On Integrating Architecture Design into Engineering Agile Software Systems

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

sita.ramakrishnan@infotech.monash.edu.au

Abstract

Agile software system development approaches have become popular since the late 1990s. Agile method has been increasingly adopted by big players in software industry such as IBM, Microsoft, Nokia and Philips with a view to improving quality and productivity. Such quality improvement goals must be measured during system development to validate the approach, and there is a need for more qualitative and quantitative studies in Agile development methods. Literature study shows that mainly XP approaches have been explored in empirical studies with reports on students' perceptions of XP in university case studies or with software development professionals. Management-oriented approaches, such as Scrum, and scaling up of the method using Agile architectures still require more detailed empirical study and evaluation. In this paper, we report on the evolution of our approach from Agile/XP, Agile/feature-driven, Agile /Scrum to Agile architecture/Scrum in the final year software engineering student project unit, and students' and supervisors' perceptions on quality and productivity from 35 student team projects sourced from the industry over eight years.

Keywords: Agile architecture, Scrum, Agile methods and techniques

Introduction

The ACM/IEEE Computer Science/Software Enginnering curriculum (2003) lists agile concepts and practices such as test-driven development, refactoring etc as core topics to be included in software engineering courses. European Agile researchers such as Abrahamsson are working towards a IEEE standard on Agile methodology – IEEE 1648 (Abrahamsson, 2009). The importance of linking architecture with quality goals is well researched and practiced in building large complex domain-specific software systems and these researchers view scalability of Agile approaches without any focus on architectural issues is not viable. Architectural design at various levels of abstraction in conjunction with user stories in an Agile method under Scrum tactics can be used to address the quality criteria as well as log time taken for story points, tasks and defects. However, the advocates of Agile approaches are wary of the value of big upfront architectural design and evaluation to the customers of the system. Researchers in software engineering method and architecture field are of the view that re-factoring on a large scale may impact on quality with

increased defects. Although technical, project management, communication and cultural issues are being addressed in the Agile development research and practice, more empirical research needs to be conducted into developing solid body of knowledge for integrating architectural design into Agile development.

We have been using Agile methods in the final year software engineering project unit since 2002 (see Tables 1-4). Our recent focus has been on instilling the importance of architecture design in agile development. Although a number of empirical studies exist that explore the benefits of XP approaches, empirical methods exploring the benefits of including architectural design in agile methods are still immature. In the next section, we look at a study which reports on better outcomes by using a specific combination of Agile methods and techniques (Parsons, Ryu, & Lal, 2007), and on recent Agile work on Scrum, Agile architecture and empirical studies (Abrahamsson, 2009; Abrahamsson, Warsta, Siponen, & Ronkainen, 2003; Ambler, 2007, 2009; Buglione & Abran, 2007; Dyba & Dingsoyr, 2008, 2009a, 2009b; Erdogmus, 2009; Evans, 2006; Hadar & Silberman, 2008; Hoffmeister et al., 2007; Kitchenhaum et al., 2002; Knoernschild, 2009; Korhonen, 2009; Lange, Chaudron, & Muskens, 2006; Margaria & Steffen, 2009; McMahon, 2006; Meier, 2009; Nord & Tomayko, 2006; Phillippus, 2009; Rajlich, 2006; Schwaber & Beedle, 2002; Sjoberg, Dyba, & Jorgensen, 2007). We also look at previous studies on students' and academics' perception on Agile practices (Hughes & Bowyer, 2006, 2007; Katira, Williams, & Osborne, 2005; Melnik & Maurer, 2004, 2005; Meneely & Williams, 2009; Muller, Link, Sand, & Mahlpohl, 2004). In the third section, we present an overview of our study and a qualitative analysis of the 35 Agile projects conducted by the various student teams involved in the study over eight years. We conclude with some summary findings.

Agile Methods in Practice – Combination of Techniques within an Agile Method

Agile software development methods have been gaining support with industry practitioners and researchers since late 1990s, with claims of improved quality and productivity outcomes. A number of Agile methods are used in software development with various Agile techniques. Parsons et al. (2007) report in their research into Agile methods in practice that practitioners seem to use a combination of techniques within an Agile method. They explore some of the techniques used within Agile methods to assess their benefits within an Agile process. They claim that the best way to adopt Agile methods is to use it in combination with other Agile methods, and that it is effective to combine XP and Scrum. They show that five of the core techniques of XP method: collaborative working, code refactoring, code regression testing, pair programming and test driven design, should be adopted to achieve the maximum benefits of XP. They conclude that successful adoption of an Agile approach requires not only selecting an Agile method but also appropriate techniques in combination to achieve the best integrated quality process and productivity improvements in system development, and report that practitioners seem to use a combination of techniques within an Agile method.

Although Agile manifesto is prescriptive about the practices they include, Agile method adoption has progressed through the use of a number of approaches, such as XP, Scrum, feature driven development etc., in combination and techniques within these methods in a piecemeal manner. Such adaptation and tailoring of Agile methods to suit the various development and organisational needs have been reported in the literature by XP, Scrum, software engineering (SE) and other Agile researchers and practitioners (Abrahamsson et al., 2003; Ambler 2004; Beck & Andres, 2004; Buglione & Abran, 2007; McMahon, 2006; Parsons et al. , 2007).

