Know Yourself and Beyond:

A Students' Global Software Development Project Experience with Agile Methodology

Serene Hang Su
Seidenberg School of Computer Science and Information Systems
Pace University
New York, NY, USA
serenesou666@gmail.com

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

Abstract*—This paper describes a global software development project undertaken in an educational setting with students from Senegal, India and the US. Similar initiatives have been carried out for five years at Pace University, but it was the first time that it adopted Agile Methodologies and Scrum for the development of a mobile application. Another novel aspect was that developers were distributed across three locations. The experience was considered as an inspiring self-knowing process by all the student participants. This paper presents the perspective of one of students involved in the project. It gathers observations and analyzes the educational value of "Knowing Yourself and Beyond" based on the accomplished collaborative work and interviews with student developers and professors. The relevant lessons learned from this project are summarized and recommendations for students are provided. *(Abstract)

Index Terms* — Self-knowing, Collaboration, Agile Methodologies, Scrum, Software Engineering Education, Global Software Development. *(Key words)

I. INTRODUCTION

Know thyself.

- Socrates (469?-399 B.C.)

The central component of the philosophy of Socrates has been inspiring us for a very long time. A recent experience in a Global Software Development (GSD) has given the student author of this paper a special opportunity to contemplate its meaning once again. As a graduate student in Computer Science at Pace University and the only experienced working IT professional among the participants, she held the roles of developer and Scrum Master in the GSD 2009 project that involved students from Senegal, India and the US.

This paper principally illustrates the educational value of "Know Yourself" and "Beyond" gained from this project from a student**'s** perspective. The findings are based on observations, the documentation, the logs, the statistics provided by the collaborative tool, and the retrospective interviews with the students and professors.

This paper is organized as follows. Section II outlines the background of this innovative GSD project series that started in 2005 and introduces Agile Methodologies and Scrum. Section III describes the project context, the process, the collaborative tool and the project outcomes. Section IV lists the observations and analyzes the whole process with reference to "Knowing Yourself and Beyond". Section V states the lessons learned and recommendations for students. Section VI concludes and suggests future work.

II. BACKGROUND

For five consecutive years, Pace University in New York City has been collaborating with institutions of higher education in Thailand, Cambodia, and India to propose models to integrate global software development into the Computer Science curriculum and have students develop software together [1,2,3,4,5]. The goal of the GSD project series is to better prepare students as future IT professionals. In 2009, Agile Methodologies and Scrum were adopted for the first time with the intention to encourage students to participate in a real-world project in the most active possible manner. Developers were distributed across three countries and three different time zones. They were exposed to software engineering processes and mobile application development and experienced what "knowing oneself and others" meant in this concrete setting.

A. Agile Methodologies

Historically, the software engineering process has been described by a strictly predefined step-by-step set of activities assuming that everything is predictable. Development begins after writing often huge and detailed documentations for planning, requirements and design and the overall project is managed in a hierarchical command-and-control manner. However, predictability is a false premise and generic development methods often fail to respond to changes.

Since 2001, more "lean", "light" and "adaptive" methodologies have been introduced under the name of Agile Methodologies [6,7,8]. The Agile principles leave developers

space for adaptation and flexibility so that they can be more responsive to changes and reevaluate the current effectiveness of the processes and adjust their work when necessary. Agile Methodologies force developers to be disciplined and keep a constant pace in moving their work forward to assure a rapid delivery of smaller software components and achieve faster and more accurate feedback from customers [9]. Some studies show that given comparable software projects, the teams using Agile Methodologies can be 50% faster to market and 25% more productive with one quarter less defects than teams using traditional software engineering practices [10].

B. Scrum

One of the most popular Agile frameworks for project management is Scrum. It is an iterative and incremental framework organized around short iteration cycles called *Sprints*. The work is done by the members of the Scrum Team in communication with the Product Owner. The *Product Owner* maintains the requirements of the product as prioritized *User Stories* in the *Product Backlog*. The *Scrum Master* is in charge of enabling Scrum by helping the team remove impediments and thus creating the best possible circumstances to achieve the goal fixed for the Sprint. Typically, one Sprint runs in the following way:

1) At the beginning of each Sprint, the team discusses and determines the *Goal of the Sprint* with the Product Owner. The user stories are broken into tasks. Team members select the tasks they want to tackle proactively, instead of simply being assigned tasks by a conventional project manager. This is an opportunity for team members to take ownership of their work depending on their own set of skills and schedule;

2) During the Sprint, everyone thrives to do the work they committed to. One of the most important practices of Scrum is the *Daily Stand-up Scrum*, which takes place at the beginning of each working day. The team members must answer three questions individually: a) What did they do yesterday?; b) What will they do today?; and c) What are the impediments that prevent them to move forward?; and

3) At the end of the Sprint, the team demonstrates what they built and get feedback from the Product Owner to incorporate in the next Sprint work. The emphasis is put on producing code that is fully tested and shippable at the end of the Sprint. To improve the team performance in the next Sprint, team members reflect on their activities in a *Sprint Retrospective*.

