School of Computer Science
The University of Manchester

KANBAN FOR SMALL SOFTWARE PROJECTS

Project Background Report

May 9, 2011

By
Lucas D. Rola

Contents

List of Figures

List of Tables

Abstract

Kanban is a tool used by Toyota and Honda to increase productivity and eliminate waste from their large-scale manufacturing processes. Recently, the system has been adapted to the *lean* software development [Kni10, Bro10a]. New variations of the system are becoming increasingly popular in the industry, but there does not seem to be a comprehensive critical evaluation of Kanban applied to small-scale software projects.

The paper describes the Kanban system, its origins and recent adaptations to the area of software engineering. It discusses the philosophy behind the system and rationale for using its individual components. The experiment's originality comes mainly in its scope. The study attempts to analyse Kanban applied to small and versatile development teams building small software projects.

Furthermore, the author proposes a research methodology consisting of qualitative methods including systematic self-observation, direct observation and a comprehensive analysis of existing archival case studies. The methodology will be used to assess the suitability of Kanban to small software projects which will range from very small of only one person to the group of four developers.

This study will involve the critical analysis of the value of Kanban for small software projects. It also promises to develop a set of guidelines for small development teams choosing to use Kanban as their primary tool for project management. Moreover, the study will also compare and comment on a range of productivity tools available for teams using Kanban.

Chapter 1

Introduction

1.1 Project Background

The information-based societies of today are in constant need of new and innovative software solutions [Iko10]. The Worldwide IT services revenue totalled $763 billion in 2009 [MT10]. As a result of this the contemporary customer-oriented software industry has become demanding and fiercely competitive. This made it an attractive market to get into for many startup software firms. These software firms are trying to find methods of managing their projects in a low-cost, reliable and efficient way.

Over the years, there have been many attempts to tackle these issues. Some of them will be described in this paper. Starting with the formal methods of the Waterfall process followed by its iterative adaptations and finishing with the *agile* school of software development. The promise of meeting today's demanding project management requirements comes from Japan, where the creator of Toyota Production System, Taiichi Ono conceived a management system known around the world as Kanban [Ohn88]. For the past few years many multinational companies got intrigued by its apparent simplicity, transparency and ease of use. Furthermore, it is showing some promising results, particularly when used by small and versatile development teams, where short feedback cycles and small iterations are common [Kni10]. Kniberg [Kni10] claims that it can provide a basic set of tools for rigorous and disciplined software development for small teams.

1.2 Aims and Objectives

One of the main advantages of Kanban is its flexible and non-prescriptive nature. It provides the team with a small set of guidelines for managing any project. After the primary rules have been applied the project manager needs to make decisions based on his teams' size and performance [Ohn88, Kni10]. The manager can make certain changes and modify certain attributes of the system. This apparent advantage of Kanban can become an issue for inexperienced users who have never used this method before.

Therefore, this paper aims to make recommendations about the best practices which could be applied as an aid to improve their current projects. In order to do that, exhaustive testing and evaluation of the method in question will be conducted. This project will address issues such as; the optimum size of the team, their experience and roles, projects size, use of time and resources. The main aim and originality of this project comes from the investigation of small projects whose scope is outlined in the next section. The project will also seek to analyse their causes and suggest some improvements to the problem. At the end of the project this paper should serve as a manual for everybody considering the use of Kanban for his or her teams software project.

1.3 Scope

In order to accomblish the objectives mentioned before, the scope of the project needs to be clearly defined. For instance it needs to be established what the term "small software project" means.

Despite the existence of many complex software metrics systems the author decided to simplify the one developed by Royce [Roy98]. Royce proposed seven metrics which are under two main categories Management and Quality. Since the quality of software projects is hard to estimate for projects of such a small scale, the author decided to focus the Management category which includes: project's budget, lines of code and people involved.

Therefore, in this paper a project is considered to be small if it meets the following criteria:

- **Lines of code:** Less than 5000
- **Budget, working hours:** Less than 1000 hours
- **People involved:** 4 or less

Projects of these nature tend to be relatively short compared to the rest of the industry. Nevertheless they are common especially when it comes to academia or small software firms. These type of projects tend to be iterative in nature, do not require complex domain knowledge, nor extensive documentation and the quality requirements are not very high either [RM00].

1.4 Report Overview

The rest of the report is organised in the following manner.

Chapter 2 will introduce some basic concepts related to this project. First it describes software development with similarities and differences to some well-established models practised in the industry. Then, the next chapter will explore the philosophy behind the Toyota Production System. After that, the section will analyse the system application to the domain of software engineering.

Chapter 3 describes the way that the project will be conducted: the research methodology, evaluation criteria and the plan which will be executed. It will outline the individual phases together with the reasons for conducting them. It will also describe the research methods which will be used to gather and analyse data.

Chapter 2

Project Background

This chapter will discuss the background to Kanban used for small software projects. We will start by looking into what management and leadership are, and why they matter in project development. This will be followed by an overview, analysis and comparison of different project management methodologies. We will then analyse the history, characteristics and status of contemporary software project management. Finally, the paper will outline and comment on some of the challenges that the software industry faces today.

2.1 Project Management in the Software Industry

2.1.1 Characteristics of Software Project Management

> "Management is the activity of directing people to accomplish certain desired objectives using available resources effectively. It involves planning, scheduling and assigning work to people." [SS05]

> "Software project management is the art and science of planning, monitoring, leading and controlling software projects." [SG05]

Software project management encompasses many subfields such as: risk management which aims at measuring or assessing risk and developing strategies to reduce or completely avoid it; requirements management which is concerned with identifying, eliciting and analysing clients' requirements; release management which is the process which identifies, documents and schedules releases of new versions of programs. Furthermore in order to successfully complete desired tasks, organisations need to exercise a certain level of leadership which is defined as:

> "The action of leading a group of people or an organization, or the ability to do this." [SS05]

However, it is not yet clear how much leadership is required to help development teams. Ikonen and Poppendiecks [Iko10, PP03], for example, both claim that some level of leadership is beneficial for project management as it helps to eliminate waste, and provides a team with a sense of direction. On the other hand Kotter [Kot96] argues that in terms of software development a clear authority can be both chaotic and liberating as it allows greater flexibility for the team, but at the same time can result in disorganisation and lack of clearly defined objectives.