However, most development teams (especially including Agile software capstone final year SE projects) may not have the knowledge and skills required to pick and choose between various Agile methods and techniques to be used in combination to suit a project (Parsons et al. 2007). The usual practice is to adopt the Agile method which is most convenient and evaluate / reflect and tailor it for improved outcomes. This tailored approach has been adopted in our SE capstone projects over the last eight years and has meant that we have evolved our Agile method to include specific techniques to address various limitations. Since Agile method practitioners are selecting and adapting from a combination of Agile methods and techniques, it is important to consider which methods and techniques in combination provide the best outcomes. We need to check for and evaluate any gaps in techniques that might impact software process improvements, quality and cost. We have been informed by the lessons learnt in running 1 year long Agile projects since 2002-2009 (Ramakrishnan, 2009), empirical evaluations (Melnik & Maurer, 2003, 2005; Meneely &Williams, 2009; Parsons et al. 2007) and by Agile methods in computer science education (Hughes & Bowyer, 2006, 2007; Katira et al. 2005; Muller et al. 2004). The fourth section presents a qualitative analysis of the 35 Agile projects conducted by the various student teams that have been used to assess Agile methods' claim of software development improvements.

Previous Studies on Students' Perception and Researchers' Evidence on Agile Practices

In Melnik & Maurer (2003), the authors reported on their experience in introducing Agile methods in 4 different academic programs in 2 institutions. They produced a qualitative analysis of students' perceptions on XP in general and 3 core practices of pair programming, project planning with planning game and test-driven development (TDD). They wanted feedback on whether students enjoyed Agile practice, problems encountered, test-driven development and how XP improved their learning. 85% of their students felt that XP teams produced better quality code with pair programming and TDD, and benefited greatly by good communication and collaboration between team members. They also reported that although the concept of TDD was not used consistently by the teams as the other core XP practices listed, they realized the importance of testing and the value of finding bugs early. They emphasised the differences between real-world and student Agile projects: Real XP teams worked at a steady pace and did not have spikes in delivering value to clients, but students are dictated by other competing assignment deadlines in other courses/units.

The ACM/IEEE Computer Science/Software Engineering curriculum lists Agile concepts and practices such as refactoring and TDD as essential topics in SE curricula. Melnik & Maurer, (2005) provide a detailed quantitative analysis of students' perceptions about Agile practices through data collected over 3 academic years from 5 different academic levels (diploma, postdiploma, junior, senior, grad) in 2 institutions with similar questions as in their 2003 work. They reported that the students in general were enthusiastic about core Agile practices. Qualitative analysis again revealed that experience in Agile team work helped them in developing communication & collaboration and adaptability skills. With 240 respondents out of 693 invitations (35% response rate), they reported that 78% believed that XP improved productivity of small teams, and 76% felt that XP improved code quality. Pair programming was liked by a large number of respondents and they felt that code inspection in pairs is more efficient than traditional debugging. They found that this was a statistically significant result in their study and contrary to industry perceptions.

Parson et al. (2007) looked at the impact of Agile methods and techniques on outcomes in Agile projects and provided empirical evidence that showed that Agile methods improve quality, satisfaction and productivity without a significant increase in cost. Their aim was to assist developers

to derive a methodology from the various methods and techniques. They reported that Agile methods were used in combination with a number of different techniques. They provided statistical analysis that showed that choosing a specific combination of techniques in an Agile method resulted in better outcomes. They also reported that the Agile adoption rate was influenced by the extent to which certain core techniques were integrated. The data set used in their paper was from Ambler (2008) survey with 4235 respondents from March 2006. In exploring the relationships between outcomes and Agile methods and techniques, Parsons et al. (2007) treated outcomes as dependent variables and the methods and techniques as independent variables. They found that there was a lack of correlation between an Agile methodology and the techniques actually used with that method. Their respondents reported that they used the following 12 Agile techniques: active stakeholder participation, Agile model driven development, code refactoring, code regression testing, collocation, common coding guidelines, continuous integration, database refactoring, database regression testing, pair programming, single sourcing and test driven design. Their survey found that XP & feature driven development (FDD), and XP & Scrum were the most popular pairs of Agile methods. Other pairs were FDD & Scrum. Agile unified process & FDD etc without XP. The data analysis showed that XP/Scrum combination was better for productivity and quality although the cost or satisfaction was the same in all the various Agile method pairs analysed. The result was plausible as XP focused on technology related best practices of programmers and Scrum dealt with project management and process metrics issues (Buglione & Abran, 2007). Their data analysis of the 12 Agile techniques used in the survey showed that the most effective techniques are: co-location and pair programming, which produced higher benefits of quality, productivity and satisfaction. They also found that from a sample size of 420 using XP, only 8 were using all the techniques. They analysed the data set of XP users to identify the association between the techniques used and the outcomes, and reported on the importance of the outcomes for these techniques in terms of 3 measures: productivity, quality and satisfaction. They found that code refactoring was the most important technique on all 3 measures. Test driven development showed up as the most satisfying. Collaboration showed up as improving productivity and quality concerns. Pair programming improved productivity and code regression testing improved quality. They also looked at the correlation between the number of XP techniques used and the outcomes from using the XP/Agile method and found that the cost factor was independent of the number of Agile techniques used. However, the 3 performance meaures/outcomes showed that the performance improved with the increase in the number of Agile techniques used. This finding is in line with what has been evident in our final year software engineering capstone projects.

