Innovation and Scaling up Agile Software Engineering Projects

Sita Ramakrishnan
Clayton School of IT, Faculty of IT,
Monash University, Australia

sita.ramakrishnan@infotech.monash.edu.au

Abstract

Software Engineering capstone projects have been running successfully since 2002 for the final year software engineering (SE) students of the Bachelor of Software Engineering (BSE) Program at Monash University, accredited by Engineers Australia and Australian Computer Society. Agile methods are being increasingly adopted in the industry. In this paper, we describe the objectives of SE capstone projects and report on how our innovative projects for supporting the software engineering projects in undergraduate programs at Monash University have evolved and have been scaled up to support agile SE capstone projects. We detail the evolution from our early innovative software engineering projects in the mid 1990s that have served as catalysts for more innovation in the early 2000, and for scaling up agile SE projects with increasing central technical infrastructure support from the School. More recently, we have adapted our approach with a combination of open-source and commercial tools under academic licence for developing and deploying these projects effectively with agile distributed teams. The paper concludes with a discussion on lessons learnt from our innovative projects in the mid 1990s and from the evolution in scaling up to agile practices for the SE capstone projects from 2002-2008.

Keywords: software engineering, capstone projects, agile practices

Introduction

Innovation in education often works best when it is driven by academic ownership of prototype projects. Innovation in teaching and learning through technology is desirable as it leads to improved learning in students and improved collaboration between students and their teaching staff (Alexander, 2008).

Scalable tools are required to address certain technical problems in general in educational institutions and in software engineering courses in particular. There must be a balance between innovative academic prototypes and centralised scalable tools that may attract better centralised support from the faculty technical services group to avoid tension between technical services and academics.

Wikipedia defines ”scalability” as follows: ”In telecommunications and software engineering, scalability is a desirable property of a system, a network or a process, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged” (Wikipedia, 2008).

Current students need to be trained to work as distributed teams as offshore software development (OSD) is gaining popularity in a number of IT organisations (Petkovic, 2006). Even without OSD, students in the current internet age demand the facility to work on capstone team projects as distributed teams and able to access central server and other resources from their homes. Working as distributed teams from their homes enable them to simulate some of the features of offshore software development and practice the agile method.

Offshore software development is a form of outsourcing which has to take on board the complexity of developing software systems when the contracted consultants reside in different countries (Kornstadt & Sauer, 2007). OSD is becoming increasingly common in industry. Vendors such as IBM Rational are promoting their Jazz technology platform (www.jazz.net) as an extensible team collaboration platform aimed at integrating people, process and assets across software development projects. University researchers in the U.S.A, Canada and Germany have received Jazz grants in 2008 to assist them with preparing student teams' skills in managing projects which are distributed across geographic and institutional boundaries www.thetartan.org/2008/3/24/news/software). CollabNet (www.collab.net) is a widely used collaborative platform (www.tigris.org) that supports globally distributed software development teams. Recently, Chair of SE, Prof Meyer at ETH, Zurich and his group have started offering distributed and outsourced SE projects for student teams over several universities to collaborate (http://se.ethz.ch/teaching/2008-H/dose). They discuss their experience with student teams in multi-site projects between universities as an academic approximation of a globalized software development as practised in the industry (Meyer & Piccioni, 2008).

In this paper, we report on how our innovative projects for supporting the SE projects in undergraduate programs at Monash University have evolved and have been scaled up to support agile SE capstone projects. We report on our work on adapting our software engineering capstone projects to distributed agile teams. Our approach has evolved from our earlier customised work with a project management, code management and process-oriented paradigm to a more integrated open source tool-based approach.

Since 2002, BSE students at Monash University undertake a full year (2 semester) capstone project unit where they work in teams on a large software project for a client. The clients are from the industry or research organisation. The objective of the final year SE capstone project is to expose the students to real-life project scenarios and current (global) SE development practices. The teams of 4-5 Monash final year SE students are formed at the beginning of the year. Each team is assigned a new project with a Monash supervisor and an external client. The student teams are required to follow an agile process and work in pairs for incremental releases. The 4-5 member teams develop an architectural and design model and divide up their tasks taking dependency and subsystem interactions into consideration. They focus on program interface descriptions and learn the importance of such SE principles by doing. The architecture-centric component-based development is especially important in agile projects with offshore setting (Kornstadt & Sauer, 2007).

One of our industry clients for SE capstone projects has expanded his development practices in India and had wanted to institute the offshoring model for the project with Monash SE students in 2008. This is in line with the accrediting body, Engineers Australia's philosophy that educational

institutions must “get closer to industry” as part of engagement with external constituencies, be aware of and be responsive to industry demands for quality SE graduates. Engineers Australia is particularly keen on our Monash University Software Engineering (MUSE) studio project which assesses students’ understanding of the key areas of Software Engineering Body of Knowledge (SWEBOK) (Bourque, 2001). In Australia, you must graduate from an accredited engineering program to be assured graduate membership of Engineers Australia (EA). Assessment for accreditation by EA (http://www.ieaust.org.au/membership/accreditation.html) is based on the curriculum: structure and content, the teaching and learning environment and the quality assurance framework

Our pedagogical approach requires student teams to learn by doing the various key areas of SWEBOK as part of their project as can be seen in the following sections. The team project involves students in skills and knowledge acquisition through technology infusion of collaborative tools in a team-based project setting and through interaction with other students, team members, academics, professional clients and visiting/invited academics. Since it is aimed at the final year of the BSE program, students are more self-directed and a constructivist learning style is promoted via cooperative and sharing tasks with collaborative tools in developing a higher level of critical thinking and learning (Ramakrishnan, 2003). In the next section, we cover the objectives of the capstone project. In the following section, we detail the evolution from the early innovative software engineering projects in the mid 1990s that have served as catalysts for more innovation in the early 2000, and for scaling up agile SE projects with increasing central technical infrastructure support from the School. Then we describe the agile practices that we have adopted (Braithwaite, 2005; & Karsten, 2007), and conclude with a final section with a discussion on lessons learned over the past 7 years from running the SE capstone projects.

