance, entrepreneurship and socialization. We therefore examined models comprising distributed sub- and lead- contractors to expose students to global supply chain management, coaches and auditors to improve software quality and deployment likelihood, and socialization between team members to improve team bonding and study its impact on software quality. The software engineering process followed by the teams was generally a loose waterfall process with iteration and feedback, and the tooling infrastructure was a mashup of technologies. The engineering tooling converged in the use of an integrated development environment with unit testing and version control, specifically Eclipse with JUnit and Subversion, while Java.net was used for bug reporting. The communication tooling comprised asynchronous mailing lists and synchronous Internet chats. A number of wikis served as artifact repositories, and thus facilitated team

awareness and project management tooling. Socialization activities were facilitated by Internet chats and by using the Second Life virtual world environment.

During each year of the initiative, students involved in the project experienced different roles: developers, clients, auditors, coaches, sub-contractors and socialization leaders. Extended teams comprised students of different roles distributed across locations; each location organized as a local team by role. The actual developer roles were always co-located. Over the four years, different issues arose. The developers often neglected critical communications with the auditors and coaches during crunch time on a project, since the developers focused on their own deliverables as a local team rather than on the deliverables of the extended team. Using too many tools for engineering, communication, project management and socialization impacted the productivity of the students negatively; many tools were new to the students and required training and a learning curve. Outside of the initial intent of their roles, students acting as sub-contractors decided to develop their own variants of the software to be developed. Additionally, students involved as clients generally asked to become developers in the next version of the project. The project spanned from fourteen to nineteen weeks each year, from initiation to deployment. Requirements and design activities took eight weeks and thus a first working version of the software was only ever expected after the tenth week. The lessons learned between 2005 and 2008 motivated a number of changes that led to the model described in this paper.

In 2009, to strengthen the notion of extended and distributed teams, and to provide the opportunity of being a developer to all the students, the model integrated a team of developers distributed across three countries, the US, India and Senegal. The project was to develop a mobile application for children and to span nine weeks, with three weeks of training and six weeks of work. We switched to the use of Agile methodologies with Scrum to focus on the incremental delivery of software early on in the project. An end-to-end tooling solution, specifically the IBM Rational Team Concert environment (RTC), was used to simplify tooling. Our study attempted to examine the following research questions:

- **Role of the Process** -- How well do Agile and Scrum practices support the work of distributed developers?
- **Role of the Tooling** -- How important is tooling in supporting distributed developers using Agile and Scrum practices?
- **Guidelines** -- How are Agile and Scrum practices best introduced into distributed students' projects?

The paper is organized as follows. In Section 2, we describe the 2009 project setting. Section 3 shows how we adapted and supported Agile software development with Scrum in our context. Section 4 highlights our findings and Section 5 discusses the role of the process and tools in our work. It also proposes some guidelines to build upon our initiative. Section 6 concludes the paper. The wiki of the project is available at [18].

II. PROJECT SETTING

In this section, we describe all the details of the 2009 project. This includes the learning objectives for the students, the student contingent, the software application to be developed, and the process and the tooling to be used.

A. Students' Learning Objectives

This paper focuses on the Fall 2009 GSD project where students from the US, India and Senegal developed a single piece of software together, a mobile application dedicated to helping children learn reading, writing, mathematics and geography through quizzes. The emphasis was on providing students with a real experience on using Agile methodologies with Scrum, supported by an end-to-end tooling solution. Additionally, students had the opportunity to gain skills in mobile application development with Java ME, including high-level interface design (e.g., lists and forms), the design patterns common in mobile application development, and best practices in usability.

B. Project Teams

The study described in this paper involved five graduate students enrolled in a Master of Computer Science degree at their respective universities – one in the US (Pace University, <http://www.pace.edu>), two in India (University of Delhi, <http://www.du.ac.in>) and two in Senegal (Ecole Supérieure Polytechnique, <http://www.esp.sn>). The project was not integrated in a course; students volunteered to participate in this project on the top of their other school commitments. Students in the US and India were attending classes during the course of the project, while the students in Senegal were undertaking an internship in the industry. Students had diverse motivations to take part in the project. The Pace student, who was the only one with prior professional industry experience, was interested in project management in a global context. The students in India were interested in mobile technology and in global software development, India being one of the main actors in the offshore outsourcing equation. The students in Senegal were interested in mobile technology, global software development, Agile methodologies and Scrum. The students in the US and Senegal had previous experience with mobile application development with Java ME. The three instructor coaches who participated to the project, two in the US and one in India have been working together for four years. Two are certified Scrum Masters.