Agility and Architecture

Agile software systems must be engineered to address quality and productivity concerns. However, incorrect interpretation of the lean documentation approach of Agile development leads to inadequate levels of architectural design information (Hadar & Silberman, 2008). There is an emerging consensus about the importance of a research theme to address architecture-centric issues in Agile software development (Babar, 2009a, 2009b; Babar & Abrahamsson, 2009; Babar, Pikkarainen, & Ihme, 2008) and for integrating architectural design techniques in Agile methods (Hadar & Silberman, 2008).

Booch (2007a) states “that best projects use a system’s architecture as a primary artefact for governance.” A system’s architecture is very relevant to the various players such as analysts, designers, testers and program managers. An analyst uses the emerging architecture to move from problem space to solution space. The design decision considers various competing objectives and constraints and makes tactical decisions on the evolving architecture. Designers delve into architectural design to explore the functional, non-functional features in the iterative, incremental space

of development. This enables the designers and end users to explore issues and discuss more specific questions on functionality, performance, re-factoring etc. that would not have been possible earlier. Proven architectural patterns should be used in appropriate contexts in future applications to improve their quality and productivity. Analysts use a system's architecture to check the connections between its various components. Similarly, testers can use a system's architecture to conduct system tests. Project managers use a system's architecture to control its incremental releases, manage risks and run tests to check the quality of its implementation and its conformance to user requirement (Booch, 2007b).

Architecture provides a foundation or a blue print from which systems are built. It is an important aspect of Agile software development and is critical for building scalable Agile software systems (Ambler, 2002, 2007, 2009). Scalability should address complexity, team size, distributed teams, compliance requirement etc. In a co-located XP setting for a small project with pair programming, collective ownership and good communication, free-form white board drawings of architecture may suffice. On larger scale multi-site projects, architectural models benefit developers by building a common vision and ownership through architecture models and documentations. Instead of building a big design up front (BDUF), Agile model driven development (Ambler, 2009) evolves in an iterative fashion, where the goal of iteration 0 is to identify the technical directions and risks and elicit an architectural vision for the project. The next iterations result in architecture model incremented with more details. The role of an architecture owner is to work collaboratively with the team members to evolve the architecture. Just like the product owner in Scrum is responsible for the team's requirements, architecture owner is responsible for the team's architecture. In an Agile architecture process, a core team could develop an initial architecture and evolve the architecture by updating the (whiteboard) models as required as the project progresses. These models are discussed with the development teams for resolving any issues. As the requirements come from project stakeholders, their active participation is essential for identifying architectural requirements. Business, technical and operational stakeholders bring their expertise into the decision making. In Agile modelling, there can be no prescriptive set of architecture models that are suitable for a project. It is appropriate to include architecture views which are relevant for the system being built. One should also try to use appropriate architectural patterns. Some of these patterns are: Model-view-controller, layers, and broker. Architecture models show system's dependencies, and UML deployment diagrams, UML activity diagrams, data-flow diagrams etc are useful for identifying dependencies. Although software (product) is the primary goal, architecture documentation is necessary in Agile systems for effective communication for distributed teams and for complex systems. The architecture documentation should contain key architecture requirements, explain critical aspects of the architecture, possibly using a navigation diagram, and an explanation of key decisions/trade-offs in design. XP notion of architecture spikes or architectural prototypes in Rational Unified Process (RUP) is used to show that an architectural iteration works by checking if it works as expected (Ambler, 2009).

Margaria & Steffen (2009) propose extreme model-driven development (XMDD) as a new development paradigm and teaching direction in the computer science curriculum. It is aimed at continual involvement from the customer throughout the systems life cycle. They argue that the traditional development process is no longer applicable in this world of heterogenous, distributed organisational systems which must adapt to changing requirements. They argue for a curriculum which addresses this need with a model-driven, light-weight development paradigm which supports collaboration and cooperation and puts the user/customer process in the center of the development process and the domain/application expert in charge of evolution. This paradigm shift to Agile software development is similar to the views expressed in Rajlich (2006), Dingsoyr, Dyba, & Abrahamsson. (2008) and ITEA AGILE project (Abrahamsson, 2009).

Agile adoption rate is increasing and according to Ambler (2008), over 69% of analysed organisations are using Agile practices on their projects. Software process improvement strategies based on CMMI (capability maturity model integrated) are seen as heavy software development processes with heavy-weight plans and documentation imposed by plan driven method with compliance criteria imposed by CMMI but also as an indicator of organisational maturity. How does one address the needs of CMMI compliant organisation to go Agile or when the clients of the Agile organisation require a level of CMMI compliance. Diaz, Garbajosa, and CalvoManzano (2009) provide an experience report by mapping CMMI (capability maturity model integrated) level 2 to Scrum practices. They show how Scrum techniques of Agile method can be used to identify a set of practices to achieve a certain level of CMMI capability level 2 and help with CMMI compliance issues for small to medium enterprises with a lightweight and flexible CMMI with Scrum/Agile. Ambler (2009) discusses an Agile process maturity model (APMM) with level 1 Agile process such as Scrum and Agile modelling that address a part of the development life cycle and suited for small, co-located teams, level 2 process covering the full Agile system development cycle with a risk and value driven strategy, and level 3 process where scaling factors such as distributed teams, team size and compliance to regulation are considered. Ambler states that APMM and CMMI are complementary but address different strategies. Lean methods that address scalability are likely to be the way forward for Agile software development (Ambler, 2009).

The Agile literature was a bit silent on the impact of architecture design as can be seen from the papers discussed above in the previous section. Exclusive use of Scrum approach can result in focus on functionality and features alone without regard for architecture/design quality and result in performance and product quality issues.