Software Engineering (SE) Capstone Projects at Monash University

In this section, we commence with the aims and objectives of the SE capstone projects and describe the role and expectations of the project supervisor(s), clients and student teams. The author as the lead lecturer sources suitable clients. Students form their own teams. They are given short descriptions of the projects and are asked to fill a preference list. Based on students’ capabilities, the author may veto their preference at times. Otherwise, the teams get to choose the projects that they want from the list of projects available. The process of development and assessment is described partly in SWEBOK terms and laid out in the hurdles and milestones tables (Figures 1 & 2).

Final year Software Engineering students are required to undertake a large software project for a client and work in teams. The teams are responsible for their own project management, with guidance from a supervisor. The teams are expected to use proper techniques and tools that enhance product quality including appropriate analysis and design tools, quality reviews, appropriate software testing and a high level of user involvement. The quality of the product is not judged solely on the merits of the software but also is dependent on packaging. Furthermore, documentation must be produced at the appropriate point in the development of the product.

In addition to the products produced by the project, the process by which these products are produced is an important aspect of the subject. These processes relate to how/when/why certain actions are undertaken by the group. It is important for the group to observe their process with the aim of improving it. For example, the meeting process (group meetings, supervisor meetings and

client meetings etc) - the products of these meetings relate to minutes, action lists and information for further work. The process relates to how a project team identified and recorded information, how they involved all people present, why they took a certain approach to an interview and whether it was successful. It relates to the group reviewing not just the products but also reviewing the process by which the product is produced.

The project runs for two semesters, inclusive of all non-teaching periods. Students begin the project in semester 1 and complete it at the end of semester 2 (late February to late October in total). Lectures/Seminars are held during teaching weeks as announced. It should be stressed that these are seminars and not formal lectures and hence groups and individuals are expected to actively participate by contributing experiences and asking questions. Since 2005, industry experts such as from IBM have been invited to run seminars and workshops on specific tools such as IBM Software Architect, Rational automated testing etc. Other seminar sessions cover SE areas such as agile methods, CVS, SVN and other commercial and opens-source tools. Student seminar talks, group presentations, incremental releases (Prototypes) and walkthroughs take place during the scheduled seminar time in both semesters.

The project life-cycle is constrained by a fixed schedule for hurdle times and milestones (deliverables) throughout the year. The roles of students in the project are to be determined by the project team at the beginning of the first semester. For example, each project is expected to assign the role of project manager to one of its members for the duration of at least one semester. The role of a project manager includes keeping track of and reporting the project status, planning, work allocation and time management. The project supervisor (an academic) provides advice on this but is not expected to manage the project. The author is in charge of this subject and has the overall responsibility for the subject. She supervises one of these capstone projects per year, but the remaining projects are allocated to other academics who form part of supervision panels during some of the assessments. Students are expected to deliver their software and software artefacts (requirements, design, test cases, code, documentation etc) according to the milestone schedule published on the subject web pages. After delivery, often the client may request changes. After consultation with the supervisor, some of those changes will be termed ”reasonable changes” which students must complete by an agreed time. Some changes may require balancing the task schedule because of resource and time constraints. Hence students are advised to track change requirements and approval processes thoroughly - best with a formal tracking system - to document agreements with the client.

Role of the Supervisor: Each group is assigned a supervisor for the whole year. They need to meet with the client at least once at the start of the project to form an understanding of the project requirements. The supervisor meets with their student project team each week during semester 1 and 2. The role of the supervisor is to: advise and assist the work flow of the group; review the work produced by the group and suggest changes/improvements as necessary; detect significant deviations from the project scope and time estimates; provide an official high-level liaison point between the client and Monash University; award marks to the group and to group members; if appropriate, the supervisor may: attempt to manage conflict resolution between group members and/or the client; make suggestions about down-sizing the initial project scope and assist with complex negotiations with the client. However, for the most part, client liaison is the group’s responsibility.

Agenda at Supervisor Meetings: Groups meet their supervisor on a schedule agreed on between themselves and the supervisor every week. Each such meeting will have at least the following agenda: record which students attended the meeting as well as any apologies for non-attendance;

review the action list from last meeting, with particular attention to the tasks that were allocated to each group member and discuss the progress of the project. This will include: A general discussion of the current state of the project; reviewing any changes to the scope as agreed with the client; reviewing any documents that are currently due and decide a time for the next meeting (planned at this stage for the same time every week).

Role and Expectation of the Client: The client should provide the student team with a realistic project, which can be completed in the timeframe allowed (12 hours/week per student for 2 semesters of study) for the final year SE (capstone) project. The client's role is to be available on a regular basis, usually once a week for students to elicit the client project's requirement and to check whether the right product has been built and whether the product has been built right. He/she should be honest in the assessment of the student teams' work and whether it meets his/her expectation/needs. The client is also required to conduct beta-testing at their site or elsewhere as agreed to before and sign-off accepting the project before the project is deemed completed by the student team. It is also important for the clients to appreciate that student teams are expected to develop software engineering process skills, learn how to manage group dynamics, manage teams, conduct meetings, time management etc. The team members may also need to learn new technical skills. The client must remember that both process and product aspects are important in the capstone project for student teams to mature into practicing Software engineers and all this takes time. This has been explicitly included as some clients were interested mainly in the product delivered and were not interested in students spending time on process aspects.

Hurdles and Milestones in SE Capstone Projects

Student teams are expected to meet the hurdles and milestones as published on the SE capstone project web page (Figures 1 & 2). Hurdles: Each fortnight, a set of progress summary documents must be prepared for discussion with the supervisor. The other deliverables are as set out in the hurdles table. In the SE capstone projects conducted by the final year student teams in a university, couple of issues such as costs/budget and time limits associated with the project does not reflect reality. In our MUSE capstone projects, student teams conduct the industry client's project for no payment. There is a fixed time limit of 2 semesters of study (late February - end of October) imposed based on the university calendar. A legal agreement is completed for each MUSE capstone project and simulates a commercial consultancy situation for the final year students. The client agrees to provide the project team with access to data and systems in order to facilitate development of the system which forms the basis for assessment for the students. The student assigns to Monash University ownership of all rights in any IP arising from the project including future copyright. The client owns the IP rights, including copyright, and grants the university a royalty free perpetual licence. A revised legal agreement must be used where there is an offshoring component for the project as the offshoring party may be a paid consultant or another student team from an overseas educational institution.