C. Mobile Application To Be Developed

In 2009, the team of five students developed a mobile phone application called *Target First Grade*. *Target First Grade* was to permit pupils in the first grade (5-6 years old) to practice mathematics, reading, writing and geography, supervised by teachers or on their own. The requirement for *Target First Grade* were for it to be used in developing countries where classes of 60 to 80 pupils are common and teachers cannot provide individual attention to their pupils. Schools may not have sufficient funds to acquire computers, but they may be able to buy cheap mobile phones to be used during class time and for homework. *Target First Grade* was

to be used in English and French speaking countries and to work on a wide range of basic mobile phones (e.g., Nokia 2600c). It was to deliver exercises for practice or testing purposes, in the form of open-ended and multiple-choice questions, and to automatically compute scores. After a lesson, teachers may decide to have their pupils do problems related to the current lesson, so the list of topics and problems a pupil attempts was to be customizable by the teachers. Teachers and parents were to have the potential to receive the results by SMS.

D. Process

From 2005 to 2008, the software engineering process followed by the teams was generally a loose waterfall process with iteration and feedback. Teams of students gathered the requirements. Requirements and design activities took eight weeks and thus a first working version of the software was only ever expected after the tenth week. For the 2009 setting, we switched to the use of Agile methodologies and Scrum. Scrum provides a simple and disciplined way to manage a project more effectively and focuses on delivering value early. Agile methodologies concentrate on working software rather than excessive documentation. For preparation, students were provided with a list of readings and videos tutorials before the project.

E. Tooling

The 2009 version of the project attempted to reduce the number of tools to be used by students for engineering, communication, project management and socialization, so it converged on the use of IBM Rational Team Concert (RTC Express-C version 2.0.0.1) as an end-to-end tooling infrastructure (<http://www.jazz.net>). Other tools could have been used, such as Rally Software (<http://www.rallydev.com>), but we chose RTC because it is a collaborative development environment built upon Eclipse technology (<http://www.eclipse.org>). We believed that the students, already familiar with Eclipse, would pick up the collaborative and process features integrated in RTC quicker than having to learn new tools from scratch. RTC integrates support for different process models, including Scrum, and provides capabilities in agile planning, source code management (RTC Source Control), work item management (e.g., user stories for requirements, tasks for project management, and defects for bug reporting) and build management. Communication and coordination are facilitated through features directly available in RTC, email and chat for asynchronous and synchronous communications, wikis for planning and for sharing artifacts, and a notification mechanism to increase team awareness. Diverse reports permit project participants to get an idea on the project health and to increase their awareness of actions, behaviors and the progress of the team or project. It provides a client interface for developers and a Web UI interface for administration and less technical roles on a project.

In this study, the developers used the EclipseME plugin (<http://www.eclipseme.org>) for Java ME development and Jazz Source Control capability for configuration management. RTC wikis and notification mechanism were

used for communication and team awareness. Emails and chats were exchanged using Google groups and Google Chat, because RTC does not offer a group email / chat facility. RTC was also used for project management and contained the various Agile and Scrum artifacts. Students were also provided with tutorials on Java ME and RTC to practice before the beginning of the project. RTC features were reinforced gradually during weekly online meetings.

III. ADAPTING AND SUPPORTING AGILE AND SCRUM

In this section, we describe how we adapted and supported Agile software development with Scrum in our 2009 project context. The 2009 project context is illustrated in Figure 1.

A. Agile Software Development and Scrum

Scrum [3] is a generic framework to manage projects. It uses iterations of fixed duration (one to four weeks) called *Sprints*. It includes three important roles. The *Product Owner* is responsible for the product vision and maintains the requirements as user stories in a *Product Backlog*. Requirements are described by *User Stories*. The *Scrum Team* develops the software. The *Scrum Master* guides the team through the process and helps the team to resolve issues. During each *Sprint Planning* session, the team commits to complete a certain number of tasks derived from the *Product Backlog* and documents them in a *Sprint Backlog*. The time to complete the tasks is estimated and adjusted during each sprint. There is a demonstration of the product during a *Sprint Demo* followed by a *Sprint Retrospective* where what worked well and not so well during the Sprint is discussed. The team meets in a *Daily Scrum Meeting* where each team member responds to three questions: what he or she has done since the last meeting; what he or she hopes to do until the next meeting; and any impediments to achieving goals. The *Sprint Burndown Chart* is a metric for the team's progress that shows the time remaining for the completion of the planned work.