III. GSD 2009 PROJECT

A. Context

The GSD 2009 project aimed at applying Agile Methodologies and Scrum in developing a single mobile application in Java ME for a Product Owner. The mobile application, called *TargetFirstGrade*, permits children (5-6 years old) to attempt quizzes in mathematics, reading, writing, and geography, and get instantaneous feedback on whether their answers are correct or not. There were five student developers distributed across the US (Pace University), India (University of Delhi) and Senegal (Ecole Supérieure Polytechnique de Dakar) who worked closely with a Product Owner to develop the software. The students were graduate students in Computer Science programs who volunteered to be part of this project. Three students on the project had previous experience with Java ME. Students were provided with tutorials and videos on Scrum, Agile Methodologies and Mobile Application Development with Java ME one month before the beginning of the project. The Product Owner was the second author of this paper and the lead instructor of the project. As a Certified Scrum Master, she also facilitated the whole project. In addition, she provided technical guidance. A professional process coach, also a Certified Scrum Master, offered weekly advice on the overall process to the developers.

B. End-to-end Collaborative Tool

The IBM Rational Team Concert 2.0 (RTC) software was used to support the software engineering process [11]. By taking into account the fact that developers are often part of distributed development teams, RTC is designed to integrate all the work artifacts, team roles, schedules, documentations, communications and project management artifacts into a robust working platform to support all the working styles and roles. RTC has a specific template for Scrum. It allows developers to plan work items, estimate the time they need to complete them, document the actual time they spent in completing them, and notify selected developers about the status of work items they may depend on. RTC provides numerous metrics (e.g., number of planned items completed and quality of planning) and charts (e.g., burndown charts). All the statistics data of this paper were produced by RTC.

C. Process

The GSD project used Scrum and Agile Methodologies for the first time in 2009, while a loose waterfall model has been used from 2005 to 2008. A certain level of rigidity implied by Agile Methodologies and Scrum rituals is useful in an educational setting for students to learn about process and their own abilities.

The complete project time line is shown in the Table II. It lasted two months with 24 days for preparation and training and 42 days (three Sprints of 14 days each) for development. The project was kicked off by a first online meeting on November 9th 2009 and finished on December 21st 2009. There was one planned regular online weekly meeting to be attended by the team, the Product Owner and the instructors. There were seven planned weekly meetings. Developers at each location were supposed to act as Scrum Master alternatively. The student author of this paper played the role of Scrum Master in the first Sprint and learned by doing. She acted as the Scrum Master in a more comfortable way in Sprint 3 when she decided with the agreement of the team and the Process Coach that it was important to have a strong Scrum Master to make the project successful. The kick-off meeting of Sprint 1 discussed the product features. The plan for Sprint 1 was discussed during the meeting. The lead professor gathered the tasks that the team was committed to

and consolidated them into a formal plan posted in RTC. A one-week review meeting took place in the middle of Sprint 1. At the end of Sprint 1, the software built during the Sprint was showcased and a Sprint Retrospective meeting was held. The process coach provided weekly comments and advice to the team. This pattern repeated in Sprints 2 and 3. The developers were more familiar with RTC as time went by.

Since the developers were geographically distributed, they were unable to benefit from the full advantage of the Daily Stand-up Scrums. As a replacement, Daily Scrums were to be posted in writing in the documentation section of RTC at the beginning of each working day. In a distributed development setting like this, developers have to heavily rely on each other's self-discipline to update their Daily Scrum. It was agreed upon developers that developers should post "Not a working day today, I will be back on <date>" when they forecasted an absence. By this way, everyone knew who worked on what at which moment and if anyone was absent. Developers were expected to plan online meetings on their own during the Sprints. They were responsible for helping each other with technical skills.