Regular Hurdle Points	Deliverable	Tracked by
Sem 1+2 weeks as announced	Lecture and seminar schedule	Lecturer
Weekly meetings 0.5 hour	Project meetings	Supervisor
Fortnightly odd weeks except week 1 sem 1 and week 13 sem 2	Progress reports	Supervisor
Sem 1 Hurdle Points	**Deliverable**	**Tracked by**
week 1	Individual project preference	Lecturer
week 3	Individual seminar preference	Lecturer
	Preliminary Project Management Plan (PMP)	Supervisor
week 5	Penultimate PMP	Supervisor
	Legal agreements signed by clients and students	Lecturer
	Preliminary Software Requirements Specification (SRS)	Supervisor
week 7	Preliminary Software Prototype	Supervisor
week 9	Penultimate SRS	Supervisor
week 11	Group peer assessment	Supervisor
Sem 2 Hurdle Points	**Deliverable**	**Tracked by**
week 3	Penultimate Software Prototype	Lecturer & Supervisor
week 5	Preliminary Project Management Report	Supervisor
week 7	Preliminary Software Test Suite	Supervisor
week 9	Penultimate Project Management Report	Supervisor
	Penultimate Software Test Suite	Supervisor
week 11	Group peer assessment	Supervisor
	Client acceptance sign-off	Lecturer
week 13	Final Project Web Site (required for grade publishing)	Supervisor

Figure 1 – Hurdles – semester 1 & 2

Sem 1	Milestone	Group Marks	Individual Marks	Assessed by
as announced	1 seminar per student		5	Lecturer, supervisors and peers (weighted)
10	Software Walkthrough	5	5	Supervisor & Lecturer
10-12	Group Presentation of Project	10		Lecturer and supervisors
11-13	Individual SWEBOK Interview		10	Supervisor panel
11	Final PMP and SRS	15		
TOTAL (50)		**30**	**20**	
Sem 2				
10	Software Walkthrough	5	5	Supervisor & Lecturer
10-12	Group Presentation of Project	10		Lecturer and supervisors
11-13	Individual SWEBOK Interview		10	Supervisor panel
11	Final Product and Report	20		Supervisor & Lecturer
TOTAL (50)		**35**	**15**	
GRAND TOTAL (100)		**65**	**35**	

Figure 2 – Milestones – Semester 1 & 2

Individual SWEBOK Interview: One interview is conducted each semester. The interview is a formal exam. It assesses the capability of the student to apply the foundation knowledge of software engineering. The supervisor picks one or two focussed areas randomly and the student being examined recaps the relevant knowledge (very briefly and to the point) and demonstrates deep understanding of its application to the current project. If the project team had good reasons not to apply the knowledge in question, the student being examined must still demonstrate understanding (for example by explaining possible application of this knowledge given appropriate change requests etc). Students must prepare well for the interview in theory and application. The interview also allows the supervisor to gain a better understanding what the student has learned from doing the subject; how well the student worked as a member of the project team.

Each student is assessed individually. The student is asked specific questions with respect to SWEBOK: software process/ product, software architecture, analysis and design methods/tools/patterns, communication with the client and the rest of the team, agile method, team work /environment/collaboration, conflict resolution, SE environment/testing methods and tools, configuration management, product quality, technical skills learned, commitment, consistency and quality of project work, software assets and documentation.

The lead lecturer for the unit (the author) together with the other supervisors review the way the group have managed the project, taking into consideration such things as: group communication, interaction and cooperation, management of deadlines and timelines, and liaison with the client and supervisor for arriving at a Project Management mark.

Risk List: Risk is everybody's business. The risk list is updated by each student project team and distributed to the client, supervisor and coordinator. The risk list contains up to ten problems that the project has or may encounter in priority order (most worrying on top, least worrying on the bottom). Each risk is augmented by the action taken to minimise the project's exposure to this risk. The risk list serves to make team members aware that risks can and should be monitored and managed.

Evolution

Innovation and Scalability with Technical Infrastructure Support

Next we briefly cover an innovative software engineering education research project which was conducted by the author in the mid 1990s to monitor students' software development process in an O-O system and to improve students' learning (Ramakrishnan, 1995). Inspection sessions are crucial for keeping a mutual shared understanding of the state of the project (Bruegge & Coyne, 1994). Inspection process is used to track defects and can be used to arrest any common patterns of errors that may be arising due to misunderstanding of concepts. An important difference between the traditional software systems development and object-oriented systems development is in the area of project management. The recommended incremental iterative cluster based development for O-O software systems forces the development team to move through the various phases of the software life-cycle during the development of a cluster. In this project, students were required to estimate the time required to complete these activities and track the actual time taken so that they can measure the resources spent by various members of a group and also gain appreciation of the difficulty in project cost and time estimation (Zweben, 1992). Graphs of class sizes, classes and methods against time were particularly relevant in this O-O scenario based development where classes which participated in more than one scenario had to be reopened in some cases for adding new methods. These statistics were also used to compare various student groups' performance and help in arriving at a more objective measure for their grades. An empirical method based on statistical principles of formal measurement in a software engineering setting (Basili, 1992) was used for the software measurement experiment.