When it comes to software engineering it was thought that management practices from the manufacturing world might be successfully transferred, and that such a transfer could be accomplished by someone with business training but no or little programming skills. Reynolds [Rey08, p. 1] however, argues that software engineering is entirely focused on design work, which makes it difficult for somebody without technical background to manage it effectively. According to him, manufacturing is the repeated construction of identical objects, while software is the construction of unique ones. He then goes on to compare a manager who is not familiar with programming languages to the managing editor of a newspaper who cannot write. This feature of software project management must be taken into consideration if we wish to effectively plan a project. The other thing that seems to separate software engineering from other types of software management is its apparent dynamic and unpredictable nature which manifests itself with rapidly changing requirements.

2.1.2 Historical Perspective

Long before everyone had their own personal computer, computer software was developed for dedicated dedicated machines and purposes. Back then, most of the programmers were skilled mathematicians who produced code which was not meant to be reused. In the 1960s Object-Oriented programming became popular and companies realised that repeatable solutions were possible with component-based software engineering. The software industry became aware of the potential that software programming had over hardware circuitry. The software industry grew very rapidly in the 1970s and 1980s. In order to complete most of the tasks companies decided to apply proven management methods from the manufacturing industry. However, most projects were not finished in time or within their budget. Poppendieck [PP03] claims that the reason behind that was the misunderstanding of the true nature of the discipline and misapplication of their methods to the field of software development.

Companies soon realised that matching user requirements to delivered products was essential to the success of their projects. As a result the Waterfall model was developed. This methodological approach to software engineering focused on producing extensive documentation of "Up Front Requirements" before the design, implementation, testing and maintenance phases. Once each of these phases had been completed, it was impossible to go back and modify it. The methodology is now widely criticised, especially by practitioners of the *agile* approach for its lack of flexibility and ineffectiveness in coping with changing requirements and system design. McConnell, on the other hand, claims that the methodology can prove useful for some projects, and although it is not as effective in the consumer-driven market of today, it might still be successfully used for certain types of software projects where application security is essential or slow continuous change is expected. He also shows that the bugs found in early stages of project development are up to 200 times cheaper to fix than without using methods based on the Waterfall model [McC04]. This might be an important argument for using the waterfall process in projects that require security and predictability.

2.1.3 Contemporary Software Industry

Today software project methods are still evolving. However, current trends are tending to lead towards iterative development with short release life cycles and continuous contact with clients [DC03]. Examples of such approaches are *lean* methods and variations of *agile* software management methodologies [SB01] in particular. These approaches have captured the attention of academics and practitioners in the past few years. There are studies which demonstrate the effectiveness of iterative practices. Together with it a new vocabulary has emerged that defines software in terms of “components”, and their behaviours as “use cases” [DC03].

Over the past few years this trend has been sustained, with some new developments in the area. One of them is the attempt to adapt Kanban, the production system used by Toyota Corporation into the field of software engineering. The system can become a good alternative for managers who need an easy-to-follow set of guidelines for their project management without having to resort to weeks of training. With the growing need for IT services around the World, there is a growing trend to create small businesses. These small firms are invaluable contributors to the economy, providing 60% to 80% of new jobs each year, and innovation through competitiveness and a fresh outlook on the market which is beneficial for the economy [DB01]. It has been debated what causes small businesses to fail, because reasons can be numerous. However, in most cases one of the key factors is bad management [DB01]. It will undoubtedly be helpful for these small companies to acquire new, innovative, and easy-to-learn project management methods.

2.1.4 Problems and Challenges

Despite having a myriad of project management methods to choose from, the Chaos Report [cha94] argues that the vast majority of software firms are having their projects abandoned, overdue, or not meeting clients’ requirements. These findings have been questioned by [EV10] who claims that these figures are “unrealistic and biased”. This might be because of the difficulty of defining what constitutes a successful project, which will be elaborated on in the following section. Charette [Cha05], on the other hand claims that the problems which the contemporary software industry is facing are due to its failure to confront reality, and there is a pressing need for a change in development practices. This can be supported by the fact that in a recent study nearly 2000 government and private sector organisations voluntarily reported the maturity of their software management practices to the U.S. Software Engineering Institute, in Pittsburgh. The research concluded that for over half of these organisations were inadequate in terms of quality management practices [Cha05].

The consequences of bad software management are not isolated from the public domain. In 1986, the London Stock Exchange started to automate its system for settling stock transactions. Seven year after starting it and spending $600 million, it decided to stop it because the management of the project was, to quote one of the managers, “delusional” [Cha05]. Investigators concluded that no one seemed to have been interested in what the status of the project was. Therefore, the issue of software management seems imperative for modern IT-dependant societies of today and its importance should not be underestimated. These are only some of the numerous examples of ineffective management and unreasonableness in software engineering. Charette [Cha05] concludes that such behaviour happens frequently and is illogical in cases where the likelihood of success is non-existent [Cha05].

Furthermore, since the “Dot-com bubble” companies have been looking for Return on Investment (ROI) of less than a year, while in the period preceding it the three- to five-year ROI was the norm. This indicates that developing software is more unpredictible and hectic than ever before. Denne [DC03] states that the answer lies in a new software development methodologies which are able to divide projects into smaller market identifiable constituents called Minimum Marketable Features (MMF). As it happens, MMF is the main domain of the Kanban system which will be described in later sections.

2.2 Philosophy Behind the Toyota Production System

2.2.1 The Birth of the Toyota Production System

The history of lean manufacturing goes back to the late 1940s. Right after the war the Japanese automobile industry was in deep crisis because of Japan's low economic growth. It needed to compete with modern mass-production oriented and technologically advanced American manufacturers whose cars were selling much better due to the state and size of their economy [Ohn88]. Back then the productivity rate of American to Japanese work forces was 1-to-9 in terms of production throughput. This prompted the founder of Toyota Motor Corporation, Toyoda Kiichiro to utter these words:

> "We have to catch up with America in three years. Otherwise, the automobile industry of Japan will not survive." Toyoda Kiichiro

According to the father of the Toyota Production System (TPS), Taiichi Ohno, the conventional model of mass production used by Americans stops being profitable when confronted with a recession or periods of slow economic growth. This is when the demand for different types of cars is high and predictions about production levels are hard to estimate [Ohn88]. After the Oil Crisis of 1973 managers at Toyota realised that imitation of the American production system was not suitable to their needs. The idea was to make a system that would minimise or completely eliminate a factory's inventory and eliminate waste, thus reducing overall production cost. The objective was to create a system which was responsive, and worked without delays, or unnecessary processes.

2.2.2 Elimination of Waste

Ohno claims [Ohn88] that the system's main goal and the fundamental principle is the total elimination of any non-value-adding activity (NVA) from the production process. This new concept of waste was considered then to be a revolutionary idea. It was believed however, that the complete elimination of waste would result in a well synchronised system that would quickly respond to change, making the Japanese production system cheaper and more efficient than the American one.