There is a growing interest in bridging the gap between Agile and architecture approaches. However, the role of the architect and architecture-related issues in Agile development still needs to be better understood and integrated with Agile method for ensuring that the Agile development process scales up to distributed large team projects. The 3 main design activities of architectural analysis, architectural synthesis, and architecture evaluation in architecture methods do not proceed sequentially but the architecture grows progressively over time (Hofmeister et al., 2007). This is so because the analysis, finding solution and evaluation for all concerns cannot be done simultaneously, as the inputs of goals, constraints etc are better understood as architecture design progresses. To drive the design process, architects keep a *backlog* of needs, issues, problems to be addressed and ideas to do it (Hofmeister et al., 2007). The backlog drives the workflow, and helps the architect in deciding what to do next. The backlog is frequently prioritized based on milestones to be met, risks to be mitigated, team schedule etc. Once the backlog item is picked by the architects, they do architectural synthesis incrementally, and the current existing design decisions get integrated with this. Thus the backlog constantly changes and sets the objectives for a particular iteration of architectural synthesis. This cycle of adding to a backlog, re-prioritizing, resolving or removing an item can happen in varying periods of hours or days. The backlog is similar to Scrum (Schwaber & Beedle, 2002). The backlog guides the activities through 3 kinds of activities, and assigns objectives for each iteration through a synthesis activity. The architect should also ensure that each iteration of each activity is done after setting objectives for that step.

Agile researchers and practitioners have reported on the need for creating a research agenda for studying architecture-centric challenges in Agile software development (Babar, 2009a, 2009b; Babar & Abrahamsson, 2009; Babar et al., 2008),and have set up http://www.acube-community.org to include research progress in this area. Babar (2009b) has reported that the most commonly occurring architecture-related challenges that Agile teams experience are: 1) incorrect prioritization of user stories without considering technical aspects: if interdependencies among user stories are not discovered early, considerable refactoring may be required impacting the whole software structure. This may be addressed by involving the Agile architect (aka solution

architect) in more management roles and being a Scrum master and also by creating a new role of implementation architect responsible for technical aspects such as getting the user stories implemented. Both architects and developers should be involved in prioritizing user stories in a team session; 2) not considering alternate design choices and evaluating their decision; 3) not focussing on quality attributes during design decisions; and 4) lack of skills, untried technologies and development in a new domain.

Agile architecture method is based on the concept of a common architecture unlike RAD and Agile supports contract first through TDD and designing to interfaces, which helps avoid integration issues. More recently, Agile architecture (spike) seems to be slowly getting adopted within the Agile community. A spike is used for experimentation by developers to learn just enough with unknown elements in a user story such as new technology, and enables them to estimate that user story (Phillipus, 2009). When a user story on the product backlog contains unknown elements that are hard to estimate, the item should be split into a spike to tease out these elements and a user story for implementing the functionality. This split allows the product owner to rank the research higher and ahead of implementation of the functionality. The spike is time-boxed and the research part of a user story is explicitly addressed and time-capping the spike will help in keeping the project on schedule.

The concept of spike is used with an architectural issue. The architectural spike is called Sprint 0 and corresponds to exploration phase in an XP project and preliminary architecture modelling. The aim of the architecture spike is to consider subsystems/components, dependencies and constraints. It is also important to include quality attributes in this process, and Sprint 0 is a good spot to specify the essential product quality requirements (Meier, 2009).

An Agile architecture method presented by Meier (2009) is a structured approach for developing candidate architectures and is an iterative and incremental approach for building quality architecture and design. Meier's method recommends the use of normal requirement gathering input information such as use cases and usage scenarios, functional and non-functional, technology & deployment requirements, and constraints, and produces as output: architecturally significant use cases, architectural hot spots, candidate architectures and architecture spikes. It considers factors that impact on architecture in a step-wise process. The Agile architecture method has 5 steps for designing an architecture: 1) identify architecture objectives, 2) use key scenarios to flesh out the design for critical parts and evaluate the resulting candidate architecture, 3) develop an application overview by focussing on the application type, deployment architecture, architecture & design patterns and technologies to be used to link the architecture/design to the implementation requirement, 4) identify key design decisions based on quality attributes, and 5) create a candidate architecture or architecture spike and evaluate it against key scenarios, quality attributes and deployment constraints. The method offers a sound approach for creating candidate architectures using design patterns etc., identifying relevant spikes and deployment constraints. Meier's method helps to identify key engineering decisions against prioritized user stories during iteration. The key decisions are about exception management, data access etc. and on quality attributes such as performance, reliability, security etc. The Agile architectural process is an iterative, incremental approach and more detail is added to the design in each iteration. This method can be used to explore technical goals, risks and architectural spikes. This method can be used to model an application and draw incremental iterations of the architectural detail using design, implementation and deployment patterns as appropriate in a free-form notation on whiteboards or using drawing tools.

Agile research studies (Korhonen, 2009; Parsons et al., 2007) suggest that Agile techniques, such as pair programming, automated tests, and continuous integration, help reduce the number of faults and improve productivity. However, they also conclude that the faults need to be handled formally. When adopting Agile methods, consideration must be given to distributed development,

tools, and progress reporting. As these are treated separately, Korhonen (2009) suggests that further research is needed with case studies to assess the defect management requirements in Agile development. According to Agile process definition (Schwaber & Beedle, 2002), quality and hence fault-fixing, is the first priority, In practice, implementation of features is seen as more important during the sprint than fixing minor bugs, so that bugs are transferred to the next sprint. Although project manager and quality manager (QM) require faults to be reported to show progress status visible, Scrum team members avoid this task (Korhonen, 2009). It is useful for QM to introduce guidelines on what faults should be reported.