Scrum is commonly used in combination with Agile methodologies in software development. Agile methodologies value individuals and interactions, working software, customer collaboration and responding to changes [13][16]. They are based on iterations involving requirements, design, coding and testing activities, as well as on the production of shippable software at the end of each cycle.

Introducing Agile methodologies and Scrum into the distributed classroom setting required some adjustments. For example, students cannot meet and interact daily due to schedule and time zone incompatibilities. The roles and ceremonies of Scrum were therefore adapted to fit our model as described in the following sub-sections.

B. Scrum Implementation

The implementation of Scrum was based on encouraging students to enforce the following principles:

- *Communicate*. Sharing information creates visibility, better decision-making and a common understanding of shared goals.

- *Empower the team.* Nothing is more powerful than a team that is in control of its own destiny – a team that thinks the only thing limiting what they can accomplish is how creative they are and how hard they work.
- *Learn and improve.* Learning is about trying something, looking at the results and then improving.
- *Deliver value early.* Build trust with people by prioritizing work, committing to deliverables and delivering them reliably.

C. Agile Implementation

The implementation of Agile was based on encouraging students to enforce the following principles:

- Active client / customer involvement.
- Capturing lightweight requirements at a high and visual level.
- Allowing requirements to evolve, but within a fixed timescale.
- Application of the 80/20 rule, i.e., spending 80% of the time on the 20% of the features that most matter.
- Regular delivery of code in a shared repository, refactored by team members. Testing done early and often, integrated throughout the project lifecycle.

D. Agile Planning

The project spanned nine weeks with three weeks of initiation and training. It comprised three Sprints of two weeks each. The developers were provided with a Product Backlog from the Product Owner and they input the hours they could dedicate to the project within their RTC profiles. At the beginning of each Sprint, the stories to implement were decided during an online meeting in conjunction with the Product Owner. The team then decomposed the stories into tasks (e.g., design, testing and research) and estimated the tasks. Often, due to time constraints, the planning had to be finished by email. RTC permits the team to monitor the progress of the team and of each developer as a ratio of finished to remaining work. The velocity of the team, normally expressed in terms of story points, was expressed in terms of the number of hours to be spent on the project during the Sprint, for simplicity.

E. Key Scrum Roles

Each set of developers comprising the Scrum team (i.e., those in the US, India and Senegal) was to take a turn in playing the role of the Scrum Master during each sprint. A professional Certified Scrum Master external to the university setting played the role of Process Coach and offered weekly feedback for the team by posting comments and advice in a RTC wiki, the wiki being the only communication mechanism used by the Process Coach. This offered an industry perspective and emphasized an appreciation for process and software quality by the students. Another instructor coach (also a Certified Scrum Master) played the Product Owner and provided the vision for the mobile application. The Product Owner interacted with the developers by chats and emails. The roles are illustrated in Figure 1.

F. Key Scrum Meetings

The Sprint Planning Meetings took place at the beginning of each Sprint, synchronously via chat, and lasted one to two hours. Students determined the stories they would implement in conjunction with the Product Owner and decomposed them into tasks. The students then answered the three questions of the traditional daily Scrum Meeting in a wiki of RTC: (1) What did I do yesterday? (2) What will I do today? and (3) What is blocking my progress (if anything)? They did this every day they worked and mentioned the days when they did not work on the project. A Scrum of Scrums took place synchronously via chat once a week and lasted between two to three hours. These meetings permitted a checkpoint during the Sprint. The Sprint Demos and the Sprint Retrospectives also occurred synchronously via chat. The students were supposed to prepare the Sprint Demos by producing a video of the software and for the Sprint Retrospectives by posting what worked well and what could have worked better during the Sprint. All chat transcripts were posted in a RTC wiki to increase the visibility for the Scrum team and the Process Coach.