The experiment in this study involved fifteen student teams (in groups of three) who were required to follow a prescribed process for an incremental, iterative development to implement a Conference Management System. They were required to use the same high level design and produce a low-level design, coding and have organised inspection meetings to review the team members' work, document the defects using the given defect types (Strauss & Ebenau, 1994). The test plan covered black-box or specification testing, program testing, white-box and integration testing. The inspection process was used to detect, record and correct the errors captured at various phases of the software life-cycle. Inspection process was used to gather statistics about the time taken to detect and record the errors. Testing captured the errors that went undetected during inspection. The emphasis of the process model prescribed for this measurement experiment was on producing charts of project management information, creating test plans that included Class (program) and integration (system) testing, and reporting various defect types found during inspection. A combination of process and product assessment (Bache & Bazzana, 1994) measurements was used in that experiment. The project management metrics collected during low-level design, inspection and testing of classes and methods were analysed using graphs. The software products were assessed by analysing the defects recorded according to various defect types during inspection and testing,

and by using the quality criteria of Boehm (Basson, Haton & Derniame, 1993, and Meyer, 1988). This experiment showed that O-O development can benefit from a formally organised process framework that cuts across the various aspects of software life cycle from planning to implementation, testing and documentation. The rate of tracking defects is an important quantifiable measure of software quality (Booch, 1994). The data collected about defects were used to measure the defect density in terms of the number of defects per class development. The feedback sessions with students were then used to discuss relationships between class sizes, complexity of a class, its methods, class development time and the number of defects found. These sessions were also used to overcome any common misunderstanding that was made visible by this detailed monitoring/feedback process thereby empowering students to achieve an improvement in the quality of the product. This experiment showed that a team based object-oriented development will definitely benefit from an organised inspection process and cost less resources to develop the system. It demonstrated the improvements that can be gained by following a prescribed O-O process product framework at a fine granularity by measuring the defect rates and defect types across OOSLC, estimation of time & effort of teams and team members.

In summary, a repeatable process experiment was set up to assess students on their O-O project. The process required student teams to work on a scenario at a time and work in an incremental iterative style of development. The students collected data about their development process and the product quality using the templates given for inspection and testing. The data across the student teams were compared using graphs to show differences visually and used in class to discuss bottlenecks and outliers. One of the important benefits of the introduction of a repeatable process in an educational context was to make the process aspects visible and available for comparison across student teams more objectively than it has been possible in the past.

However, the data collection for inspection and testing process was a manual paper template based approach by the student teams and the data analysis was conducted by the academic using excel spreadsheets. Students in the web age are reluctant to use paper based data collection methods and prefer computer based data entry where possible. So, we developed an effective online measurement system for OO Software Process, Product and Resource Tracking (Ramakrishnan, 1996). Since that system was a customised development using technologies which is now dated, it is no longer used. Also, since it was a small innovative project, it could not be morphed and scaled up to new technologies as funding was not forthcoming for a technical research assistant role. However our students in 1995-1997 did benefit from these innovative projects.

In 1998, Monash University commenced the Bachelor of Software Engineering Program and the first crop of BSE students graduated from Monash in Dec 2002. One of the subjects in the final year of BSE is a capstone software engineering project. It was run for the first time at Monash in 2002. We have sourced about new industry projects for all our student teams since 2002 for the BSE capstone projects and the final year students have successfully completed these projects. We have developed innovative projects (Ramakrishnan, 2003, 2006; Ramakrishnan & Cambrell, 2002) for exciting the students to learn and effectively practice software engineering method. During 2001-2003, we acquired funding for a dedicated Monash University software engineering (MUSE) lab for the capstone project students. Apart from the standard software/hardware infrastructure support from the School, the MUSE lab was also equipped with the latest commercial testing tools under academic licence.

In 2004-2005, we developed an innovative MUSE portal to enable student teams to record the time spent in the various phases of the project. MUSE Portal provided services that assist team based collaboration, such as a task time tracker to aid in project planning and logging time. The

coordinator or lecturer allocated the task tracker service to each student group, and also assigned release dates for the project's incremental releases. Only students in that group could access this instance of the task tracker. Students were required to log time against tasks that they did in the various phases of the project. The task tracker service tracked each student's log and student groups, and the lecturer could track the project status via the Portal. The service allowed one to filter the information by phases, eg. list analysis phase by tasks, milestones etc. The task tracker service also produced student groups' task logs as a graph, and provided a structure whereby progress in a project could be measured and monitored. In the capstone group project, a user management service was available to the academics (lecturers, coordinators and tutors) who were in charge of the subject. This service enabled the academic to populate the Portal with information about students who had enrolled in the subject. Once the Portal was populated with student enrolment information, students enrolled in that subject were authorised to use the related task tracker service. Student services were partitioned so that they could only view their project groups' data. By providing targeted software services such as task tracker with appropriate access control rights on resources required for the subject, students were able to gain experience in a professional /academic setting. Once the students enrolled in that subject were authorised to use the related task tracker service, they were able to manage, evaluate and reflect on their individual and group work. This improved their learning, helped in delivering better quality work on software engineering studio projects.

In 2002-2005, the MUSE lab's specific server requirements for handling team projects had to be supported from the funding received for these innovative projects. Since the technical services in the School were not adequately resourced, there was no timely support for setting up servers and associated tools centrally to scale up agile projects for distributed teams.

Since 2006, we have attempted to adopt a scalable distributed team software process with tool support and agile method for the management, development and delivery of BSE capstone projects to the industry clients. However, the agile software management was not consistently and uniformly handled by the different student teams. This has been scaled up and has improved in 2008 because there has been very good central support from technical services in the set up of SVN, Trac and other resources residing on the server specifically for the capstone projects. The student teams in 2008 also know that incorporation of SVN/Trac as part of their agile software configuration management is an important part of their assessment.

The next section commences with a summary of Agile methods followed by a discussion in detail how software development life cycle practised in the capstone project has evolved over the last seven years.

Agile Methods

The agile software development manifesto published by Beck et al. (2001) and other work such as Highsmith and Cockburn (2001) stressed agility values such as: " individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan". They proposed 12 principles of agile software which arise out of the prioritized agile values listed in the manifesto.

Abrahamsson et al. (2002) and Chen et al. (2004) stated that software development method can be deemed as agile when the development is done in small incremental releases with customers and developers cooperating and working together, the method is easy to learn and is adaptable.

ASD (Adaptive software development), AM (Agile modeling), XP (Extreme programming), FDD (Feature driven development), PP (pair programming), RUP (Rational software development process) and Scrum are some of the agile software development methods and each of these methods address the development problems from different perspective as the names suggest.