Ohno lists seven kinds of waste that are present in any production system. For instance, there is the waste of inventory related to overproduction and extra processes. Before it was thought that inventory should be kept reasonably large, so that orders could be completed fast. In fact, excess inventory is considered to be the worst types of waste, because it can hide all other type of waste. This may happen even when the system runs uninterrupted and uses its full capacity. It can produce unneeded items which wastes the companys resources [Ohn88]. The next one is the waste of waiting, connected to the waste of motion and transportation, which is a symptom of the system's poor synchronisation. The results are delays in production and decreased throughput of the entire system.

One of the basic rules of Kanban is to never send defective products to the subsequent processes. Without applying this principle, the stock of defective components would pile up, making it impossible to assemble properly working final products. This could cause the process to accumulate waste, not only by overproducing items, but also by using manpower, funds and time taken to create defective products. This means that the early recognition of defective products and its solution should be given the highest priority. The biggest challenge is to recognise the biggest sources of waste and their incremental removal, until there is no waste left and all NVAs become absent.

2.2.3 Just-In-Time

Production systems around the world can be categorised into *push* and *pull* oriented. The manufacturing process conceived by Henry Ford is an example of a standard *push* method in which quantity to be produced is determined by demand predictions, and inventory at hand determines the succeeding production periods. When it comes to the *pull* process used by the TPS, the process is handled in reverse order. The later process withdraws quantity needed from the earlier one, resulting in low levels of inventory and greater adaptability of the entire system to external changes [Ohn88]. The TPS uses the *pull* method which is supported by the just-in-time (JIT) approach. JIT makes it possible to decide as late as possible, thus eliminating the danger of overproduction, waiting and transportation. The perfect scenario would be to assemble and deliver a product immediately after a customer placed an order. This results in a higher level of system's adaptability, making it better suited to changes of requirements and market fluctuations [Ohn88].

2.2.4 Production Levelling

Ohno claims that production consistency is more effective than rapid changes and fluctuations in manufacturing processes [Ohn88]. Using JIT with innovative management methods and Kanban board makes it possible to implement "production levelling". This is achieved by limiting the capacity of certain stages in the production process, which creates what is referred to as "flow". The reduction in workload on each workflow state helps to reduce inventory and equipment depreciation, which prevents individual constituents of the system from becoming overloaded. As a result of the equal distribution of resources it becomes easier to eliminate any defects present in the system. This in turn also facilitates the elimination of waste related to the production of defective products which, as was mentioned before, is one of the central objectives of *lean* manufacturing.

2.2.5 Teamwork

For years, It has been a common practice for Japanese companies to rotate their employees across different departments in their factories, asking them to work at different stages of the development process. This allows for development of different competencies and expansion of employees' skillsets. According to Ohno [Ohn88] it is more profitable for a company to have a skilled team of cross-functional members rather than a team of specialists. As a result members can be easily replaced because their skills overlap. This creates a higher level of involvement and cooperation within a team, and gives employees a sense of belonging which will be further discussed in the section devoted entirely to cross-functional teams in *agile* and *lean* software development.

The second most important pillar of TPS after *just-in-time*, is the idea of "autonomation" or "automation with a human touch". Ohno argues that machines cannot work independently and need to be maintained and controlled by a skilled workforce. This means that when an abnormal situation arises, the worker would stop the production line and diagnose the problem, learning from it. This allows employees to engage in problem-solving activities, allowing them to become an independent, self-organised and qualified workforce which does not need to rely on external support. Furthermore, the principle of autonomation prevents defective products and overproduction waste. In principle autonomation is a quality control system, which ensures that similar problem will never recur in future [Ohn88].

2.3 Lean Principles in Software Development

The application of lean philosophy to software engineering has been of interest in industry recently. Poppendiecks has rigorously analysed and listed seven lean principles for software engineering [PP03, Hen10] which can be seen in table 2.1. Also according to Poppendieckss the most important principle is the elimination of waste [PP03]. The first column corresponds to the principles of lean software development while the second one contains tools that ca be used to successfully implement them in practice.

Lean principle	**Tools to implement the principle**
Eliminate waste	Seeing waste, value stream mapping
Amplify learning	Feedback, iterations, synchronisation
Decide as late as possible	The last responsible moment, making decisions, options
Deliver as fast as possible	Pull systems, queuing theory, cost of delay
Empower the team	Self-determination, motivation, leadership, expertise
Build integrity in	Perceived integrity, conceptual integrity, refactoring, testing
See the whole	Measurements, contracts

Table 2.1: Lean principles and tools as in [PP03]

2.3.1 Waste in Software Development

Because waste is listed as the most important principle of lean software development, the author decided to describe and discuss its type, source and origin. As mentioned before the two fields are concerned with different types of artefacts. Therefore, the type of waste being present in both fields also differs in certain aspects. Poppendiecks [PP03] as seen in table 2.2 compares the corresponding types of waste in software engineering and manufacturing.

The seven wastes of manufacturing	**The seven wastes of software development**
Inventory	Partially done work
Extra processing	Extra processes
Overproduction	Extra features
Transportation	Task switching
Waiting	Waiting
Motion	Motion (humans, handoffs)
Defects	Defects

Table 2.2: Comparison of wastes present in manufacturing and software development [PP03]

Partially Done Work and Extra Features

Poppendiecks also shows that partially done work tends to become obsolete, and might become a financial and managerial obstacle to finishing projects [PP03]. This is because of the developers' uncertainty as to whether the partially done software will work properly when integrated with the rest of the project. Unfinished development is wasting a company's resources and therefore can carry high financial risk. Also adding extra features to the project which are not needed by a client produces waste which is similar in nature to partially done work. For instance, a company might have an unfinished part of a project which was successfully developed and tested, but unless it can be sold it remains to to be an investment with no return. Consequently, minimising or completely removing extra these two types of waste can be both risk-reducing as well as waste-reducing strategy.

Extra Processes

Software engineering is often full of paperwork, requirements documentation and other documents. Paperwork consumes resources and slows down a team's productivity. According to Poppendiecks [PP03] the worst type of paperwork is the one that carries no value for anybody on the team, but is done merely for procedural purposes. Poppendiecks suggests using a template-based format in order to reduce the size of requirements documentation and rapidly validate and understand its content [PP03]. Also unreasonableness and overburdening people with work produces waste as it can cause task-switching, fatigue and general loss of productivity, and proliferation of defects.