Overview of Our Study and Qualitative Analysis of the 35 Agile Projects Over Eight Years

The objective of this study is to ascertain students' and supervisors' perceptions on Agile methods and if they have changed over the eight years of operation at Monash University. The study focuses on Agile software engineering practices coming from extreme programming (XP) as well as Agile/feature-driven, Agile/Scrum, and Agile architecture.

Analysis of the data collected during the course of the project (over 2 semesters) on Agile methods, are from the 2 SWEBOK (Software engineering body of knowledge) interviews per project team member, 4 incremental software releases per team, and weekly Scrum meetings. Agile methods and techniques were covered during the lecture seminar series.

When quizzed about agile approaches with open-ended questions such as:

1) Did they benefit and enjoy the project unit and why?
2) Did they like Agile or prefer the Spiral or other approaches they were familiar with?
3) Reflections on what worked and what did not, and why?
4) Answers to specific questions on coding standards, XP, pair programming, collective ownership, collaboration/communication, test driven development, Scrum techniques/spikes/sprints, software configuration management, defect management, software configuration management, SVN/Trac, integration testing, Agile architecture, quality & productivity.

The overwhelming response to question #1 from students in 2002-2009 has been very positive in terms of relevance, interest and work (industry) integrated learning, and for question #3 the responses gave the following: things that worked were pair programming, coding standards, incremental releases, collaboration & communication; under things that did not work some teams expressed a view that pair programming worked but took more time as 2 people were involved or because the students' calibre were different. Most started with test-driven development but some reverted back to code then test or test later on when time pressures came into play. What has been evident during 2002-2009 is that even within one year's team with what may be seem as a prescribed subset of Agile method/techniques, teams may embrace only some techniques. They try to explain away the reason as being due to: time table clashes with other team members for not using pair programming consistently, non use of defect management tools due to the size or nature of the language used in the project, Agile being more applicable in industry where members do not have competing assignment requirements and other classes etc.

In 2002-2004, there was also a heavy emphasis on requirement engineering with acceptance testing and architecture and design models upfront, with XP with pair programming, code standards, incremental releases and test driven development. Hence, the response for #2 & #4 was spiral and XP implementation techniques. In 2002, 100% of the team members had positive responses re-

garding pair programming and attributed it to improved quality & productivity. They also embraced the other XP techniques (see Table 1). In 2003, 2 exceptionally bright teams, and another team put in practice all the prescribed XP techniques in Table 1 (60% adoption). In 2004, 4 out of 6 teams (66.6%) were positive about and practiced pair programming and other techniques listed in Table 2.

In 2005, 100% (5/5 teams) practiced pair programming and other Agile techniques listed in Table 2. However, MUSE (Monash University Software Engineering) portal task time tracker service was not uniformly used by the teams. They were not using the online task tracker consistently to record the time taken to do various tasks in a release.

In 2006, 100% of the students practiced the Agile techniques listed in Table 3 except refactoring and CVS/SVN. Only one team re-factored their code. The version control system, Subversion (SVN) was used by 1 team and 2 teams used the older Concurrent version system (CVS).

In 2007, 100% of the teams used Agile techniques listed in Table 4. Since 2007, student teams have access to a server for SVN and they use it enthusiastically to manage their code assets. In 2008-2009, all teams followed Agile/feature-driven approach with Scrum tactics. However, one team member in 2008 re-factored the code and moved away from Agile/Scrum and team members half way through the project to deliver “quality product” as he put it. This sort of hijacking is possible in a student team project when the other team members do not speak up in a group setting. They only owned up during individual SWEBOK interviews.

Table 1: Bachelor of Software Engineering Studio (Capstone) projects: 2002-2003

Clients (Melbourne based)	Year	Teams of	Project title	Technologies used
Jack Verdins, IBM	2002	4	Demo of IBM's Websphere development platform	WebSphere Application Studio Developer 4.0, WebSphere Application Server, WS technology
Columbia Australia Pty Ltd, SME	2002	4	E-Commerce application – stock mgmt	Tomcat (JSP/ Servlet Server), Java IDE (JBuilder 4+), SQL DB
South Port Engineering, SME	2002	4	ClearView, Visual Sales data analysis	Java, MS-SQL
Web Strategy, SME	2002	4	Ecommerce system - Legal Wills – online payments, customer record mgmt	IIS, JMailer, Visual Studio .NET, MS Access 2000, Crystal reports
Rotary	2003	4	Rotary water wheel	PHP, CSS, PostgresSQL, CSS
Cosmic Zone, SME	2003	4	Marketing, online sales site	PHP, MySQL,
Go for IT! Global networks, SME	2003	4	Interactive website for online purchase & workshops for victims of crime	PHP, MySQL, CSS
Faculty of Medicine, Monash Uni	2003	4	Virtual surgical museum (Online interactive learning tool)	PHP, MSSQL, Javascript
South Port Engineering, SME	2003	4	Platform independent browser based sign controller	ASP.NET, C#, Visual C++, MS Access, Ndoc

Source: http://www.csse.monash.edu.au/~sitar/BSE-FinalyearCapstoneProjects-2002-2009.pdf http://www.csse.monash.edu.au/~sitar/CSE4002/CSE4002-and-alumni/ (for more details)