XP (Extreme programming) evolved from issues with long development cycles in traditional software development method, and some key principles and practices were put together to form the XP method (Beck, 2000). The life cycle of XP consists of five phases: exploration (with customer stories); planning (with stories for next iteration, priorities and effort estimates); iterations to release (with continuous review & continuous integration of code base with pair programming including analysis, design, planning for test & testing); productionizing with small releases with customer approval; maintenance and death (Abrahamsson et al., 2002; Beck, 2000). Feature driven development (FDD) is an agile approach but focuses only on the design and development phases (Abrahamsson et al., 2002).

In 2002, we proposed a middleweight process for engineering flexible and adaptable software (Haider, 2002). We observed that organizations used heavyweight and traditional software process methods such as Rational Unified Process (RUP), Catalysis, Waterfall and spiral methods. Such methods inhibited flexibility in software development. Agile method was seen as offering solution to this problem. However lightweight agile methods such as XP lacked support for planning. We compiled a middleweight process that incorporated agile notions, Kobra method and the component paradigm to produce flexible agile software. Kobra method embodies some of the principles of agile modeling (Atkinson et al., 2001).

Rational Software Process (RUP) was not considered agile as it contained detailed guidelines for the process phase (Abrahamsson et al., 2002, Cohen 2004). However, more recently, the adoption phase has been revamped (Abrahamsson et al., 2002, Ambler, 2008) and agile modeling is used in RUP for business modeling, requirement definition and analysis and design phases. Agile modeling by Ambler focuses on 11 best practices listed in 4 categories of iterative and incremental development, teamwork, clarity and validation.

Ambler (2008) discusses scaling up agile approaches and looks at team size, geographical distribution, complexity of systems, compliance requirements etc. He has mapped the RUP process of inception, elaboration & construction, transition and production to agile terminology and practices. He provides detailed strategies for scaling up agile method by including the full system development life cycle, agile model driven development and incremental independent testing.

The term ‘scrum’ comes from a strategy used in the rugby game “to get an out-of play ball into the game” with teamwork (Schwaber & Beedle, 2002). The scrum approach has been developed to manage the system development process. It includes the notions of adaptability and productivity. It focuses on how team members should operate to produce a flexible system in a changing environment. It considers things such as requirements, time frame, technology and resources that are likely to change during the process, and requires a flexible process to be able to be responsive to changes. Scrum also includes management activities to identify and improve existing engineering practices, e.g. in testing. Scrum process has three phases: pre-game with planning and architecture/high-level design; development phase with sprint (agile) practices such as sprint planning meeting to i) decide on the goals and functionality of the next sprint with customer, product owner and scrum team, and ii) meeting with the scrum master and the team to look at how the incremental work has been implemented; and post-game phase of integration, system testing and documentation. The system is developed in Sprints during the development phase. Sprints are it-

erative cycles where incremental development of functionality occurs. The architecture and design evolves during Sprints. Each sprint includes the phases of requirement, analysis, design, evolution and delivery. Schwaber and Beedle (2002) have identified six roles in scrum process and practice: scrum master, scrum team, customer, management and product owner selected by scrum master, customer and management. Scrum provides project management framework which when combined with XP makes XP scalable to larger software projects.

Practice of Distributed Team Software Process and Agile Method at BSE Program at Monash

Since 2002, Monash University Software Engineering (MUSE) Studio lab is equipped to access commercial tools with academic licences and open source tools in SE development environments, methods, testing and configuration management. Student teams are required to use these SE tools and follow agile processes such as pair programming, collaboration, teamwork in SE process and product development, and testing, and for continuous integration and incremental software releases. The studio is used for practical applied SE tasks and for preparing students as professional software engineers in the IT industry locally or as contractors in the globalized workforce.

In 2002-2003, our emphasis was on using a tailored middle-weight process which requires students to devote enough effort in requirements and design models and move more to the agile method during implementation and testing. However, their SE product and process artefacts were not accessible uniformly in a virtual environment.

In 2004, a service-oriented portal was built for the capstone project teams (Ramakrishnan, 2006). It acted as a front end for exposing services such as task tracking and project planning for these teams. Support infrastructure services of user authentication, role and service allocation for members etc were managed through the portal environment.

Since 2006, general discussion forums for the project teams are held in the Blackboard site for the project unit. Wiki page, google docs etc are also used by some teams. Google group was used by one team in 2008 for their project team discussion and email. Also, since 2006, we have adopted various open source tools for students' use in the capstone projects. Student teams have used Trac (http://trac.edgewall.org) which is a web-based software project management and bug/issue tracking system. Trac also provides an interface to Subversion and an integrated wiki. In 2006, we moved from using CVS version control system to SVN and started migrating from the custom-built Portal to the school's servers and repositories for storing project and process assets as the Faculty restructuring resulted in some consolidation of technical infrastructure support. Also, Trac, SVN and other tools including commercial tools under academic licence that are used currently are very well suited for our capstone projects in an offshore setting. We are currently working with more integrated open source collaborative development platforms that support distributed teams and are also exploring platforms and tools that foster collaborative software development such as IBM Rational's Jazz technology platform and CollabNet tools to support our requirement to support OSD in capstone SE projects for Monash BSE program.

Agile Method and Software Configuration Management

Since 2005, BSE capstone students have been using Open Source tool, Eclipse that supports agile methods. One of the useful features in Eclipse for XP is automatic refactoring. This enables students to rename classes and methods easily. The student teams are able to take advantage of the

refactoring feature alongside the versioning tool. Since the concurrent versions system (CVS) does not support some of the refactorings in an XP set up, we have adopted Subversion (SVN) since 2006. Some of the other benefits of using SVN are: easy installation process, repositories viewable through a web browser, and binary files able to be version controlled using SVN's internal binary diff methods. Also, with SVN, we have one version per commit which means that all the files committed together receive the same version number (Eliasson, 2005). Student team members find this very useful with Eclipse/Subclipse to see at a glance which files were changed during a commit. Some student teams use this SVN feature (one version number per commit) to commit files on a story basis and benefit from the traceability provided. Most teams however seem to prefer to split the story into tasks and commit a task at a time, thereby achieving task based versioning.

