ELEN 7046– Software Technologies and Techniques

Course Project: Individual Report

June 2015

Name	Student Nr
Adrian Freemantle	1307968

Abstract

Agile development methodologies are a popular approach in small development teams for projects that have variable scope and uncertain requirements. The author attempted to determine if the same approach to dealing with uncertainty in requirements could be applied to dealing with uncertainty in the team dynamic during the implementation of a proof-of-concept survey application. Each of the agile practices adopted are evaluated and their implementation within the project described. The expected and actual value that these practices delivered are evaluated and the importance of good technical communication and object-oriented design skills are shown to play important in role the failure or success of these practices. The author concludes that agile development methodologies are useful in dealing with uncertainty in requirements and project scope, but not with uncertainty regarding the team's skills and capabilities.

Table of Contents

1 Introduction
This paper will discuss the author’s experience in attempting to build and lead an agile team from a group of developers with diverse skill sets and experience levels while completing the goal of developing a proof-of-concept survey application. The initial problems experienced by the team that that lead to the adoption of an agile development methodology and the subsequent challenges faced by the author in fulfilling the Scrum Master role will be discussed.

Each of the agile practices adopted will be evaluated and their implementation within the project described. The value that these practices were expected to add will be compared with the actual outcomes. Practices like pair programming’s success will be compared with the failure of practices such planning poker. Good technical communication skills will be shown to be a solution to many of the problems encountered.

The use of an agile development methodology within the project will be critically evaluated and the importance of team members being co-located, having strong object oriented design skills and the ability to self-organise will be shown to be important aspects to the success of an agile project. It will be argued that a more structured development methodology should be used in the absence of these properties.

The author concludes that agile development methodologies are useful in dealing with uncertainty in requirements and project scope, but not with uncertainty regarding the team’s skills and capabilities.

2 Defining a Way-of-Working
Team-V consisted of five members from a variety of industries including banking, life insurance and document management with four of the members being software developers and one systems analyst. Work experience ranged from five to eighteen years and software-development language skills included C#, Delphi, COBOL and Java. The group’s goal was to develop a proof-of-concept survey application within the constraints provided by the project requirements.

Despite the project requirements being relatively simple and well defined, the group had not made any progress after the first group design session. The variety of skill-sets and preferred programming languages caused conflict as the members could not reach consensus as to which language, architectural style or technology platforms should be used. Concerns were raised that there would not be enough work for each member to make a contribution significant enough for their individual reports, while some members felt that their lack of experience in the technologies being considered would leave them at a disadvantage.

The level of uncertainty within the team resulted in Analysis Paralysis, a state where intelligent individuals with good intentions are unable to make progress as a result of over analysing a problem [1]. A defined way of working was needed that would allow the team to make progress and deliver a working solution within the allocated time.

2.1 Selection of a Development Methodology

Agile development methodologies are well suited to dealing with uncertainty [2] and for this reason the author proposed that the group use such a process. Rather than adopting a specific agile methodology the group selected individual practices such as user stories for requirement specification, planning poker for estimates, and pair programming for transferring skills between team members. These practices formed the basis of the group's way-of-working, a process that would promote self-organisation of team members while allowing the solution to evolve in an organic manner [3].

Self-organization does not mean that the team lacks structure or direction, instead members are disciplined and work together toward reaching a common goal without the need for a centralised command-and-control form of leadership. This concept was difficult to grasp by team members who came from highly-structured development environments as they were uncertain as to what was expected of them. For this reason it was decided provide some structure to the team by adopting the roles used in the Scrum methodology; the Product Owner, Scrum Master and Development Team [4].

2.2 Team Structure

The Product Owner represents the client and is the primary source of requirements. He understands the problem domain, why the system is being developed and who the users of the system will be. As he represents the client he also prioritises features according to business value and drives the acceptance-testing process [4]. Kyle Rangan assumed the Product Owner role.

The Development Team is responsible for providing estimates of effort for each requirement and the incremental implementation of system's functionality. The Development Team is self-organising around individual skill sets and thus claim and divide the work among themselves rather than having task allocated to them [4]. The Development Team was comprised of Chris Viljoen, Derrick Lew and Darryl Smith.

The Scrum Master is as a facilitator, mediator and mentor that guides the Development Team on the adoption of agile practices. It is a leadership role that uses influence rather than authority and serves the team rather than managing it [4]. The author assumed the role of Scrum Master as he had prior experience in building and leading agile teams. As part of the role the author accepted the responsibility of organising group meetings, keeping minutes of all discussions, ensuring members were kept up to date on all developments and ensuring team members kept commitments that had been made.