D. Outcomes

The Product Owner produced a Product Backlog with 45 User Stories. 15 User Stories were completed by the team and accepted by the Product Owner. During Sprint 3 some developers decided to hastily implement an unplanned feature (making the mobile application available in French) and this resulted in problems for the team to merge codes in the repository. The Product Owner had to go to retrieve a working version from the code repository. Videos on software were not prepared for the demos. At the time this paper is written, some developers are still working on improving the application.

The project summary on stories planned and closed, working hours planned and actually spent are shown in Table I. Please note that not all student developers precisely reported their activities by closing work items or correcting the time they spent on tasks. Some developers also worked on tasks unplanned and they did not add them in the work item log. All this impaired the statistical data gathered in RTC.

TABLE I. PROJECT DATA (PARTIAL) SUMMARY

	Sprint 1	Sprint 2	Sprint 3
Number of planned stories	10	18	18
Number of completed stories	1	2	12
Work hours planned	59.25	82	153.5
Work hours done	46.75	77.5	67.5
Work hours done compared to planned	78.90%	94.51%	43.97%

1) Closed stories and product functionality.

As Sprint 1 was mainly used to get familiar with the technologies and processes, the only story that was closed was "Learn Java ME". Sprint 2 was not very productive as the planning and work occurred too late. Sprint 3 dragged most of stories from previous Sprints. 14 stories were closed in total in the last two Sprints. They consisted of the welcome screen, the four-topic list selection menu, 2 questions of mathematics, and all the questions of reading and writing.

2) Work hours

The work hours done compared to the work hours planned in the three Sprints are 78.9%, 94.51%, and 43.97% respectively. The explicit gap of Sprint 3 is largely due to the fact that some developers did not close work items upon completion and the time spent for some tasks was not accounted for in RTC.

The Burndown Chart over the entire project process is showed in Figure I. It shows the difference between planned work hours and remaining work hours, and reflects unrealistic planning estimates as the project goes on. Ideally, the chart should show a trend toward zero hours of remaining work as the sprint comes to an end. This chart does not completely reflect the real situation as many work items were not closed and estimated times were not updated as they should in RTC.