Waiting

Waiting caused by staffing, excessive requirements documentation, reviews, testing and approvals are amongst the biggest wastes in software development [PP03]. In case of a new requirement delays can slow down the team's ability to respond quickly. The amount of damage to the project that can be done by waiting varies between projects. Poppendiecks [PP03] suggests deciding as late as possible in managing software projects as a method of dealing with uncertainty and making informed decisions.

Motion

When the sum of all knowledge of a given team is unevenly distributed, people learn from each other by asking questions, which often results in physical movement. Development and coding requires a great deal of concentration, which is usually lost through movement and task-switching; although [IKA10] argues that team communication should not be considered waste since it generally improves a team's skill-set and involvement in the project. There are also other things that can move like documents, requirements and code which needs to be sent from coders to testers. Also, in the case of documentation of requirements, there are a number of things that are not explicitly expressed. This results in incomplete documentation and people's misinformation which can generate additional waste.

Defects

Defects in software development can either mean a problem lasting three minutes or several months depending on its severity. If the problem is small it can be fixed right away by a developer. It can also be ignored and left in the system without being fixed which will make it even more difficult to fix in later stages. For this reason Poppendiecks advices fixing defects as soon as possible [PP03].

Management Activities

There is no direct correlation between management activities and adding value to a product. However, they do have an impact on waste reduction in an organisation as stated by [PP03]. Project tracking, and control systems also do not add value but help to reduce waste and implement JIT philosophy. This is supported by Ikonen who claims that "the amount and significance of waste can be significantly reduced with the right leadership" [Iko10]. There should be a balance between system complexity and its usefulness for a particular project [PP03]. Poppendiecks also claims that seeing waste is an ongoing process of changing the way people think about things, and is usually easier to identify in a crisis.

2.3.2 Cross-functional Development Teams

As mentioned before a cross-functional team is one of the most important characteristics of a "Lean-oriented" development process. Therefore, as a result of its significance for the methodology it needs to be discussed.

In [Hen10] there are numerous examples supporting self-organised software teams. Most of these sources support autonomy, collaboration and initiative of self-organised and cross-functional teams. Cohen and Bailey for instance categorised the advantages of having an empowered and lean software team. These amongst others include increased job satisfaction and commitment to work, trust in the management, increased participation, and less redundancy [CB97]. Autonomous teams might also prove more effective when projecs are higly innovative and are developed under uncertain and changing circumstances, thus requiring more creativity and inventiveness from their members [Hen10]. This is supported by Pink who in his speech [Pin09], shows the ineffectiveness of reward-based, incentivised production systems, in the context of complex and creative types of tasks. Furthermore, Karhatsu suggests a "subtle" form of control over the team, which can help to sustain team's sponteneity and creativity while strenghtening its sense of direction and purpose. [Hen10, Iko10].

On the other hand other sources cited in [Hen10] question the effectiveness of autonomous software teams. Some of them claim no connection between empowerment and success . Others identify the limitations of such teams in the contex of big organisations [Hen10]. However, big software projects are not the subject of this paper. Karhatsu also shows that the self-organised team can easily become disintegrated, and ineffective if it does not follow certain lean principles [Hen10].

2.4 Components of Kanban in Software Development

According to Kniberg [Kni10] the main principles of Kanban in software engineering are: *a*) Visualising the workflow *b*) Limiting work in progress (WIP); and *c*) Measuring lead time

2.4.1 Visualising Workflow

The main way of visualising the workflow is by using the tool called Kanban board. It can for example be a whiteboard with stickers on it or an electronic card wall system. Figure 2.1 shows a basic Kanban board with the most common sections used. It is divided into columns each of which represents a different stage in the product development. The board in figure 2.1 is divided into five stages each capable of receiving different components of the project. Most Kanbans include their own ordering of columns but typically, each version would have its own version of "To do", "Work-in-progress", and "Done" column. When a request or ticket is added into the "Work-in-progress" section it needs to stay on the board until it reaches the "Done" column. The entire process will be discussed at length in section 2.5.

The Kanban board is the most integral and important element contributing to the system's transparency and its ability to reduce waste. It makes the entire process transparent to the team, exposes waste, bottlenecks, queues and most important all waste [Kni10]. Furthermore,

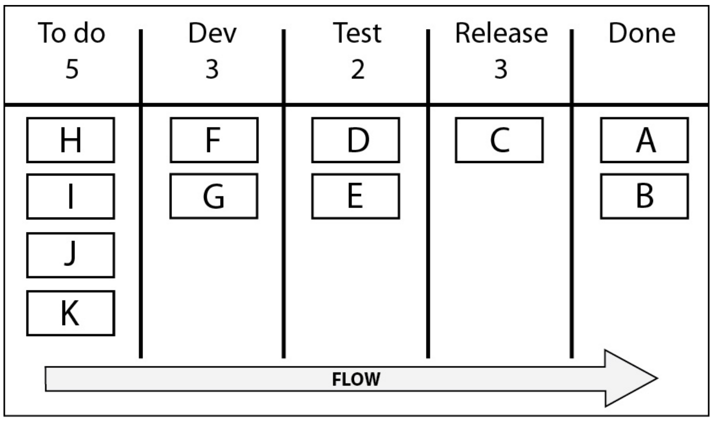


Figure 2.1: Example of a basic Kanban board as in [Kni10].
Numbers show WIP limits for each individual process in the workflow.

from the team's point of view it contributes to cultural change, encouraging collaboration and discussion, which is supported by early case studies [Kni10, Iko10]. This results in constant improvement of the system providing the team with the sense of control and responsibility for the entire project [Kni10].

2.4.2 Limiting work in progress

Kanban prescribes designating a column which is used for the things that are currently being worked on. It needs to be limited based on the teams current performance, cycle times, number of people involved in the project, teams structure and other considerations. Setting the optimum number of items allowed to be in work in progress is essential to the systems efficiency and should be set for each Kanban system separately [Kni10]. Setting the limits too high, will overburden the team with work creating waste and idle tasks. On the other hand, if the limits are set too low, then people can become idle which leads to bad productvity. Kniberg [Kni10] suggests first setting the WIP by taking the number of members in a given team and subtracting one from it. The reason is to increase collaboration and communication within the team [Kni10].

Having a limited WIP allows a team to limit the work it has to its current capacity. The method of limiting WIP is based on the Theory of Constraints which increases teams' responsiveness to prospective problems helping them to finish more items in less time and fewer defects [Kni10].

2.4.3 Measuring cycle and lead time

There is a difference between what the users of Kanban call lead and cycle times. As cited by [Roo10] cycle time is the time measured when the actual work begins on the request and ends when the title is ready for delivery. On the other hand the lead time clock starts when the request is made and ends at delivery. Lead time is more important from the customers point of view while cycle time is the way of measuring process capability by the developers [Roo10]. As it can be seen in figure 2.2 lead time is normally much longer than. It might take developers 100 days to agree on a feature with a client, and result in 1 day of development work.