3 Requirements Analysis

The developers in the group wanted to start coding as soon as possible as the project requirements were perceived to be relatively simple and well defined. It was however clear that the group did not share a common understanding of the requirements and implementation details. A product vision was needed to create a shared understanding of the purpose of the system, who the users would be and what value they would derive from it [5].

3.1 Product Vision

Creating the product vision was a collaborative process, facilitated and guided by the Scrum Master, which involved all group members. The goal was to arrive at a technology agnostic description of the intended purpose of the system by having the group members openly discuss and debate their interpretation of the requirements.

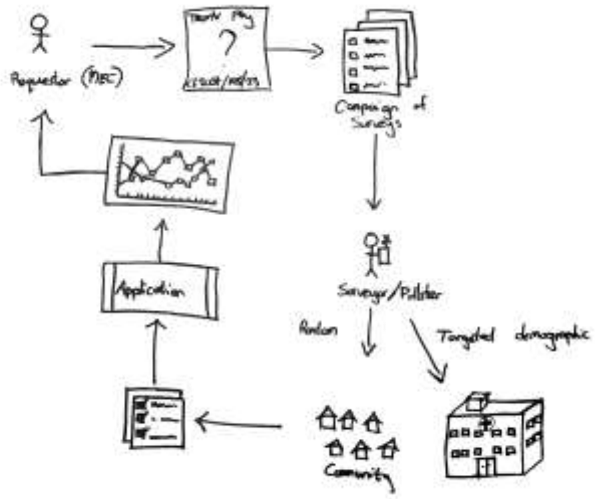


Figure 3-1 Whiteboard sketch of the product vision

The product vision was captured in simple but effective a format described in Geoffrey Moore's book *Crossing the Chasm* [6]:

> ***For*** *local government officials* ***who*** *want to monitor service delivery in communities t****he*** *Mobile-Poll system is a portable and offline-capable survey application t****hat*** *allows pollsters to conduct a campaign of surveys and capture data electronically*. ***Unlike*** *paper based surveys that require manual data entry and* ***unlike*** *online surveys that require an internet connection* ***our product*** *combines offline survey capabilities with automatic online data-synchronisation.*

By discussing the requirements in a technology agnostic way the team had reached a deeper and shared understanding of the problem domain. The process proved to be effective in making assumptions explicit and creating a shared understanding of what the solution needed to achieve.

This process also served as a brainstorming session where innovative ideas were shared and explored. An example of one such idea was the suggestion that a standard paper based questionnaire, a smart phone camera and Optical Character Recognition (OCR) software could be a used to capture surveys. Many such ideas were discussed and were considered in terms of their viability and feasibility.

It is the author's opinion that this process was extremely valuable in the formation of a team-oriented mind-set instead of an individual-oriented mind-set. Group members felt their opinions were valued and that ideas were accepted or rejected based on merit.

3.2 Requirements

User stories are a lightweight method used to capture essential requirement details in the language of the problem domain. They also serve as a planning tool for the purpose of estimation and prioritisation. The format is not intended to provide detailed specifications but serves as a reminder of functionality to be developed [3].

Once the group had a shared product vision the next he goal was to have the Scrum Master and Product Owner define the system requirements in the user-story format while the Development Team would be responsible for assigning an estimate of effort to each user story. Once all stories had been captured and estimated the Product Owner would allocate a priority to each of them. The template used to define a user story can be seen below [4]:

> **As a** *user role*
> **I want** *some functionality*
> **So that** *some derived business value*

The first user story was written in a way that encompassed the essence of the system's functionality in one sentence:

> ***As a*** *local government official* ***I want*** *to monitor service delivery in communities* ***so that*** *I can benchmark the current state and monitor service delivery improvement over time.*

This provided the basis from which all other stories could build, each going into finer detail until each story would take a no longer than a week or two to implement. The process of recursively decomposing the system into its sub-features is also known as creating a Work Breakdown Structure [7, p. 109].