SE method & techniques: to cover SWEBOK, *IEE Std for SRS, QA/testing - 2002-current,* 2002 -2003- XP - pair programming, test driven development, incremental releases, collaboration/communication, , collective code ownership, project coding standards, architecture

Table 2: Bachelor of Software Engineering Studio (Capstone) projects: 2004-2005

Clients (Melbourne based)	Year of	Teams	Project title	Technologies used
Tomato Source, SME	2004	4	Quick Bill – web based billing system	PHP, MySQL
7seals@com, SME	2004	4	Stock ordering system	J2EE, PDA
Digital Bridge, SME	2004	4	J2EE apps for website support & project management	J2EE
Rotary	2004	4	Rotary water (contd from 2003)	PHP, CSS, PostgresSQL
Peter Harding Pty Ltd	2004	4	Java/Eclipse tool for practicing Russian	
Faculty of Arts, Music, Monash	2004	4	Working with sound system for the web	
Everyday interactive networks	2005	4	Confidential	Java, Eclipse, Mercury QTP, JUnit
Confidential	2005	4	Billing system	PHP, MySQL, XML, SSL, test/csv
Confidential	2005	4	Confidential	
OpalTree System Pty Ltd	2005	4	Javaweb for CADRia)	J2EE web services, Sun's Java webstart technology
NCS Pearson Pty Ltd	2005	4	Coaching college student admin, test marking & reporting system	ASP.NET/SQL server, Optical mark reader, OPSCAN scanners

Source: http://www.csse.monash.edu.au/~sitar/BSE-FinalyearCapstoneProjects-2002-2009.pdf (for more details on the unit and projects)

http://www.csse.monash.edu.au/~sitar/CSE4002/CSE4002-and-alumni/ (for more details)

SE method/techniques: to cover SWEBOK, *IEE Std for SRS, QA/testing - 2002-current*, (2004): XP - pair programming, test driven development, collaboration/communication, collective code ownership, project coding standards, automated testing, , MUSE Portal – task time tracker service with release dates for incremental releases, CVS, architecture

Table 3: Bachelor of Software Engineering Studio (Capstone) projects: 2006

Clients (Melbourne based)	Year	Teams of	Project title	Technologies used
Faculty of Medicine, Monash	2006	4	Medical Imaging research DB	Open microscopy environment (OME), OME image server
Faculty of Medicine, Monash	2006	4	Graphical framework for medical visualization & segmentation	Open source tools – itk, vtk (C++ libraries): VTK – visualization toolkit, itk – image registration & segmentation toolkit
7@seals.com	2006	4	Migrate J2EEapps to portal apps	Rational application developer, MySQL Apache Pluto 1.0, Portlet, JEE
7@seals.com	2006	4	Single sign on	Rational application developer, Microsoft active directory on MS Server, Kerberos server on Linux box
Confidential	2006	4	Interactive mood engine (iME)	Agents
Victorian partnership for advanced computing	2006	4	Web services management interface	Web services
Faculty of IT	2006	4	Health Portal	Open source search engine, Postgres metadata DB, java driven interface, webcrawler

Source: http://www.csse.monash.edu.au/~sitar/BSE-FinalyearCapstoneProjects-2002-2009.pdf (for more details on the unit and projects)

http://www.csse.monash.edu.au/~sitar/CSE4002/CSE4002-and-alumni/ (For more details)

SE method & techniques: to cover SWEBOK, *IEE Std for SRS, QA/testing, 2002-current,* **2006- Agile - pair programming/ XP, test driven development, collective code ownership, project coding standards, refactoring, Agile & configuration management, track defects, planning game, incremental releases, automated testing, CVS /SVN, collaboration/communication, architecture**

Table 4: Bachelor of Software Engineering Studio (Capstone) projects: 2007-2009

Clients (Melbourne based)	Year	Teams of	Project title	Technologies used
Bluetounge entertainment, Gaming software company	2007	4	Performance data tracking DB	C#, SQL server backend, csv files
Confidential	2007	4	Project Eden (for managing client's time)	PHP, MySQL
V-SME	2007	4	Web based service desk	Web 2.0, ITIL framework, LAMP – Linux Apache MySQL PhP, Java-script, CSS, Ajax, XML, XHTML, Ja-va, automted testing
Tolls Corporate IT – Mobility product team	2008	4	Knowledge base DB	C#, SQL server
Bluetounge Entertainment Pty Ltd	2008	4	Performance review tool	PHP, MySQL
V-SME	2008	4	Intranet documenta-tion system	Web 2.0 + Ajax, MySQL, Ruby on Rails
Consultant (V-SME)	2009	3	AusVita –online info/service provider for international stu-dents	Ruby on Rails, Ajax, SQLite2, SVN, RSS
ANZ	2009	3	Online finance tools	Flex, MySQL, PHP, CSS

Source: http://www.csse.monash.edu.au/~sitar/BSE-FinalyearCapstoneProjects-2002-2009.pdf

http://www.csse.monash.edu.au/~sitar/CSE4002/CSE4002-and-alumni/

SE method & techniques: to cover SWEBOK, *IEE Std for SRS, QA/testing - 2002-current,* **2007-2009- Agile - pair programming/ XP, test driven development, collective code owner-ship, project coding standards, incremental releases, automated testing , refactoring, Agile & configuration management, collaboration /communication, planning game, architecture, Agile/Scrum/scaling Agile –architecture spikes, UML, SVN/Trac - track defects**