In our innovative projects in 1995 (Ramakrishnan, 1995), we used Word documents and Excel spreadsheets to track bugs. These are inefficient and more error prone than automated issue-tracking. Trac, an open-source automated issue-tracking system takes care of raising, managing and fixing issues. Trac is light-weight compared to Bugzilla which is another popular Open-source issue tracking solution. Trac provides seamless integration between issue and release management and source code repository and agile team communication approach. Trac's wiki syntax enables the student teams to reference Subversion revisions, files and change-sets. So, a comment in an issue can link to a particular Subversion change-set (or revision number). SVN with Trac is therefore particularly useful for SE team projects.

Since 2006, use of SVN has been standard practice for capstone projects. However, Trac tool has not used consistently by all the student teams. They were mainly using SVN for version control and not in the fuller context of agile software configuration management. This has also been partly due to the technical services not making the SVN server with Trac available to the capstone groups in the past years from early semester 1 in a 2 semester project. In 2008, all the student teams have used SVN with Trac and have reflected positively on the benefits.

Software configuration management can be considered from 2 perspectives: 1) customer focussed traditional configuration activities such as: identification, control, status accounting and audit, and 2) developer aspects of SCM such as, version control, build management, concurrency control, change management and release management (Asklund, 2004, & Eliasson, 2005). As can be seen from the hurdles and milestones prescribed for the capstone projects (Figures1 & 2), the student teams are expected to follow XP practices of pair programming, collective ownership of code, continuous integration, incremental releases and planning games. The collective ownership means that there is parallel work, and 2 pairs in a 4 member team may be changing the same code at the same time. Copy-merge work model is practised to synchronise developers. Continuous integration required the teams to integrate changes into the common working (production) code once a task or story is done. As part of the integration process, regression tests must be done to ensure that the new code did not break the system. Only then, the new code is released and integrated into the production code. Incremental releases are used as a vehicle for showing what has been produced by the team. These are demonstrated in the software prototype and walk through sessions (see Figsures 1 & 2). After refactoring, unit tests must be rerun to check that the code still works. Student teams did not refactor much as they were worried about the increased risk for merge conflicts when refactoring along with other changes in a parallel fashion.

Planning game in XP is about scheduling aspects in an XP project. The industry client provide the requirement stories and resources (time) and the development team estimate the teams's time and risks in implementing the requirements. The game gets executed based on the game plan if parties

agree or some requirements get shifted to the next iteration. Since the capstone project is a software engineering project, we are interested in not just an XP approach, but an agile software configuration management (SCM) approach. This means that the traditional configuration management aspects such as configuration identification which are not part of core XP should also be covered as part of SCM in capstone projects. Configuration items such as source code, stories, unit tests and acceptance tests are items that may change and are therefore stored in the repository. As part of configuration control, stories in XP are prioritized and planned during the planning game which takes the role of change control requests in SCM. Traceability between source code changes and the stories is handled by version control tool, and in our case, SVN. Configuration status accounting is taken care of by the tracker which keeps track of stories and tasks and their progress status which is used by the student team and the supervisor. Configuration control and configuration status accounting assist the team in seeing what changes are currently implemented and by whom. They can offer further help to team members by including proper commit comments such as what changes were made and why.

The Scrum process is an incremental process of software development and is often used with agile software development (Schwaber, 2004). Scrum includes a number of practices and a set of roles. Scrum master tracks tasks and resolves any obstacles in achieving the goals set. A project manager is responsible for project planning, resource allocation, process, deliverables etc. In some organizations, a project manager gets trained for a scrum master role. The main roles in Scrum are the Scrum master, product owner and the team. Scrum master is similar to a project manager, the product owner looks after the stakeholders' interest and the team includes the developers. Scrum master is responsible for the scrum process and makes sure that the team uses agile practices. The scrum has three phases: pre-game which includes planning and high-level design, mid-game which involves sprint cycles and post-game of closure activities when the product development has been completed and the product is released. A scrum sprint cycle is normally an iterative cycle of 1-4 weeks. The product is developed in a series of iterations or sprints. Before a sprint is started, a planning meeting is held to decide on the features to be implemented in that sprint. The sprint has four steps of: developing the product (implement, test & document), wrapping the work for integration, reviewing the work in this sprint, and adjusting for changes in the plans or requirements. In a sprint cycle, Scrum master holds a 15 minute planning meeting each day to ask 3 questions about accomplishments since the last meeting, to be accomplished by the next meeting and any obstacles in reaching the goal. Then the development is carried out. A sprint is closed with a review meeting where the new functionality and progress made since the last sprint is demonstrated. The sprint cycle is repeated until the development is finished. A product backlog and a sprint backlog are also parts of the sprint cycle, and contain a prioritized list of features requested by the customer, and features to be done in the current sprint respectively.

The scrum process has been adapted in the capstone projects to fit within the constraints imposed by the academic environment. For example, students have other subjects that they are studying as well as part of the final year, and they do not meet face-to-face every day with the academic of this unit or with their team members. In the capstone projects, the author (as the lead academic for the subject) has taken the role of the scrum master in the first semester and held 15 minute planning meeting once a week during the lecture/seminar session. Once the student teams have released the first iteration of their project (sprint cycle), the project team leader in the student teams take on both the scrum master and project manager role. A prototype demo in week 5 of each semester is followed by a walk through and demo in week 10 of each semester (see Figures 1 & 2). These correspond to the incremental releases and sprint cycles (5 week iteration) in the

capstone project. We use both the traditional names and the agile terminologies to make sure that the students are familiar with both.

Conclusions

As described in an earlier section, the author set up a software engineering measurement experiment in 1995 where 15 student teams (in groups of 3) were required to follow a prescribed process for an incremental, iterative development for an O-O system.

The objective of the project was to improve the quality of the process and product by measuring the activities of a scenario based development of an O-O system by various teams. As part of the experiment, students were given a plan of activities to follow so that product quality improvements can be made in the context of a clear process model. Student teams received procedures to be followed in an inspection process with template documents of inspection meeting notices, defect analysis, summary and management report. Regular inspection meetings were held by the teams where defects found were recorded based on various categories and graphs produced to show the time spent versus defects found during inspection and also during testing. Graphs to show class size versus time and the number of methods developed versus time were also produced.