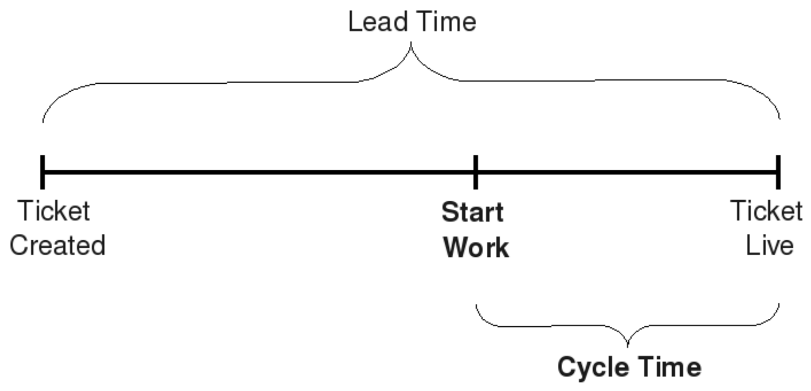


Figure 2.2: Cycle and lead times as in [Roo10]

Brodzinski [Bro10b] suggests measuring cycle time from the moment that an item is placed in the WIP and ends up in the ‘‘done” column, rather than as from the‘‘backlog” column. As a reason for that he states that his team can often shuffle the requests in the ‘‘backlog” column which makes it difficult to measure the cycle time precisely.

Measuring cycle time allows developers to measure the systems throughput and provides them with an indication of its performance. It also allows for the effective allocation of a teams resources. For example when a team does not complete a specified number of tasks within some period of time it might indicate that the WIP limits are set too high, or the number of columns is improperly organised. The cycles should not be too long nor too short to allow receiving regular feedback from the client and giving the team enough time to make a high quality component for client demonstration. It also allows developers to make predictions about the completion of each task, thus managing client’s expectations more effectively [Kni10].

2.5 System in Operation

2.5.1 Setting the system

Before the actual work can be started the team members need to be assigned roles. For the purpose of discussion let us suppose that there is a team consisting of four members. There is Andy who is a project manager and developer, John and Mark who are also developers and Emma who acts as a tester. Andy is reponsible for The ‘‘Release” column which is used for integrating the software modules and accepting them by the client. He also supervises the project and arranges the tickets in the ‘‘To do”. The order depends on the client’s current priorities. Also, the team that has just been described is cross-functional which means that each of its members can perform a different role within a team if necessary.

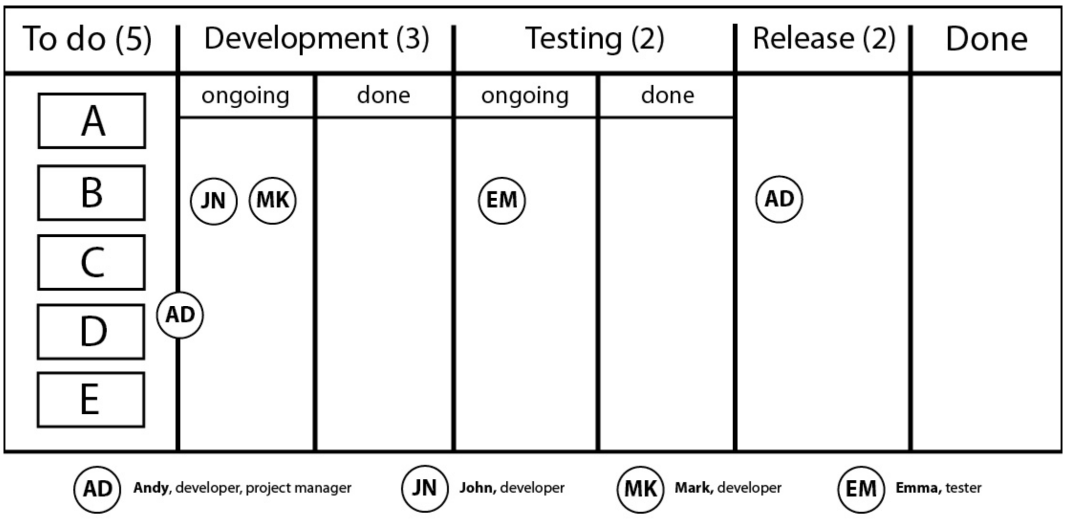


Figure 2.3: Initial Kanban board.

As shown in figure 2.3 the team decided to split their workflow into five sections. ‘‘To do” section can hold up to 5 requests. Columns ‘‘Development” and ‘‘Testing” were also split into two additional sections ‘‘ongoing” and ‘‘done”. Kniberg [Kni10] suggests this solution as a way of indicating which orders are completed, and can be moved to the subsequent process. The team also has decided to set the WIP limits in ‘‘Development” column to 3, because of having two nominal developers with one project manager who as mentioned before can work as a developer. Moreover, the team decided to set ‘‘Testing” and ‘‘Release” section’s WIPs to 2 since the team has got only one nominal tester and testing is relatively quick.

Finally, before the features can be posted on the Kanban board, first the project in agreement with the client needs to be splitted into user stories in the form of Minimum Marketable Features (MMF) [DC03, Coh04].

2.5.2 Kanban Working Properly

Firgure 2.4 is an example of Kanban working properly without any unexpected problems. In this case the tickets can easily move from the left side of the board to the right through every stage of the process. Once the ticket was selected as a priority it goes to the ‘‘Development” section after which it waits in the ‘‘done” column to be moved into the testing phase. It can happen only if the WIP is not reached by the ‘‘testing” phase. The entire process is self-explanatory and can be easily understood by looking at figure 2.4.