G. Key Scrum Artifacts

RTC uses plans to organize the Product Backlog, Sprints and releases. We organized each plan such that it comprised the planned work items (e.g., user stories and tasks), the burndown chart of the current iteration, shared documents (e.g., design and chat transcripts) and code conventions, and wikis for daily Scrum meetings, comments from the Process Coach, Sprint Demo and Sprint Retrospective. The Product Backlog of *Target First Grade* comprised 45 user stories – 18 high, 16 medium and 11 low priority user stories. After discussion during each Sprint Planning, the *Product Owner* made the *Sprint Goal* visible.

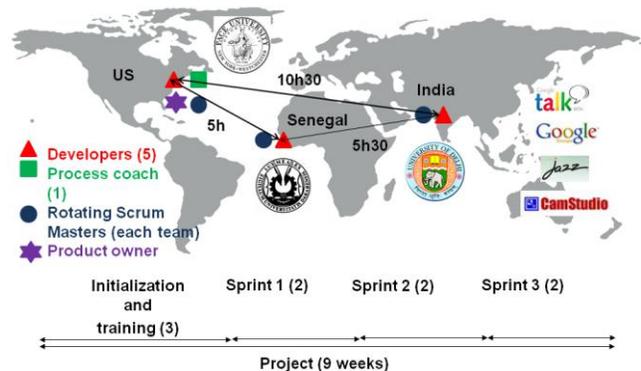


Figure 1. Setting and Timeline of the Project.

IV. FINDINGS

In this section, we highlight the outcome of the project, and summarize what worked well and what was problematic.

A. Project Outcome

Table I presents the list of stories planned, implemented and accepted by the Product Owner, the velocity of the team in terms of the planned and actual work hours, and the total

number of tasks estimated and closed. The quality of planning includes only estimated items.

TABLE I. SUMMARY OF PROJECT STATISTICS (*RTC DATA).

Metrics	Sprint 1	Sprint 2	Sprint 3
Number of planned stories*	10	18	18
Number of stories implemented by the Scrum team and accepted by Product Owner	1	2	12
Planned work hours*	59.25	82	153.5
Actual work hours done*	46.75	77.5	67.5
% of tasks estimated*	80%	75%	75%
Tasks closed / Total number of tasks*	36/41 (88%)	43/63 (68%)	17/61 (28%)
Quality of planning*	73%	38%	70%

Figures 2 and 3 present the burndown of each sprint and the burndown of the overall project. The students used Sprint 1 to learn RTC, Scrum, Agile and Java ME; they undertook small tutorial exercises. The Sprint 2 burndown shows that the developers planned too late and could never catch up. This was due to holidays (e.g., Thanksgiving in the US and Eid in Senegal) and exams (India). In Sprint 3 developers, focusing on development, did not update the status of their tasks so tracking progress and visibility were at risk.

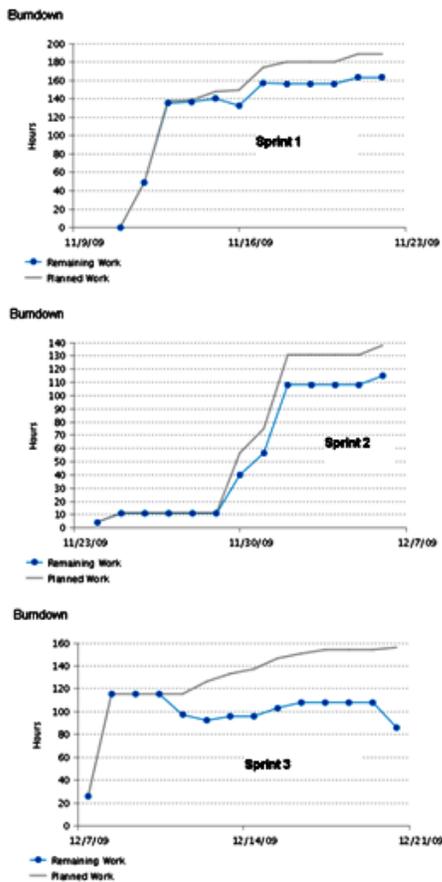


Figure 2. Burndowns of Sprints 1, 2 and 3.

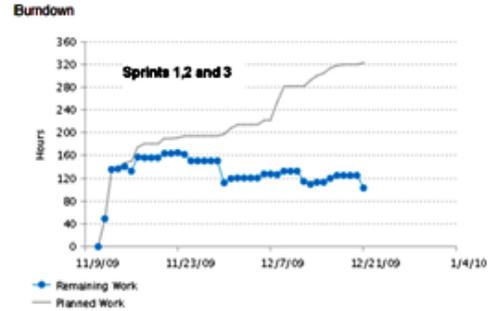


Figure 3. Burndowns of Sprints 1, 2 and 3 Consolidated.