Even with one experiment, it was possible to detect some trends. The whole experiment rested on students' cooperation. One group who had not had regular inspection meetings owing to time table clashes and other constraints of the team members had relied on testing alone to detect errors. Their work was of a lower quality and compared to other teams, they had used more development time. The scenario based approach is an incremental iterative approach in system development and a collaborative team project requires the team members to have a shared understanding of the system. The experiment showed that a team based O-O development will benefit from an organised inspection process and cost fewer resources to develop the system. We had a very positive response from the teams during formal student evaluation report for the subject.

The lessons learnt from this experiment are relevant also in the agile context. Even with tool support for recording defects etc, the teams need to communicate and collaborate effectively for common code ownership and other development and configuration related activities. As we have seen in earlier sections, agile method with tool support is necessary to sustain and scale up the software process for distributed teams.

Next, we provide a subjective evaluation for the Software Engineering capstone projects undertaken from the supervisor and students' perspective over the past seven years. In 2002-2003, we introduced a middle weight process that student groups followed in their full year SE Capstone project. Capstone projects in 2002 were aimed at supporting XP SE practices, such as pair programming, collaboration, teamwork in SE process & product development & testing and continual integration and incremental software releases (Fowler, 2000). Also, using IEEE Standard, IEEE 1074 (IEEE, 1998) or Team Software Process (Humphrey, 2000) for the Capstone project would have required our student groups to follow a heavy-weight disciplined process. Pure extreme programming (Beck, 2000) was seen as light weight and without a disciplined process, capstone projects could have deteriorated to ad hoc development. So, the emphasis in 2002 was on using a tailored middle-weight process which required students to devote enough effort in requirements and design models and move more to the agile method with automated unit testing and version control during implementation. Introducing a process culture and matching the process weight to the student groups' abilities and tolerance level was seen as important to successful outcomes in capstone projects. By monitoring and recording problems/successes with the process used in 2002, it

was found that some of the process tracking needed to be tightened to ensure that the student teams manage their time better in the project.

In 2004, an enhanced MUSE Studio Lab was created with more hardware/software resources and a service-oriented portal for student teams to record the time spent in the various phases of their team project. Prior to this Portal service, students kept track of their project tasks as there was no online central repository set up for the teams. Also, since the academic was not able to track the teams' tasks online, there was a time lag in managing this process. During 2004-2005, the task tracker service provided by the Portal streamlined and improved data collection, reporting and sharing of project life cycle information. This ability to monitor and share detailed project status information enabled student teams to manage their project resources more effectively. For example, they were able to monitor tasks on-line, see which are slipping from the schedule, and thus allocate appropriate resources to ensure its completion. Since it is an e-service, the lecturer/academic was able to keep track of student teams on-line and offer appropriate advice as needed. Other services such as file management, CVS etc were also available on the Portal server as project team accessible assets.

However, collaboration features such as issues, bug tracking and revision control were not available as integrated portal services. Also, the lecturer/supervisor had to do the set up each year and populate the Portal with information about students who had enrolled in the subject. Once the Portal was populated with student enrolment information, students enrolled in that subject were authorised to use the related task tracker service. The MUSE servers had to be managed by the academic.

Since the University and our Faculty of IT started using WebCT and now Blackboard (2006-current), student enrolment information in a subject is captured by central IT service department and no longer need to be done by the academic. Also, since the beginning of 2008, the school's technical services have been resourced to assist with the provision of servers and necessary tools such as SVN, Trac etc. for the MUSE capstone projects, the agile software configuration are able to be managed consistently across the various teams.

We have shown a continually improving, aligned quality product and process in place for the SE capstone projects. Our students have always rated the MUSE Studio project very highly as they are given the opportunity to practice the SE skills they learn in the four years of study on a team-based project for an industry client. They also often get to learn new technology in implementing the solution in a real-world setting. This kind of collaborative industry relevant project in the final year of SE education addresses some of the concerns of Computer Science education expressed by some academics (Arora & Chazelle, 2005; Narasimhan, 2006).

In conclusion, we have described an evolution of how the capstone projects have been able to grow the agile process with focus moving from XP/pair programming, automated testing and incremental releases in 2002-2003 to task tracking and CVS/version control as well in 2004-2005, and on to a more integrated approach with tools support including SVN/Trac in 2006-2008 for agile software configuration management as well.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: Review and analysis*. VTT Publications 478.

ACM/IEEE Computing Curriculum. (2003) *Software engineering*. Joint IEEE Computer Society/ACM Task Force on Computing Curriculum, July 2003.

Alexander, J. (2008). Towards an understanding of scalability, innovation and academic culture. *International Journal of Technology, Knowledge and Society*, *4*(3),159-168.

Ambler, S. W. (2008). Agile software development at scale. In B.Meyer, J. R. Nawrocki, & B. Walter (Eds), *2nd IFIP TC 2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2007*, Poznan, Poland, Oct 2007, LNCS 5082, pp. 1-12.

Asklund, U., Bendix, L., & Ekman, T. (2004). Software configuration management practices for eXtreme programming teams. In *Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques - NWPER'2004*, Turku, Finland, August 17-19, 2004.

Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Lagua, R., et al. (2001) *Component-based product-line engineering with UML*. Component software series, Series Editor: Clemens Szyperski.

Arora, S., & Chazelle, B. (2005). Is the thrill gone? *Communications of the ACM, 48*(8), 31-33.

Bache, R., & Bazzana, G. (1994). *Software metrics for product assessment*. McGraw-Hill.

Basili, V. R. (1992). The experimental paradigm in software engineering. In H. D. Rombach, V. R. Basili, & R. W. Selby (Eds.), *Experimental Software Engineering Issues: Critical Assessment and Future Directions, International Workshop, Germany*, LNCS 706, Springer-Verlag, pp. 3-12.

Basson, H., Haton, M. C., & Derniame, J. C. (1993). Use of quality characteristics graphs for a knowledge-based assistance in software quality management. *Proceedings of Software Quality Management, Elsevier Science and CMP*, Southampton, pp.807-818.