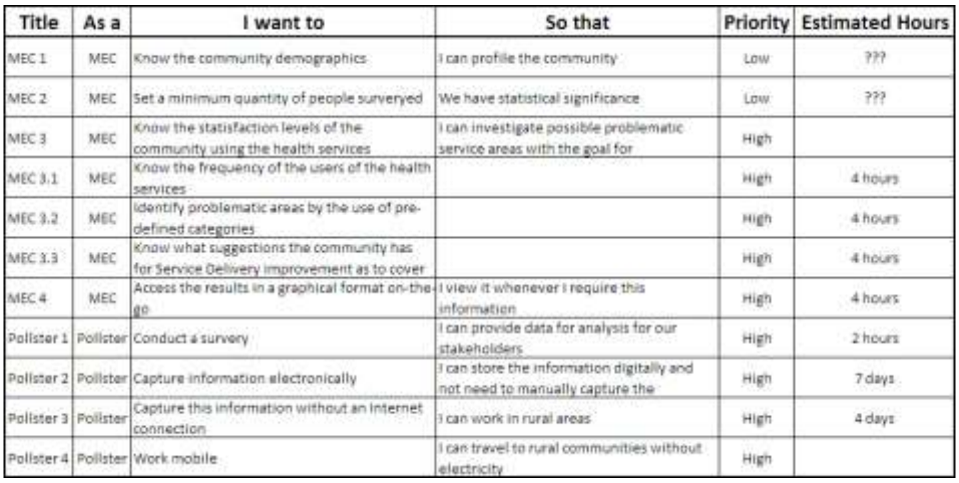
The team was surprised that the stories were being written on index cards and not being captured directly into an electronic document or spreadsheet. The advantage of using a paper based system was that a card could be passed around the table for evaluation as soon as the user story had been completed. This allowed the Development Team to provide an estimate of effort while the next user story was being written.

3.3 Estimation

The Development Team used the Planning Poker technique to estimate each user story as soon as it had been completed. This estimation technique is done through the use of a card based estimation game with the goal being to provide each developer the opportunity to provide an estimate without being influenced by more experienced team members [4]. During each estimation round team members select a card with a value representing their estimate and present it. If no consensus is reached the highest and lowest estimators are asked to justify their estimates after which the process is repeated until a reasonable consensus has been reached [8].

This estimation technique proved to be of little value for obtaining accurate estimates within this particular project group. Planning Poker is an expert-opinion based estimation technique which relies on the estimator's expertise in the technology platforms and problem domain [7, p. 109]. Some of team members had to make a large number of assumptions as they had no prior experience with the technology, development methodology or problem domain. This led to deviations of an order of magnitude between the low and high values with estimators frequently being unable to justify their estimate. The process proved frustrating and was eventually abandoned.

The image below provides an overview of the outcome of the analysis process.



Title	As a	I want to	So that	Priority	Estimated Hours
MEC 1	MEC	Know the community demographics	I can profile the community	Low	???
MEC 2	MEC	Set a minimum quantity of people surveyed	We have statistical significance	Low	???
MEC 3	MEC	Know the statisfaction levels of the community using the health services	I can investigate possible problematic service areas with the goal for	High	
MEC 3.1	MEC	Know the frequency of the users of the health services		High	4 hours
MEC 3.2	MEC	Identify problematic areas by the use of pre-defined categories		High	4 hours
MEC 3.3	MEC	Know what suggestions the community has for Service Delivery improvement as to cover		High	4 hours
MEC 4	MEC	Access the results in a graphical format on-the-go	I view it whenever I require this information	High	4 hours
Pollster 1	Pollster	Conduct a survery	I can provide data for analysis for our stakeholders	High	2 hours
Pollster 2	Pollster	Capture information electronically	I can store the information digitally and not need to manually capture the	High	7 days
Pollster 3	Pollster	Capture this information without an internet connection	I can work in rural areas	High	4 days
Pollster 4	Pollster	Work mobile	I can travel to rural communities without electricity	High	

Figure 3-2 Summary of user stories, estimates and priorities

4 Design, Implementation and Testing

Agile design is the *continuous application of principles, patterns, and practices to improve the structure and readability of the software* [3]. This can be seen in the practice of test-driven development where the design, development and test phases of the software development lifecycle all merge into a single process. When combined with pair programming, where two developers work on the same code, the synergy between these practices can result in a system that is well designed, simple, clean and maintainable.

Agile developers are dedicated to keeping the design as appropriate and clean as possible. This is not a haphazard or tentative commitment. Agile developers do not "clean up" the design every few weeks. Rather, they keep the software as clean, simple, and expressive as they possibly can every day, every hour, and every minute. They never say, "We'll go back and fix that later." They never let the rot begin. [3, p. 159]

The Development Team's high level design resulted in two main sub-systems; a client application that would be developed using web based technologies such as HTML5 and AngularJS, and a server application that would be developed in C#. The following sections will discuss the author's experience in mentoring the server-application developers in practices like pair programming and test driven development.

4.1 Test-Driven Development

Test driven development is the practice of developing features by first creating a failing unit test and then developing the simplest possible functionality that will make the unit test pass. This process is repeated until the entire feature is considered complete. This process can be described as [9]:

Think: Determine the single piece of functionality that the test will evaluate.
Red: Create a failing unit test.
Green: Implement the simplest possible functionality to make the test pass.
Refactor: Improve the internal design of the class
Green: Ensure all unit test continue to pass.
Repeat: Repeat the process until all desired functionality is implemented.