B. What Worked Well and What Was Problematic

The effectiveness of the Agile and Scrum implementation, as adapted to our distributed student context, is presented in this section. Note that only some of the practices are presented, due to space limitations, and we focus on the process and tooling aspects.

TABLE II. SUMMARY OF IMPLEMENTATION EFFECTIVENESS.

Practice / Principle	What Worked Well	What Was Problematic
Agile Planning	<i>Tool</i> – The developers were familiar with how to set up Sprint Backlogs in RTC by Sprint 2.	<i>Process</i> – Estimates did not improve over the Sprints due to unrealistic implicit goals of the students (developer heroes). Absences were not factored into the planning of Sprint 2. Late planning in Sprints 2 and 3 due to holidays and exams.
Scrum Roles	Rotating Scrum Master <i>Process</i> – Three out of five developers experienced the Scrum Master role. Developers at one location wanted to dedicate time as Scrum Masters and took two turns.	<i>Process</i> – Scrum Masters did not facilitate Scrum Reviews, which led to delays and absence of working software to demonstrate. <i>Tool</i> – No visibility as to who is the Scrum Master.
Scrum Meetings	Daily Scrum <i>Process</i> – Daily Scrums helped to detect some issues (e.g., Internet availability).	<i>Process</i> – Scrums were not done regularly, which reduced visibility for the Process Coach and the whole team. Reasons for impediments were not detailed enough to act upon. Inconsistencies in the chronology led to confusion. <i>Tool</i> – Team member absences are not automatically populated.
	Sprint Demo <i>Process</i> – The developers realized at the final Sprint demo that demonstrating software remotely requires preparation.	<i>Tool</i> – No software was demonstrated in any of the Sprint reviews. A technically savvy Product Owner had to check out the current version during Sprint demos. Videos and screenshots were not prepared.
Scrum Artifacts	<i>Tool</i> – The developers chose the high priority user stories to work on.	<i>Process</i> – Granularity of the task decomposition was too coarse so tasks stayed

	Team got an organized set of wikis for Scrum artifacts (e.g., Process coach feedback, Sprint retrospective and code convention).	open for a very long time. User stories were dragged from Sprint to Sprint without looking at velocity. <i>Tool</i> – Product backlog and Sprint backlogs are not presented in a straightforward way. Tasks related to stories are not presented together.
Communications	<i>Process</i> – Students shared screen shots of the product to achieve consistency of the user interface. In Sprint 2, students managed to have a chat all together.	<i>Process</i> – No time was used for deciding how to work more smoothly together and integrate work, dragging problems from Sprint 1 to Sprint 3. The time difference between the three countries was problematic.
Empower the Team	<i>Process</i> – The developers decided on the stories they wanted to implement in each Sprint.	<i>Process</i> – The developers did not work as one team, but as three smaller separate teams.
Common Codebase	<i>Tool</i> – The developers looked at each other's code in RTC to learn about MVC.	<i>Process</i> – Current builds were not visible for demonstration and testing. <i>Tool</i> – Difficult for students to get used to RTC Version Control, so code was exchanged by emails.
Shared Standards	<i>Process</i> – Names of variables for Java ME artifacts were standardized. <i>Tool</i> – The developers used a set of conventions to name their MVC packages and classes.	<i>Process</i> – No standard was established for versioning of the software.
Testing	<i>Tool</i> – In Sprint 1, each developer was assigned a task to test the implementation of another developer. That permitted students to choose the main screen of the application.	<i>Process</i> – The developers tested their own work but did not test each other's work after Sprint 1.

V. DISCUSSION

In this section, we discuss the role of the process and tools in the project. We also propose guidelines for others to build upon our initiative.

A. Role of the Process

Agile methodologies and Scrum increased the transparency of both the process and the software product in development, which is crucial when developers are distributed. Their practices and principles require developers to be disciplined and regular in their work linked with engineering, communication and project management. In students' global software development projects, instructor coaches will need to factor in time to check that developers / students are doing the tasks linked with process and management (e.g., estimations, decomposition of tasks and standardization of code). Students need practice to get to that level of discipline.