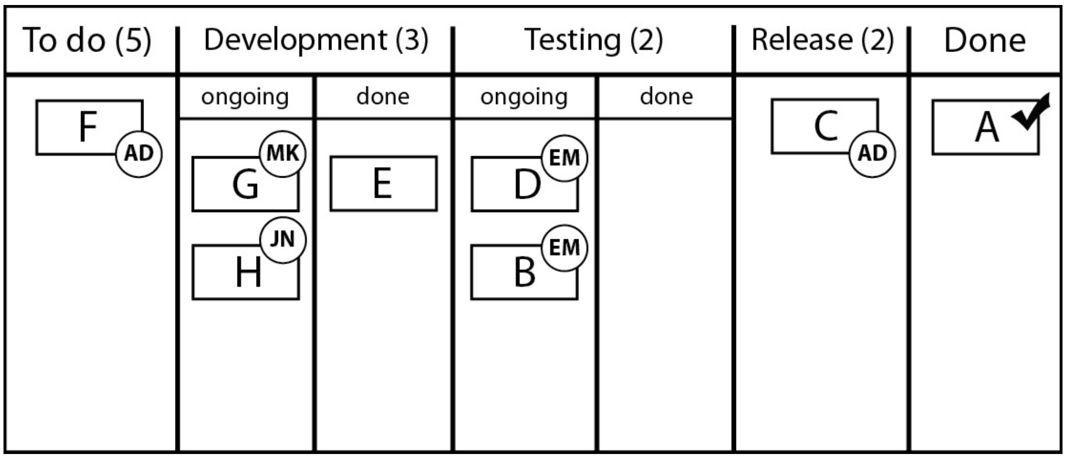


Figure 2.4: Kanban board with an uninterrupted workflow.

2.5.3 System Malfunctioning

When an abnormal situation arises items can no longer move freely across the board. As can be seen in figure 2.5 there is a problem with item ‘‘B” which is referred to as a ‘‘blocker”. In the process of testing the items, Emma reported item ‘‘B” as being unable to compile. She tried to fix the problem herself but failed after which she decided to report the problem through the Kanban board. Thanks to the visual nature of Kanban the team is able to immediately identify the problem and respond accordingly.

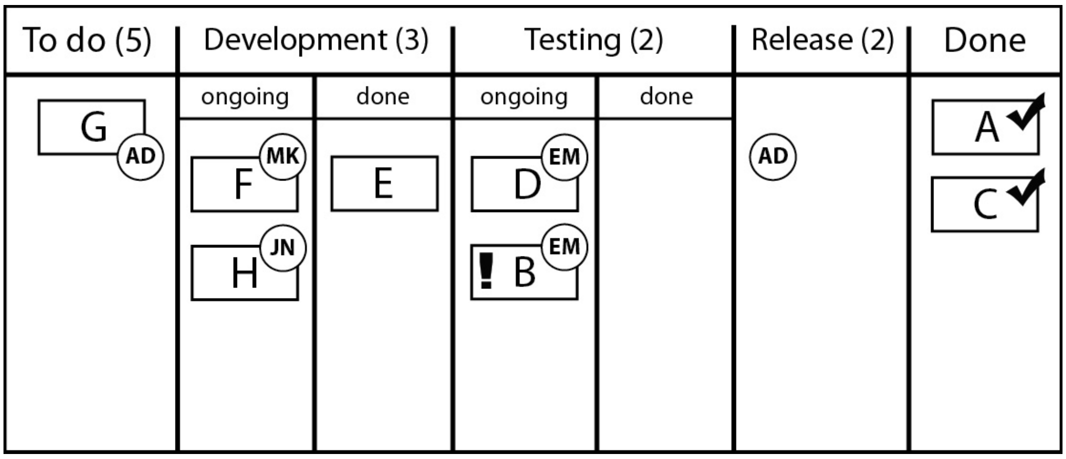


Figure 2.5: Kanban board blocked by a process.

2.5.4 Fixing the System

In order to remedy the situation, relocation of the team’s resources is required. As it can be seen in figure 2.6 due to the ‘‘blocker” being present in the system, Andy, Mike and John are either completely idle or unable to utilise their full capacity.

Since John and Mike were involved in the development stage of the item mentioned, their expertise can help prove useful in solving the problem. Furthermore, when the team is working together on the bottleneck they are likely to learn form each other and contribute to the team's overall competency and understanding [Kni10].

Therefore, Andy decides to send John to assist Emma in fixing the problem and, since he finished working on item ‘‘H”. As seen in figure 2.6 Andy is being idle in the ‘‘Release” section of the process, therefore his skills can be utilised to test item ‘‘D” in order to make it ready for moving to the ‘‘Release” section.

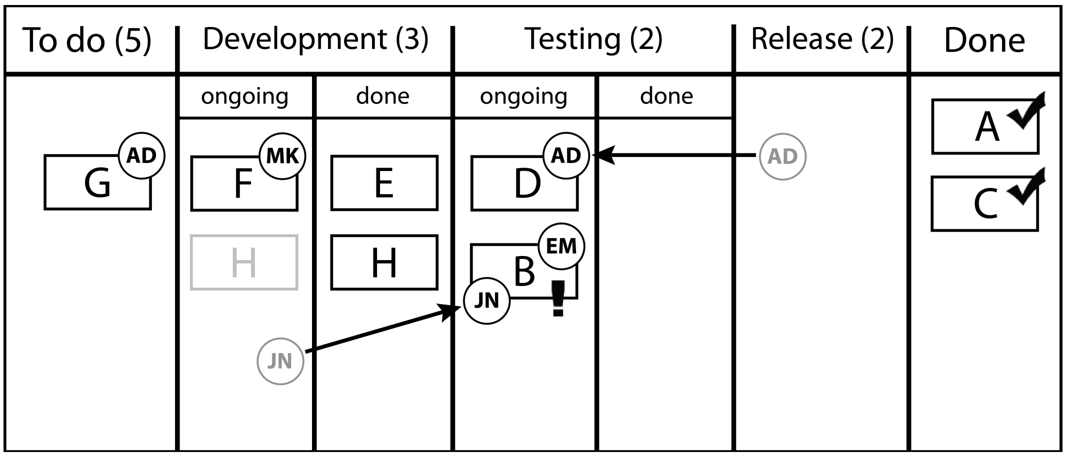


Figure 2.6: Kanban board with a blocker and team's response

Chapter Summary

In this chapter a description of the software industry has been presented with its history, current trends and recent developments. The section's primary focus was to describe the philosophy and application of *lean* production methods developed by Toyota Corporation to the domain of software engineering. It described in detail the most recent adaptation and characteristics of Kanban to the domain of software engineering. The chapter also described how Kanban system might be used in the context of small development teams. It also described how small teams' capacity can be fully utilised by the effective use of Kanban.

Chapter 3

Research Design and Methodology

The purpose of this section is to describe the theoretical framework behind the project. It will describe the methods to be used in order to collect, validate and interpret the data. Each of these methods has its applicability in the field and risks. Consequently, the reasons for choosing them will be provided and possible ways of mitigating their risks listed. The section will also address the risks associated with the project itself and outline the plan for its implementation.