Prior to 2009, student teams produced architecture/detailed design diagrams but they were not linked to the releases. In 2009, students practiced the Agile techniques listed in Table 4 including architecture spikes and sprints etc. However, the architecture sprints were not done for initial re-

leases and only taken up when they realised the impact it would have on their assessment marks. They did realise the benefits of architecture/design sprints when faced with integration issues and elaborated on architecture/design views/drawings for the following software releases. Both teams were initially focussed on the new technology option for web 2.0 and rich internet application that they had chosen – Ruby/Rails and Adobe/Flex and said that the technology supported model view controller (MVC) pattern for their design. As stated by Buschman (2009) in the article about pragmatic architect, the students had to be reminded about the various perspectives and the abstraction levels that an architect needs to concerned with. One of the teams had to produce 3 components for an On-line financial calculator application with Flex. Initially, they split the tasks between the team members without taking into account the dependency relations that existed between the components owing to planned reuse, and decided to allocate one component per team member. An architecture sprint with a diagram showing links between various component items was used by this team to realise this problem and take steps to overcome avoidable schedule/estimate overruns. Each member was allocated a part of each component development. This team could have also better handled the crosscutting issue of security by working on an architecture spike initially to understand the technology better. Both teams in 2009 were initially reluctant to use the integrated software configuration management and project management system, Trac, for defect management and said that with a test driven approach, they fixed any errors detected on the spot and did not see the need for a separate defect management process. The notion of ownership of errors/faults and allocation of resources/time spent for it has always been something student teams have been rather reluctant to do and have wanted to roll the task into development time in progress reports.

Summary and Conclusions

This research has explored previous studies on students' perceptions of Agile methods and has shown how the Agile method adoption is a case of partial adoption of various techniques from an Agile method. The studies have shown that it is better to blend multiple Agile methods which are complementary, and that it is important to use a minimum number of techniques to achieve the best project outcome. Our eight year experiences introducing Agile methods in the final year software engineering projects indicate that students are very enthusiastic about learning Agile practices and comment that it will be very useful in the industry setting as well. Qualitative analysis show that experience of working in Agile teams promotes the development of technical (software architecture/design, construction through pair programming, test driven approach, configuration management), managerial/documenting (planning game, user stories, defect management, progress reports) and professional skills (collaboration, communication, cooperation, collective ownership, adaptation). Agile purists and eminent SE researchers are joining forces in adopting Agile architectures in Agile software development, and it is important to build an empirical body of knowledge to integrate architecture-centric approaches in Agile method. We have reported on some findings from our 2009 projects on Agile architectures in Agile development.

References

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

Abrahamsson, P. (2009). Agile methodology – Achieving a radical improvement in software engineering, Scaling-up to high performance businesses. *M-ITEA 2 Magazine*, April 2009, no. 3.

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. *Proceedings of the 25th International Conference on Software Engineering*, May 2003, pp. 244-254.

Ambler, S. (2002). *Agile modeling: Effective practices for extreme programming and the unified process*. Wiley-Computer publishing, John-Wiley & Sons, Inc.

Ambler, S. (2004). *The object primer* (3rd ed.). Agile Model Driven Development, Cambridge University Press.

Ambler, S. (2007). *Agile software development at scale*. In B. Meyer, J. R. Nawrocki, & B. Walter (Eds.), *In Proceedings of CEE-SET 2007*, LNCS 5082, IFIP International Federation for Information Processing, pp. 1–12.

Ambler, S. (2008). *Ambysoft: Agile adoption survey*. Available at: http://www.ambysoft.com/surveys/agileFebruary2008.html

Ambler, S. (2009). *Agile architecture: Strategies for scaling agile development*. Available at http://www.agilemodeling.com/essays/agileArchitecture.htm

Babar, M. A. (2009a). *Designing and evaluating architectures in Agile way*. Available at http://www.acube-community.org/wikis/images/7/74/TalkAtVTT-WikiVersion.pdf

Babar, M. A. (2009b). *Going Agile? Beware of architecture-related changes and challenges*. Available at http://www.acubecommunity.org/wikis/index.php/Architecturecentric_Methods_and_Agile_Approaches

Babar, M. A., & Abrahamsson, P. (2009). Architecture-centric methods and agile approaches. In P. Abrahamsson, M. Marchesi, & F. Maurer (Eds.), *XP 2009, Lecture notes in business information processing (LNBIP) 31*, Springer-Verlag, Berlin Heidelberg, pp. 232-233.

Babar, M. A., Pikkarainen, M., & Ihme, T. (2008). Identifying and understanding architecture-centric issues in agile software development. *Flexi Newsletter, Jan*, pp. 10-11. Available at http://www.acubecommunity.org/wikis/

Beck, K., & Andres, C. (2004). *Extreme programming explained.* Addison-Wesley,

Booch, G. (2007a). The irrelevance of architecture. *IEEE Software, 24*(3), 10-11.

Booch, G. (2007b). The economics of architecture-first. *IEEE Software, 24*(5), 18-20.

Buglione, L.,& Abran, A. (2007). Improving estimations in Agile projects: Issues and avenues. *Proceedings of Software Measurement European Forum (SMEF)*, pp. 265-274.

Buschmann, F. (2009). Introducing the pragmatic architect.*IEEE Software*, *26*(5) 10-11.

Diaz, J., Garbajosa, J., & CalvoManzano, J. A. (2009). Mapping CMMI Level 2 to Scrum practices: An experience report. In *Proceedings of 16th European Conference, EuroSPI 2009,* CCIS42, Springer-Verlag, Berlin, Heidelberg, pp. 93-104.