B. Role of the Tooling

The student developers reported that they would not have been able to work together without using an end-to-end tool. RTC was the only collaborative tool for software development that they practiced with and so students could not compare it with other tools. It took time to find a way to organize plans in RTC that would permit each Scrum role to work effectively with each other. The developers found RTC difficult to use at first but over time this became simpler. Familiarization with the RTC source control facility was the biggest issue for students.

C. Guidelines for Instructors

In Table III, we present guidelines for instructors interested in embarking on a similar initiative. These guidelines highlight how to address the issues presented in Table II.

TABLE III. INTRODUCING AGILE AND SCRUM IN DISTRIBUTED STUDENTS' PROJECTS.

Initialization	<ul style="list-style-type: none"> • Find collaborators based on pre-existing strong relationships. • Collect information about the students, instructors, countries, institutions, etc. • Define and distribute the scenario that assigns the Scrum roles, describes the project plans and explains the organization of the key Scrum meetings. • Involve a professional to act as a Process Coach and provide regular feedback to the team to have an external and expert eye on the project. • Involve a committed Product Owner, who is not an instructor coach or a developer, to propose the software application to be developed. This is to avoid assumptions and to permit a better reflection of industry scenarios. • Include rotating Scrum Masters from the Scrum team in the setting and release them from their development work at that time such that they become better Scrum team members. • Select a minimal set of tools for engineering, communication, project management and socialization, including an end-to-end tooling infrastructure dedicated to Agile methodologies and Scrum. This is to permit students to become productive quickly. • Organize a preliminary meeting for socialization where students will learn about each other before collaborating. All team members, including instructor coaches, should attend. • Share schedule and working hours of the team to be able to do better Agile planning and to know who will be absent when. • Have the students sign a consent form that sets up some net etiquette rules (e.g., maximum time to answer emails).
Training	<ul style="list-style-type: none"> • Gather resources on process and technologies and make them available to students. Video resources and freely available online tutorials should be privileged. • Direct students to resources and have a way to check their progress with them. • Have instructor coaches at each location play the XP Game [15] to introduce students to Agile planning in a non-technical context. • Have instructor coaches at each location conduct a more technical exercise on Agile planning (e.g., Elephant Carpaccio [2]).
Facilitating and Monitoring	<ul style="list-style-type: none"> • Schedule meetings for a regular day and time. • Have the Process Coach provide weekly feedback to be read by the Scrum team.

	<ul style="list-style-type: none"> • Mix synchronous and asynchronous communications between the Scrum team and the Product Owner to prevent information being missed in a single channel. • Require developers to do the daily Scrum as a team commitment act and to increase project visibility. • Require developers to be present and to dedicate the necessary time for the Sprint Planning. Developers should not leave before the Sprint goal is agreed upon and stories are decomposed into tasks and estimated. • Require developers to be prepared for the Sprint retrospective. • Require videos for the Sprint demo. • Have developers publicize their absences. • Keep track of the team communications (e.g., chats) to increase team awareness and monitor communications. • Have rotating Scrum Masters perform their work of the Scrum Master using a checklist, with points to check. Require them to summarize what went well, what did not go well and how they will use what they learned as a team member.
Conclusion and Retrospective	<ul style="list-style-type: none"> • Formally close the project with a statement from the Product Owner. • Thank the different parties involved. • Summarize what went well on the project, what did not go well and why. • Determine how to refine the model.

VI. CONCLUSIONS

This paper described a small educational project where the software development model involved a distributed team of student developers for the first time. The educational model transitioned the process to Agile and Scrum and made use of an end-to-end tooling infrastructure. The five students involved in this project were pleased to be part of this project and qualified it as “memorable” and “enjoyable”. The study was used to evaluate and improve our model to scale and involve more students in the future. The GSD 2010 project, which is currently underway, involves a model based on Table III. It further seeks to compare RTC with Rally Software more systematically.

ACKNOWLEDGMENT

This work was supported by an IBM Jazz Innovation Award “*Evolving a Tooling Infrastructure for Global Software Development Projects: Toward an End-to-end Solution with Jazz*” and a Sun Microsystems Change Your World Equipment grant. We thank the five students involved in the 2009 version of the project and all the students involved in this GSD initiative to date. We thank Tom Reivik from the IT team of the Pace University Seidenberg School of CSIS for his help with installing RTC at Pace University.