Beck, K. (2000). *Extreme programming explained*. Boston: Addison-Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., et al. (2001). *Manifesto for agile software development*. Retrieved Feb 13, 2009, from http://AgileManifesto.org/

Booch, G. (1994). *Object-oriented analysis and design with applications*. Readings, MA: Addison-Wesley.

Bruegee, B., & Coyne, R. F. (1994). Teaching iterative and collaborative design: Lessons and directions. In J L Diaz-Herrera (Ed.), *7th SEI CSEE Conference, San Antonio, Texas, USA*, LNCS 750, Springer-Verlag, pp.411-428.

Bourque, P., & Dupuis, R. (Eds.). (2001). *A guide to the software engineering body of knowledge* [trial edition]. Los Alamitos, CA: IEEE CS Press.

Braithwaite, K., & Joyce, T. (2005). XP expanded: Distributed extreme programming. *Proceedings of the 6th International conference on Extreme Programming and Agile Processes in Software Engineering*, vol. 3556, pp.180-188, Sheffield, Springer, June 2005.

Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. *Advances in Computers, 62*, 1-66. New York: Elsevier Science.

Eliasson, M. (2005). *Advantages of SVN over CVS when working with XP Projects*. Available at http://www.cs.lth.se/home/Lars_Bendix/Research/ASCM/In-depth/Eliasson-2005.pdf

Fowler, M., & Foemmel, M. (2000). Continuous integration. Available at http://www.martinfowler.com/articles/continuousIntegration.html

Haider, Z. (2002). *A middleweight process for engineering flexible and adaptable software*. Bachelor of Software Engineering Honours' Thesis (supervised by Dr Sita Ramakrishnan), Melbourne, Australia: School of Computer Science & Software Engineering, Monash University.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business innovation. *Computer, 34*(9), 120-122.

Humphrey, W. (2000). *Introduction to the team software process*. Boston: Addison-Wesley.

IEEE. (1998.) *Std. 1074-1997. IEEE standard for developing software life cycle process*. Piscataway, NY.

Karsten, P., & Cannizzo, F. (2007). The creation of a distributed agile team. *Proceedings of the 8th International conference on Agile Processes in Software Engineering and Extreme Programming*, Como, Italy, June 2007, pp. 235-239, LNCS 4536, Springer.

Kornstadt, A., & Sauer, J. (2007). Tackling offshore communication challenges with agile architecture-centric development. *6th Working IEEE/IFIPConference on Software Architecture WICSA2007,* pp. 6–9, Mumbai, India, Jan 2007.

Meyer, B. (1988). *Object-oriented software construction*. Hemel Hemstead: Prentice-Hall.

Meyer, B., & Piccioni, M. (2008). The allure and risks of a deployable software engineering project: Experiences with both local and distributed development. *IEEE 21st Conference on Software Engineering Education and Training (CSEET 2008)*, pp. 3-16, Charleston, S. Carolina, April 2008

Narasimhan, L. V. (2006). A second opinion on the current state of affairs in computer science education – An Australian perspective. *Issues in Informing Science and Information Technology, 3,* 445-458. Available at http://informingscience.org/proceedings/InSITE2006/IISITNara114.pdf

Petkovic, D., Thompson, G., & Todtenhoefer, R. (2006). Teaching practical software engineering and global software engineering: Evaluations and comparisons. *Proceedings of ITiCSE200*6, June 2006, Bologna, Italy, ACM Publ.

Ramakrishnan, S. (1995). An ongoing experiment in OO software process and product measurement , *In Proceedings of the International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific)*, Prentice-Hall, November 1995, pp.91--105.

Ramakrishnan, S. (2003). MUSE studio lab and innovative software engineering capstone project experience. *Proceedings of ITiCSE2003*, Thessoloniki, Greece, ACM Publ., June 2003.

Ramakrishnan, S. (2006). A service-oriented portal for software engineering education. *Proceedings of the International Conference on Computers and Advanced Technology in Education (CATE 2006)*, pp. 4-6 Oct 2006, Lima, Peru.

Ramakrishnan, S. & Cambrell, A. (2002). An in-forming web-based environment for a bachelor of software engineering degree – DoIT. *Proceedings of the Informing Science and IT Education Conference - InSITE 2002*, Cork, Ireland, June 19-21, 2002. Retrieved from http://proceedings.informingscience.org/IS2002Proceedings/papers/ramak053infor.pdf

Ramakrishnan, S. & Menzies, T. (1996). An ongoing OO software engineering measurement experiment. *Proceedings of the Software Engineering: Education and Practice (SE:E&P96)*, IEEE Computer Society Press, January 24-27, 1996, pp.160-165.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River, NJ, Prentice-Hall.

Schwaber, K. (2004). *Agile project management with Scrum*. Microsoft Press.

Strauss, S. H., & Ebenau, R. G. (1994). *Software inspection process*. McGraw-Hill.

Wikipedia. (2008). *Scalability*. Retrieved Feb 13, 2009 from http://en.wikipedia.org/wiki/Scalability

Zweben, S. (1992). Effective use of measurement and experimentation in computing curricula. In H. D. Rombach, V. R. Basili, & R. W. Selby (Eds.), *Experimental Software Engineering Issues: International Workshop, Germany*, LNCS 706, Springer-Verlag, pp. 247-251.

Biography

Sita Ramakrishnan is a senior academic in the Clayton School of IT, Faculty of IT, Monash University, Australia. She holds a PhD in Validating Interoperable Distributed Software and Systems. She has active research interests in modeling and validation of distributed software components, component-based and service-oriented architectures and testing, web technologies in education, teaching and learning. She has published refereed papers in International Journals & Conferences on software engineering on quality, reuse, software metrics, evaluation, testing and SE Education. She has been an organizing and Program committee member of a number of International conferences and reviewed a number of conference and journal articles. She has played a leading role in the curriculum development of Bachelor of Software Engineering course at Monash University. She is Director of Software Engineering degree program in the Faculty. She managed the process of formal accreditation of the software engineering course program by the Institution of Engineers of Australia and Australian Computer Society. Dr Sita Ramakrishnan is a member of IEEE.