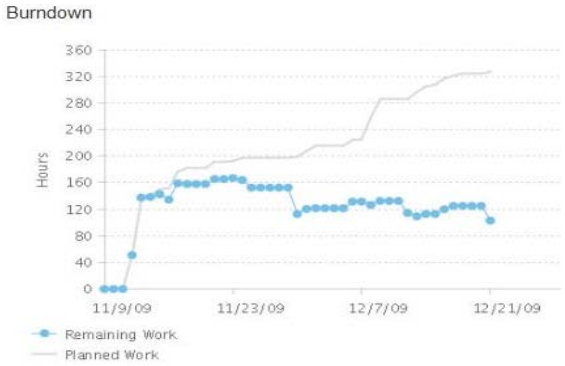


Figure 1. The burndown chart for the entire project

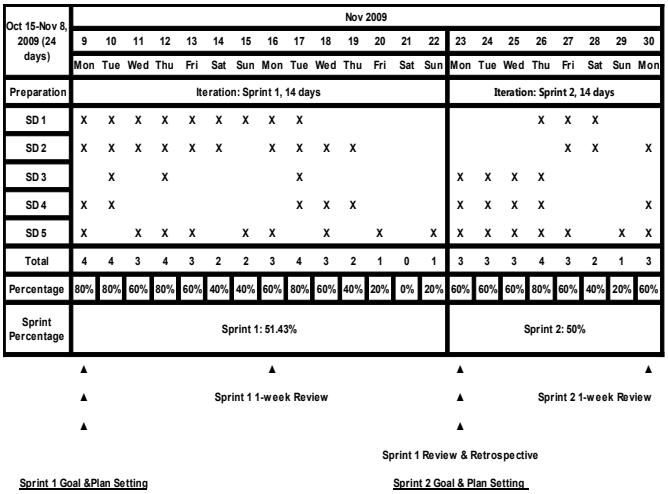
IV. OBSERVATIONS AND ANALYSIS

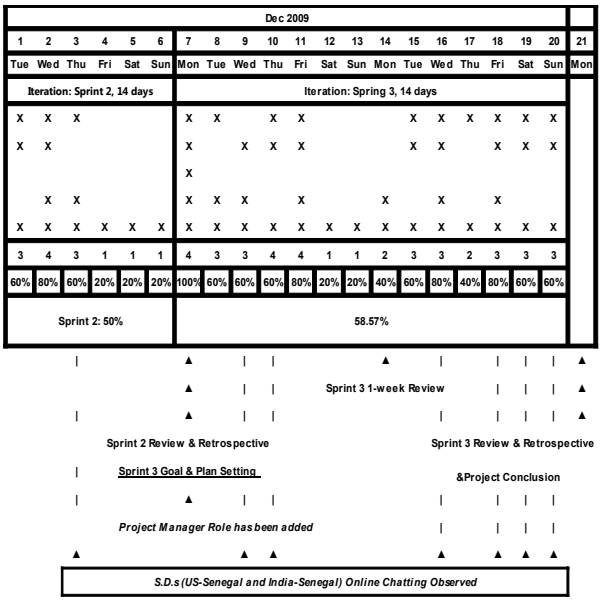
As the only experienced IT professional involved in this project, the student author of this paper has observed an interesting self-learning process experienced by the whole team. This section will look at the Daily Scrum as one of the most important Scrum artifacts and states observations for the three Sprints.

A. Why is the Daily Scrum important?

Within a distributed development team, Daily Standup Scrums become impossible. The time difference, Internet and infrastructure issues (e.g., power cuts) render synchronous daily update meetings difficult too. In this project, developers were required to update their Daily Scrums in RTC at the beginning of their working day and to explicitly describe their past and future activities and their difficulties. From the written Daily Scrums, it was easy to notice if developers were doing something different from what they planned to do. The written Daily Scrums were a simple and obvious way for developers to expose themselves in front of the other team members; via this way, the overall team performance was visualized and the entire team awareness was promoted.

TABLE II. PROJECT TIME LINE AND DAILY SCRUM ACCOMPLISHMENT RATE





Notes :
S.D. = Student Developer
"X" denotes that this SD completed the Daily Scrum at the beginning of the day.

1) Developers can learn about themselves, and their planning, estimating, and executing skills.

The Daily Scrums are instruments that help collect the following information in retrospect: How well do developers plan and then execute their plans? Are they getting more self-disciplinary? Do they understand better their own abilities, strengths and weaknesses? Are they getting more realistic about what they can and cannot do within one day? Are they actually working on tasks that matter and help them achieve the Sprint goal? Daily Scrums permit analysis and reflection. They provide a direct, explicit and easy way for developers to learn about themselves. And by taking reference on others' progress in tasks of similar scale, developers can learn about themselves even better.

2) Developers have to work at a constant pace.

Daily Scrums require constant-pace progress that ensures that the team always has something ready for the Product Owner. They help increase the transparency of the project and prevent unhealthy burn-one-night-for-one-week working styles that are common to students.

3) Developers see where the whole team stands anytime and can help each others in case of absence and difficulties.

Daily Scrums provide everyone with an idea about who is doing what and when. The team thus works as a united entity and work can be reassigned with consent if a member is absent or has difficulties.

B. Observations on Sprint 1

At the beginning of Sprint 1, most of developers were still studying Scrum and unaware of the importance of keeping their own Daily Scrums up-to-date. 51.43% of the Daily Scrums were completed. Some developers skipped days or wrote the activities of several Daily Scrums at once, which did not conform to the Scrum rituals. The quality of Daily Scrums was poor too. Working items were vaguely expressed and did not carry much value (e.g.: "Learning Java ME", "Try to do tasks allocated to me", and "Unstable Internet"). Consequently, problems that impeded progress were not directly presented. Developers did not communicate outside of the planned meetings. The student author, as Scrum Master of this Sprint, was getting used to Scrum practices.

C. Observations on Sprint 2

The situation aggravated in Sprint 2. The Sprint 2 Planning took place during the second week of the Sprint, because of the absence of some developers (e.g., exams and religious holidays). Lots of absences at planned meetings were also noticed during Sprint 2. Some developers disappeared and did not update their Daily Scrums during 10 consecutive days without notice. These absences were not factored in the planning and unavailable in the scheduled absences of the developers in RTC. This increased the frustration amongst developers. The students playing the role of Scrum Master in Sprint 2 did not follow the practices as planned. Compared to Sprint 1, a few more updates of work items were posted in RTC and a few more email notifications of changed items were exchanged. As showed by the retrospective meeting, the biggest problem of the team that became apparent was the lack of communication and coordination across the whole team. Consequently, the software built during Sprint 2 did not integrate the planned User Stories and the team accepted

serious warning from the Product Owner. At the end of Sprint 2, the student author felt deeply that it was time for her to take over the Scrum Master role in the following Sprint. At that time she knew better how to act as Scrum Master and hopefully could help foster communications and make sure developers were in control of the process. With the help of the lead professor and the Process Coach and with the consent of the other developers, the student author became Scrum Master of Sprint 3, in addition to her inherent developer role.