Dingsoyr, T., Dyba, T., & Abrahamsson, P. (2008). A preliminary roadmap for empirical research on agile software development. *Proceedings of Agile 2008*, Washington, DC, USA, IEEE Computer Society, Los Alamitos, pp. 83-94.

Dyba, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50* (9-10), 833-859.

Dyba, T., & Dingsoyr, T. (2009a). IEEE Software voice of evidence: Empirical studies of agile software development. *IEEE Software* Web Extra, Sep/Oct 2009.

Dyba, T., & Dingsoyr, T. (2009b). What do we know about agile software development? *IEEE Software*, Sept 2009.

Erdogmus, H. (2009). Architecture meets agility. *IEEE Software, 26*(5), 2-4.

Evans, S. (2006). *Agile architecture approaches.* Available at: http://blogs.conchango.com/simonevans/archive/2006/02/03/Agile-Architecture-Approaches

Hadar, E., & Silberman, G. M. (2008). Agile architecture methodology: Long term strategy interleaved with short term tactics. *Proceedings of Object oriented systems languages and applications (OOPSLA'08),* 19 Oct 2008, Nashville, Tennessee, USA, ACM publication, pp.641-651.

Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., & America, P. (2007). A general model of software architecture design derived from five industrial approaches. *The Journal of Systems and Software, 80,* 106-126.

Hughes, J., & Bowyer, J. (2006). *Incorporating test driven development and continuous integration into the software engineering curriculum*. Retrieved from http://www.imamu.edu.sa/DContent/IT_Topics/Incorporating%20Test%20Driven%20Development.doc

Hughes, J., & Bowyer, J. (2007). Agile methods in computer science education, A case study of an pproach for teaching and learning about agile methods, In *HE Academy of Information & Computer Science.*

Katira, N., Williams, L., & Osborne, J. (2005) Towards increasing the compatibility of student pair programmers. *27th International Conference of Software Engineering* (Missouri, May 2005).

Kitchenhaum, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., & Rosenberg, J. (2002). Preliminary guidelines for empirical rsearch in software engineering. *IEEE Transactions on Software Engineering, 28*(8), pp.721-734.

Knoernschild, K. (2009). *Agile architecture.* Available at: http://techdistrict.kirkk.com/2009/05/06/agile-architecture

Korhonen, K. (2009). Migrating defect management from Waterfall to Agile software development in a large scale multi-site organisation: A case study. In *XP 2009, LNBIP 31*, pp. 73-82, Springer-Verlag Berlin Heidelberg.

Lange, C., Chaudron, M., & Muskens, J. (2006). In practice: UML software architecture and design description. *IEEE Software, 23*(2), 40-46.

Margaria, T., & Steffen, B. (2009). Agile IT: Thinking in user-centric models. *Proceedings of Third International Symposium, ISoLA 2008*, Porto Sani, Greece, CCIS 17, Springer-Verlag, Berlin, Heidelberg, pp. 490-502.

McMahon, P. E. (2006). Lessons learned using agile methods on large defense contracts. Systems & Software Technology Conference, *CrossTalk, The Journal of Defense Software Engineering*, May 2006, pp. 25- 30. Available at www.stsc.hill.af.mil

Melnik, G., & Maurer, F. (2003). Introducing Agile methods in learning environments: Lessons learned. *Proceedings XP/Agile Universe 2003*, LNCS 2753, pp.172-184, Springer-Verlag Berlin Heidelberg.

Melnik, G., & Maurer, F. (2005). A cross-program investigation of students' perceptions of agile methods. *Proceedings of International Conference Software Engineering (ICSE2005)*, St Louis, Missouri, USA, ACM.

Meier, J. D. (2009). *Agile architecture method*. Available at http://shapingsoftware.com/2009/03/02/agile-architecture-method/

Meneely, A., & Williams, L. (2009). On preparing students for distributed software development with a synchronous, collaborative development platform. *Proceedings SIFCSE 2009*, March 2009, Chattanooga, TN, USA.

Muller, M. M., Link, J., Sand, R., & Mahlpohl, G. (2004). Extreme programming in curriculum: Experiences from academia and industry. *International Conference on Extreme Programming and Agile Processes in Software Engineering*, Garmisch-Partenkirchen, Germany, June 2004.

Nord, R. L., & Tomayko, J.E. (2006). Software architecture-centric methods and agile development. *IEEE Software, 23*(2), pp. 47-53.

Parsons, D., Ryu, H., & Lal, R. (2007). The impact of methods and techniques on outcomes from agile software development projects. *IFIP International Federation for Information Processing,* vol 23, Organizational Dynamics of technology-based innovation: Diversifying the research agenda, eds.McMaster, T., Wastell, D., Ferneley, E., and DeGross (Boston: Springer), pp.235-249.

Philippus, E. (2009). *Architecture spikes.* Available at www.agilearchitecting.com/

Rajlich, V. (2006). Changing the paradigm of software engineering, *Communications of ACM, 49*(8), 67-70.

Ramakrishnan, S. (2009). Innovation and scaling up Agile software engineering projects, *Issues in Informing Science and Technology,* Volume *6,* 557-574. Available at http://iisit.org/Vol6/IISITv6p557-574Ramakrishnan573.pdf

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

Sjoberg, D., Dyba, T., & Jorgensen, M. (2007). The future of empirical methods in software engineering research. *Future of Software Engineering (FOSE'07)*, IEEE 2007, pp.358-378.

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.