REFERENCES

[1] Baheti, P., Williams, L., Gehringer, E., Stotts, D., and Smith, J. “Distributed Pair Programming: Empirical Studies and Supporting Environments”, UNC-CH Technical Report TR02-010, March 15, 2002.

[2] Cockburn, A. Elephant Caraccio Exercise. <http://alistair.cockburn.us/Elephant+caraccio> (Last access May 11, 2010).

[3] Deemer, P. and Benefield, G. The Scrum Primer: An Introduction to Agile Project Management with Scrum. <http://www.goodagile.com/scrumprimer/scrumprimer.pdf> (Last access May 11, 2010).

[4] Gotel, O., Kulkarni, V., Say, M., Scharff, C., and Sunetnanta, T. “A Global and Competition-based Model for Fostering Technical and Soft Skills in Software Engineering Education”. *Proc. CSEE&T’09*, 22nd IEEE-CS Conference on Software Engineering Education and Training, Hyderabad, India, 17-19 Feb 2009.

[5] Gotel, O., Kulkarni, V., Neak, L., and Scharff, C. “Working Across Borders: Overcoming Culturally-Based Technology Challenges in Student Global Software Development”. *Proc. CSEE&T’08*, 21st Conference on Software Engineering Education and Training, Charleston, S. Carolina, USA, 14-17 April 2008.

[6] Gotel, O., Kulkarni, V., Neak, L., Scharff, C., and Seng, S. “Introducing Global Supply Chains into Software Engineering Education”. *Proc. SEAFOOD’07*, 1st International Conference on Software Engineering Approaches For Offshore and Outsourced Development, Zurich, Switzerland, 5-6 Feb 2007.

[7] Gotel, O., Scharff, C., and Seng, S. “Preparing Computer Science Students for Global Software Development”. *Proc. FIE’06*, 36th Annual Frontiers in Education Conference. Borders: International, Social and Cultural, San Diego, USA, 28-31 Oct 2006.

[8] Hanks, B. Tool Support for Distributed Pair Programming, Ph.D. Dissertation, Computer Science, University of California, Santa Cruz, 2005.

[9] Hazzan, O. and Dubinsky, Y. “Teaching a Software Development Methodology: The Case of Extreme Programming”. *Proc. CSEET’03*, International Conference on Software Engineering Education and Training, Madrid, Spain, March 20-22, 2003.

[10] Herbsleb, J.D. “Global Software Engineering: The Future of Socio-technical Coordination”. *Proc. ICSE’07*, The Future of Software Engineering (ICSE-FASE’07), Minneapolis, USA, 2007.

[11] Ho, C., Raha, S., Gehringer, E., and Williams, L. “Sangam: a distributed pair programming plug-in for Eclipse”. *Proc. OOPSLA Workshop on Eclipse Technology Exchange*, Vancouver, British Columbia, Canada, October 24 - 24, 2004.

[12] Hossain, E. “Coordinating Mechanisms for Agile Global Software Development”. *Proc. ICGSE’08*, 3rd International Conference on Global Software Engineering, Bangalore, India, 17-20 Aug 2008.

[13] Manifesto for Agile Software Development. <http://agilemanifesto.org> (Last access May 11, 2010).

[14] Paasivaara, M., Durasiewicz, S., and Lassenius, C. “Using Scrum in Distributed Agile Development: A Multiple Case Study”. *Proc. ICGSE’09*, 4th International Conference on Global Software Engineering, Limerick, Ireland, 13-16 July 2009.

[15] Peters, V. and Van Cauwenbergh, P. The XP Game. <http://www.xp.be/xpgame.html> (Last access May 11, 2010).

[16] Schwaber, K. and Beedle, M.A. *Agile Software Development with Scrum*. Pearson Education, 2008.

[17] Sutherland, J., Schoonheim, G., Kumar, N., Pandey, V., and Vishal, S. “Fully Distributed Scrum: Linear Scalability of Production between San Francisco and India”. *Proc. Agile 2009*, Chicago, USA, 24-28 Aug 2009.

[18] Wiki of the GSD 2009 project is available at <http://atlantis.seidenberg.pace.edu/wiki/gsd2009> (Last access May 11, 2010).

[19] Woodward, E., Ganis, M., and Surdek, S. *A Practical Guide to Distributed Scrum*. IBM Press, July 2010