D. Observations on Sprint 3

With the continuous reminders and supervision of the new Scrum Master, the average Daily Scrum accomplish rate increased from 50% in Sprint 2 to 58.57% in Sprint 3. The quality of the Daily Scrums increased too. Developers used more specific wording (e.g., "Learning command listeners in Java ME", "I am going to do Task 223 that is part of User Story 79" and "I have an exam today and tomorrow, will be back next Tuesday"). The team members met outside of the weekly planned meetings. As shown in Table II, there were 7 online chat meetings organized by student developers autonomously in Sprint 3. Note that none of this was observed in Sprints 1 and 2. More documentation (e.g., meeting minutes, chat logs, work items change logs, and email notifications linked to items changes) was made available in RTC compared to the two previous Sprints.

Some of the developers were glad to know that everybody was more involved and wanted to move the project forward. Other developers were not receptive and did not change their behaviors despite repetitive messages from the Scrum Master, lead professor and Process Coach. The author investigated the reason of the unchanged behaviors of these developers after the end of the project in a friendly conversation with them. They reported that they understood very well the principles of Scrum but just lacked motivation to do all this work that they judged administrative. After all, that was a "school exercise" done by the "voluntary contribution of students" (words from a student developer). But, the student author has to state that a project like this provides an unprecedented opportunity for inexperienced students to get their feet wet before going to the workplace and nobody affords to lose it.

V. LESSONS & RECOMMENDATIONS

The student author conducted post-project interviews with the developers and professors as a retrospective analysis of the project. This section will present some educational lessons learned on the project and draw the recommendations for students.

A. ABOUT "KNOWING YOURSELF": TO KNOW AND TO GROW

1) Education and process method: A tool to dig into student's inner side.

There is no better time than school time in one's life to grab and cherish all the possible opportunities and examine one's motivations, passions, strengths and weaknesses. Every software development project created a new opportunity for developers and other team members to get to know themselves better and deeper. Agile Methodologies and Scrum reinforce this idea and particularly emphasize the planning, estimating, and executing abilities of each participant. These are just the essential skills that every one of us need to practice and master, no matter what kind of tasks we need to face and cope with in future's workplace or in life.

2) Motivation is the driving power.

Students got in touch with different roles in a software development project. They could see more clearly what interested them the most and which area they were good at and thus could define a career path that they would likely want to follow (e.g., development or management). There were things they did not like or were not good at but that they were required to follow for the sake of the overall project. We should always be aware of these things. Sometimes we have to compromise: we have to do what we should do, not what we like to do. When working as a team, it is important to think about the big picture.

The student author would like fellow students to stand in their future employers' position and re-think their lack of motivation at some point in the project. Potential employers want to hear what professor says about candidates and like to see samples of their work. This type of school project offers students a great opportunity to create work that they can showcase to potential employers. Students have to understand the importance of necessary compromise in following rules, cooperating, changing behavior when necessary to achieve a common goal and produce good quality work. The student author would like to ask fellow students to keep in mind that professors can be a source of reference letters for them. There is no reason not to take advantage of a golden opportunity like this. After all, *at school, we are taught a lesson and then a test. In life, we are given a test that teaches us a lesson!* (Tom Bodett, American author, 1955-)

3) Work hard and work smart.

Knowing one's strengths and weaknesses, motivations, and passions is not enough. Choosing sustainable working methodologies is also very important. Some developers were so much interested in development that they tried to implement stories that they did not plan and ended up implementing less than expected due to the difficulties of integration with other developers' code. That resulted in the criticisms of the Product Owner. It is always a priority and a "safety protection" to work at a constant pace, to stick to the plan, to report candidly, and to keep everyone healthy. We of course like to work hard but we need to work smart too. This is the mature and balanced manner to produce an outcome more efficiently and more healthily, not only for oneself but also for the team.

4) Choose a depth-led and breath-followed learning path.

As software engineering students and future IT professionals, we should certainly firstly focus on our majors. But we should also take any possible chance to broaden our knowledge by taking courses in other disciplines and participating in different projects while at school.

Entrepreneurial and management skills are very useful in the software development industry.