This practice improves the design of the system due to the developer having to continuously evaluate the design of the class being developed. Refactoring is made easier as any breaking changes will be detected by the unit tests and coupling between classes is reduced in order to test each class independently [3].

The three developers working on the server application had to be able to make progress without affecting or being dependant on the progress of another developer. The loose coupling that test-driven development requires allowed various parts of the system to evolve independently.

The act of writing a failing test before implementing functionally required a significant change in the way the developers thought about the design of their classes. The author found that developers with some understanding of the principles of object oriented design found it easier to adapt to this practice. The use of pair programming did however help to convey some of these principles as they could be explained in context while implementing the unit tests.

4.2 Pair Programming

Pair programming is a practice where a pair of developers work together on the same code at the same time. This can be described as “two developers, one keyboard”. While one developer is focused on writing the code, the other monitors the code being written for defects or design improvements. This practice is especially useful for the transfer of domain knowledge and technical skills between team members [3].

This practice proved to be exceptionally effective at transferring skills between individuals and was well received by all team members that that used it. On more than one occasion an extra developer would join a pair-programming session just for educational purposes. It was encouraging to hear how these skills were being adopted by some of the team members in their work environments.

The practice was also an effective means of reducing defects regardless of the experience level of the observer or coder or their preferred programming language. On more than one occasion the author had a COBOL developer point out logic or syntactic errors in his code.

A problem encountered with this practice was that the observer could easily become too actively involved and “steer” the coder. This problem appeared to be most common when the coder had significantly less experience than the observer. The author found that it was important that both parties had a clear understanding of what was being developed and that small but regular design discussions could avoid this problem.

5 Critical Evaluation

The author's previous experience in building and leading agile teams had been with team members that he had some seniority over, who were co-located, had a reasonably good understanding of the principles of object oriented design. The lack of any authority, the distributed nature of the team members and a poor understanding of the principles of object oriented design presented challenges that the author had not previously encountered.

Organising regular meetings where the all team members could attend was difficult and at times impossible. Using the available time after class for a "weekly stand-up" meetings proved to be an important factor in keeping team members up to date with what needed to be done and to ensure that commitments were being adhered to, but despite this constant problems were encountered with regards to team members not meeting commitments.

The inability of some team members to meet commitments caused conflict within the group, particularly when members who were paying for the subject themselves felt that a particular group member was placing the group project in jeopardy. The author had to mediate in these situations and had to apply selective pressure in a diplomatic manner in order to motivate problematic team members while maintaining healthy working relationships within the group. For the author, these situations highlighted the true difference between leading through influence and not through authority.

The value of a product vision to the group was higher than the author had anticipated and it highlighted the danger of assumptions and the importance of good communication. This same theme was encountered during pair programming sessions where the author frequently assumed that his coding partner understood what was required. In this situation the author learnt to appreciate the value of simple UML diagrams as an effective technical communication medium.

The use of planning poker as an estimation technique was entirely unsuccessful within the context of this project. The author failed to appreciate extent to which expert-opinion based estimation techniques rely on the estimator's expertise in the technology platforms and problem domain. Later discussions with team members indicated that estimating user stories as opposed to technical specifications proved unintuitive and there was uncertainty as to whether the estimate should only consider the specific functionality being described or whether the development of infrastructural code, which is commonly reused between features, should also be included. The author has investigated potential solutions to the problem but found many contradictory opinions and no definitive answer to the problem.

The author realised that strong object-oriented design skills were required for the successful use of test-driven development. A lack of such skills resulted in the practice adding little value as the design was not improved and productivity was significantly reduced. Pairing an unexperienced coder with an experienced observer mitigated this problem and proved to be an effective means of teaching the principles of object-oriented design as the coder could immediately apply the principles within the context of the tests being created.

Pair programming exceeded the author's expectations in its ability to transfer skills and improve code quality. All experience-level combinations of observer and coder were found to have be beneficial but also to exhibit various problems. The author found that the root cause of these problems lay in assumptions and communication gaps. Small but frequent design discussions between coder and observer using UML as a technical communication tool proved to be an effective way of dealing with these issues.

While the project was completed successfully, the author would be hesitant to use an agile approach on such a project again, especially as the uncertainty in this project was not the requirements but the capabilities of the team. Co-location, strong object oriented design skills and the ability to self-organise are important properties required in an agile development team. The lack of these properties in the group combine the well-defined requirements meant that a controlled quality approach would have been more appropriate in the execution of this project.