3.1 Research Methodology and Data Collection Methods

Although it is possible to use more than one type of data, the author decided that the main approach of his research will be qualitative. This is due to the project's nature which will involve extensive descriptions of mechanisms and meanings which are more suitable to be analysed qualitatively [Hen10]. Moreover, in qualitative research it is possible to make use of existing archival materials which is suitable for this experiment [Yin03] .

The empirical part of this research will consist of three case studies. The first two will be conducted on the author's own projects. These include, a small software project developed in the course of his masters degree and a project which aims at monitoring the process of his dissertation writing. The third case study will be conducted on a group of students from the School of Computer Science at The University of Manchester who will be using Kanban as a part of their dissertation projects. It will also include data from documented case studies which are publicly available. The research methodology will mostly consist of extensive interviews, direct observation, and the analysis of written records. The approach of the experiment is to analyse projects of different scale and lenght in order to perform a comparative analysis.

3.1.1 Systematic Self-Observation

The goal of Systematic Self-Observation (SSO) is the generation of field notes that are accurate descriptions of participants' experience. They are highly subjective, and the observer is encouraged to describe the experience in his or her own words. According to [RR02] it is an adaptable, practical, economic and productive research strategy as well as a legitimate source of information and insight. The method can be successfully applied to the study because of its minimal financial requirements. What is required from the researcher are critical thinking skills and the ability to accurately record his insights of a given phenomenon. The data will be collected by making notes, photographs and audio recordings. One of the major risks of this method is its subjective nature which can lead to scientific biases. For instance a confirmation bias which is a tendency of people to favour information that confirm their hypotheses regardless of its credibility. The next subsection will explain the ways of reducing this risk.

3.1.2 Direct Observation

There are two types of observation that are available to the researcher. The first one is called "Participant observation" which is performed by an investigator immersing himself in a group or context being observed. The method allows an observer to query certain aspects of the observed situation and directly influence participants' decisions and actions [Tro06]. Despite the method's ability to yield a meaningful and comprehensive data. One of its main disadvantages especially in the context of this research is a considerable amount of time needed to establish a rapport with its subjects, and become a part of their culture [Tro06]. This aspect could prove impossible to implement, mainly because of the short duration of the MSc project.

The second type is called "Direct observation" which is a non-intrusive way of observing subjects' behaviours in their natural environments [Tro06, Yin03]. In contrast to participant observation this method requires an investigator to become detached from his or her object of study. This non-intrusive form enables researchers to use more objective ways of inquiry [Tro06]. It also does not require longitudinal preparations which is relevant to this experiment. Moreover, it tends to be more focused than participant observation [Tro06].

Both of these methods are not free of possible scientific biases. For instance the confirmation bias that was mentioned before can occur in both methods. Moreover, students might be subject to the so called Hawthorne effect which happens in response to the fact that participants are being studied. The effect manifests itself in change of participants' behaviour which leads to unreliable results. Another challenge is related to the capturing of data which might overload the observer's memory. The risk can be minimsed by developing a template for making notes and by a careful design of an experiment.

As suggested by [Yin03] the triangulation of data sources can be an effective way of minimising the risks mentioned. Following this, the author intends to use multiple sources of evidence in order to increase the objectivity of the observation. These will include daily notes from all experiments which will be published on his blog [Rol11] on peoples' behaviours, their decisions made, and how the project will be progressing. Together with thematic interviews it should serve as a complete framework of gathering data.

Study context

The main participants in the experiment will be MSc students from the School of Computer Science at the University of Manchester. There are seven students participating in the project and their programming experience is on average two years. They are involved in the development of Personal Learning Environment (PLE) and supervised by Dr. Mark Van Harmelen. Choosing postgraduate students as the equivalent of professional programmers is supported by the findings of [HRW00]. He claims that there are minor differences between the levels of technical competency of students and professionals [HRW00].

3.1.3 Thematic Interviews

Another method of eliciting information from the users are interviews. Interviewing can be categorised into structured, semi-structured or unstructured. For this kind of research the semi-structured interview is more appropriate since it allows the interviewer to interpret the answers and decide which questions to ask, or how to modify the old ones so they can yield more data.

In semi-structured thematic interviews the interviewer concentrates on certain themes instead of specific questions [Hen10]. The result is that the questions inside given themes may vary which allows for greater flexibility in eliciting information from the interviewees. The purpose of the interview for this paper will be recognising sources of waste in small software projects. Its purpose will also be to ask participants whether they find the system useful for their purpose or what complaints they might have related to its functionality.

The interviews will be done right after the projects as to increase the reliablity of the data through better recall of past events by the participants. They will last about half an hour and will be recorded in audio. The reliablity of thematic interviews might be further increased by creating a good framework for interviews [Hen10] which will include several possible questions for each theme. Furthermore, the validity of each answer will be checked against the data gathered by the author. This will be done by asking an interviewee to provide the interviewer with descriptive examples supporting his answer.

3.1.4 Archival Case Studies

In order to stay abreast with the latest innovations in the field of lean software development, this research will make an extensive use of the recent sources of information about the domain. This will be conducted by the use of blogs and case studies that are available on the internet. Also, when making use of the online material one has to be aware of its elusive nature. For this reason, the author decided to make use of the sources which have been available on the internet for at least a year. The author also decided to stay in contact with their respective authors, should he need to inquire about some data related to their projects.

Two sources seem particularly suitable for this purpose. The first one is a blog called *Kanban story* which is being edited by Pawel Brodzinski who is a project manager and quality engineer responsible for managing a firm responsible for the development of internet banking applications. He and his team are actively using Kanban in their work. Their experience with the method and insights are extensively documented on Pawel's blog [Bro10a]. Moreover, the author is always available to share his opinions and explain certain methods that his team is using which is helpful. The second source is a case study from the *Software Factory* which is a software engineering research setting at the University of Helsinki [Abr10]. It is described by its authors as an advanced R&D laboratory for conducting software projects. The case studies from this project can also be successfully used by the author to gather data and the analysis of existing case studies. All of the projects mentioned vary in size and complexity. This wide spectrum of project's scope will provide the author with an opportunity for their objective evaluation and comparison.

3.2 Data Analysis and Evaluation

After the project has been completed the data from different sources will be consolidated and conclusions will be drawn. The author will seek to understand how Kanban impacts software projects of different sizes, from the team of one person to the one of four people. The hypothesis is that as the group dynamics change and there will be less communication pathways involved, the cooperation and team's learning might be further amplified, together with the elimination of waste related to decision-making and waiting. On the other hand since there are less people involved it might result in smaller throughput of the entire team causing the system to be more of a burden and less of a helpful tool. These will be answered after the data from different sources has been collected and critically analysed against each other. Consequently, the conclusions about the effectiveness of Kanban for small software projects and recommendations for its use will be made.