B. ABOUT "BEYOND": TO INTERACT WITH THE OUTSIDE WORLD AND TO REFLECT ONESELF

1) Knowing others is an inverse process to know oneself better.

Working in a team implies to get to know, understand, tolerate, appreciate, and learn from others. That is an enlightening experience. We can have team members with completely different characters or working styles and we will have to be open-minded and humble, turn inwards, and examine ourselves before drawing any hasty conclusions. Things are not always like how they appear to be and sometimes we are just blocked by our narcissism. Given the rigidity of Scrum requirements, we have to work interdependently and expose our working style in front of others. We get to know ourselves better by mirroring ourselves with others and by analyzing the difference with someone who did better than us: why did I spend nearly twice as much time than him given a similar work item size? Why was my planning quality worse than hers? How can I make an improvement? When will I be able to catch up with them?...

2) Understanding "teamwork" and preparing to help others is a process to grow oneself.

Scrum practices force us to know what the other team members are doing. The developers need to get ready to take complete responsibility of others in case some colleagues encounter difficulties or absences. We learned to shoulder more responsibility to ensure that the team will succeed as a whole. If the team succeeds, everyone succeeds. If a member fails, the only chance for the team to succeed is to ensure that everybody knows what is going on and that the team is able to find solutions to move forward.

VI. CONCLUSION & NEXT STEP

The post-project interviews indicated that all the students had a positive learning experience by going through a steep learning curve in knowing more about themselves, others, collaboration, technical skills, and process method within a short period of time. In the future we would know better how to go about this type of project and how to mitigate problems rather than succumb to them.

Obviously, some of the educational values illustrated in this paper are not new, but Scrum rituals and the obligatory adherence to them help students know themselves in a more direct, explicit and easy way. Scrum can be considered as a good pedagogical exercise and be used in similar projects.

This project brought to the forefront concerns about how to use Scrum efficiently in a distributed development environment and how students can grow as productive members in a global distributed software development team after knowing themselves better. In the next step, we would suggest an approach with dedicated Scrum Masters. i.e., Scrum Masters who do not participate in developers' work, to ensure that accurate data are collected to examine the process more efficiently.

ACKNOWLEDGMENT

The authors would like to thank Dr. Olly Gotel and Prof. Vidya Kulkarni for the invaluable advice and support in the project. The authors also thank all the students involved.

REFERENCES

[1] 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. of the Conference on Software Engineering Education and Training (CSEET 2009)*, Hyderabad, India, February 17 - 19, 2009.

[2] Gotel, O., Kulkarni, V., Neak, L. and Scharff, C. "Working Across Borders: Overcoming Culturally-Based Technology Challenges in Student Global Software Development". *Proc. 21st Conference on Software Engineering Education and Training (CSEE&T 2008)*, Charleston, South Carolina, US, April 14-17, 2008.

[3] Gotel, O., Kulkarni, V., Neak, L., Scharff, C. and Seng, S. "Introducing Global Supply Chains into Software Engineering Education". *Proc. 1st Intl. Conference on Software Engineering Approaches For Offshore and Outsourced Development (SEAFOOD 2007)*, Zurich, Switzerland, February 5-6, 2007.

[4] Gotel, O., Scharff, C. and Seng, S. "Preparing Computer Science Students for Global Software Development". *Proc. 36th IEEE Annual Frontiers in Education Conference. Borders: International, Social and Cultural (FIE 2006),* San Diego, California, US, October 28-31, 2006.

[5] Gotel, O., Scharff, C. and Seng, S. "Incubating the Next Generation of Offshore Outsourcing Entrepreneurs". *Proc. Symposium on Information Technology and Entrepreneurship (ITE 2005),* Oklahoma City, US, April 19-20, 2005.

[6] Highsmith, J.A. *Adaptive software development: A collaborative approach to managing complex systems*. Dorset House, 2000.

[7] Highsmith, J.A., Cockburn, A., & Boehm, B. *Agile software development: The business of innovation*. Computer*, September* 2001.

[8] Yourdon, E. *The 'light' touch.* Computerworld, September 18, 2000.

[9] Fowler, M. Put your process on a diet. *Software Development, 2000*

[10] RallyDev Agile Impact Report, See: www.rallydev.com/downloads/document/103-the-agile-impact-report-proven-performance-metrics-from-the-agile-enterprise.html (accessed March 2010)

[11] IBM Jazz. See: www.jazz.net (accessed March 2010)