6 Conclusion

In section two, the Scrum Master was described as being a facilitator, mediator and mentor that guides the team through influence rather than authority. The role proved to be more demanding than the author had anticipated and each of these qualities was put to the test during the execution of the group project.

Building and leading an agile team from a group of developers with diverse skill sets and experience levels proved to be problematic. Developers from highly-structured development environments battled with the concept of self-organisation. Poor object oriented design skills among some members resulted in the author spending a large amount of time coaching them in these principals.

Practices such as a product vision, user stories and pair programming were found to be easy to adopt and added value to the team as a whole, while practices such as planning poker failed entirely. Test-driven development met with mixed success depending on the coder and observer pair with the primary problems being caused by a lack of sufficient communication between the pair.

Agile development methodologies were shown to be most useful in dealing with uncertainty in requirements and project scope, but not with uncertainty regarding the team's skills and capabilities.

7 References

[1] Cunningham & Cunningham, Inc., “Analysis Paralysis,” [Online]. Available: http://c2.com/cgi/wiki?AnalysisParalysis. [Accessed 28 June 2015].

[2] Agile Project Management, “Management of Uncertainty in Agile Projects,” 22 February 2015. [Online]. Available: http://managedagile.com/2015/02/22/management-of-uncertainty-in-agile-projects/. [Accessed 29 June 2015].

[3] R. C. Martin, Agile Principles, Patterns, and Practices in C#, Prentice Hall, 2006.

[4] P. Hundermark, “Do Better Scrum,” [Online]. Available: http://www.agile42.com/en/agile-info-center/do-better-scrum/. [Accessed 20 06 2015].

[5] ScrumSense, “Product Owner Manual,” September 2010. [Online]. Available: http://www.scrumsense.com/resources/product-owner-manual/. [Accessed 25 06 2015].

[6] G. Moore and R. McKenna, Crossing the Chasm, New York: Harper Collins, 2002.

[7] B. Hughes and C. Mike, “Software Project Management,” McGraw-Hill, 2009.

[8] J. Grenning, “Planning Poker,” April 2002. [Online]. Available: http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf. [Accessed 15 June 2015].

[9] J. Shore, “Red-Green-Refactor,” The Art of Agile, 30 November 2005. [Online]. Available: http://www.jamesshore.com/Blog/Red-Green-Refactor.html. [Accessed 28 06 2015].

Appendix A – Project Timesheet

Date	Description	Start time	End time	Duration
2015-04-01	Weekly team meeting	19:00:00	20:00:00	01:00:00
2015-04-08	Weekly team meeting	19:00:00	21:00:00	02:00:00
2015-04-15	Weekly team standup	19:00:00	21:00:00	02:00:00
2015-04-23	Weekly team standup	19:00:00	20:30:00	01:30:00
2015-04-29	Weekly team standup	19:00:00	21:00:00	02:00:00
2015-05-02	Team design and requirements session	11:00:00	15:00:00	04:00:00
2015-05-03	Requirements and design session	11:00:00	15:00:00	04:00:00
2015-05-06	Weekly team standup	19:00:00	20:30:00	01:30:00
2015-05-10	Scaffolding Source Solution	10:00:00	16:00:00	06:00:00
2015-05-11	Scaffolding Source Solution	16:37:32	19:52:46	03:15:14
2015-05-12	Scaffolding Source Solution	18:15:00	22:35:13	04:20:13
2015-05-13	Weekly team meeting	19:00:00	21:00:00	02:00:00
2015-05-20	Weekly team standup	19:00:00	20:30:00	01:30:00
2015-05-24	Pair programming session for parser engine	11:00:00	14:30:00	03:30:00
2015-05-26	Meeting with Kyle to discuss Reports	18:00:00	20:30:00	02:30:00
2015-05-27	Parser engine with Derrick	16:24:23	18:40:13	02:15:50
2015-05-28	Fixing infrasturcture bug with InMemory Database	17:56:00	20:15:00	02:19:00
2015-05-30	Testing Mongo DB	09:00:00	14:20:00	05:20:00
2015-05-31	Server code with Kyle, Chris and Derrick	11:00:00	17:30:00	06:30:00
2015-06-10	Weekly team standup	19:00:00	21:00:00	02:00:00
2015-06-13	Team group coding session for integration	08:45:00	15:30:00	06:45:00
2015-06-20	Presentation preparation with Darryl	11:00:00	15:00:00	04:00:00
2015-06-21	Group presentation preparation	11:00:00	15:00:00	04:00:00
				74:15:17