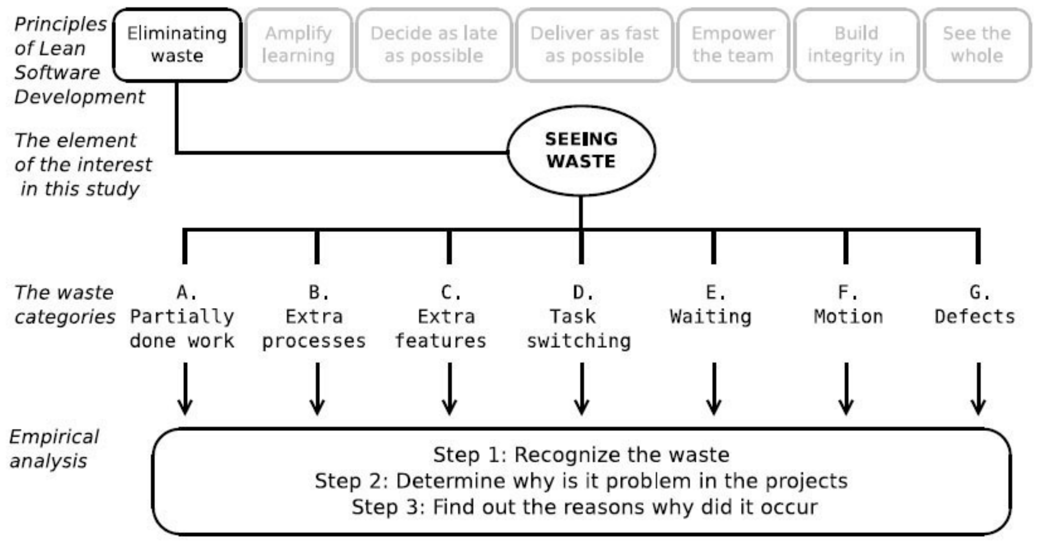
3.2.1 Project Success Framework

It is still a question of debate what constitutes a successful project. Different people's opinion of this issue depends on their role within a given organisation. As shown by Shenhar [SDLM01] projects can be financially successful even if initially they are considered failures. In fact most high-tech projects are perceived as most likely to create overruns, due to their high risks prospective return on investment [SDLM01].

The success of the PLE project will therefore be evaluated using the first and the second dimension of Shenhar's model of project sucess [SDLM01] which measures (1) project efficiency (meeting schedule and budget) and (2) impact on the customer (meeting functional performance, meeting technical specification, fulfilling customer needs, solving a customer's problem, and finally customer satisfaction. This will be performed by a product-owner and a person who acts as a client.

3.2.2 Waste Analysis

The author will also seek to analyse waste present in software projects of different size using a framework developed by [IKA10] figure 3.1. Following Poppendieck's and Ikonen's guidlines [PP03, IKA10] the first step is to recognise waste corresponding to one of the seven categories of waste. Step two is concerned with determining the source of waste and the final step involves finding its causes. This will be evaluated by three different team members each of them representing different roles: less experienced, more experienced, and the team leader. The reason behind this is to create more perspectives and greater objectivity of the method. As mentioned before the data collection will be done using thematic interviews, asking users to provide arguments supporting their claims related to their recognition of waste.


Principles of Lean Software Development
Eliminating waste
Amplify learning
Decide as late as possible
Deliver as fast as possible
Empower the team
Build integrity in
See the whole
The element of the interest in this study
SEEING WASTE
The waste categories
A. Partially done work
B. Extra processes
C. Extra features
D. Task switching
E. Waiting
F. Motion
G. Defects
Empirical analysis
Step 1: Recognize the waste
Step 2: Determine why is it problem in the projects
Step 3: Find out the reasons why did it occur

o

Figure 3.1: Waste analysis model suggested by [PP03]

3.2.3 Kanban Oriented Productivity Tools

The project will also compare different productivity tools that are available to the practitioners of Kanban. Some examples of this wide range of productivity tools available include: Teams Board [Tar], Kanbanery [Pol] and Kanban Tool[Lab]. These seem to be amongst the most widely used by *lean-oriented* teams. These tools are also available on various portable devices i.e. *iPhone, iPad, BlackBerry, Android operating system based phones* etc. The author will critically review the most popular tools and make recommendations about the most appropriate for a particular team or project size. This will be conducted on his own projects and also by critically evaluating different case studies that were mentioned before.

3.3 Project plan

As can be seen in table A.1 the project to be undertaken can be divided into five phases including: literature survey, background report writing, methodology implementation and testing, case studies, methodology evaluation and finally the dissertation writing phase. The main purpose of the literature review is a comprehensive acquisition of information related to the techniques, methodology, structure and history of the Kanban system together with the knowledge of Agile Software Development. Its other purpose will be to survey research methods used by researchers to evaluate different project management methods.

From the middle of March the writing of this paper has been being done which have been running simultaneously with the next phase. Methodology implementation and testing will take place in two stages. The first one is focused on self-observation and testing Kanban on a small software project that will be a part of the Developing Web Applications" module. The next part of this stage is focused on managing the author's dissertation using the system. The second stage will involve the small team actively using the Kanban system. Its purpose is to gather data and assess case studies which will have been gathered before dissertation writing. The final stage will be the dissertation writing which will be further subdivided into phases which will run in parallel with testing the system by the author.

Chapter Summary

This chapter discussed different methods of data collection and its evaluation. It first presented the different methods of collecting data ranging from observation of case studies, surveying of case studies from literature and performing interviews. The advantages and challenges of all the methods were listed and reasons behind choosing them were elaborated upon. The section also presented the evaluation framework which will be used to critically analyse each of the project to be undertaken in the course of this study. Finally, the plan for the implementation of the project was given.

Bibliography

[Abr10] Pekka Abrahamsson. Unique infrastructure investment: Introducing the software factory concept. *Software Factory Magazine*, (1):2--3, 2010.

[Bro10a] Pawel Brodzinski. The kanban story:. `http://blog.brodzinski.com`, May 2010. Accessed 02/05/2011.

[Bro10b] Pawel Brodzinski. The kanban story: Measuring lead time. `http://blog.brodzinski.com/2010/05/kanban-measuring-lead-time.html`, May 2010. Accessed 23/04/2011.

[CB97] Susan G. Cohen and Diane E. Bailey. What makes teams work: Group effectiveness research from the shop floor to the executive suite. *Journal of Management*, 23(3):239 --290, June 1997.

[cha94] The CHAOS report. Technical report, Standish Group International, 1994.

[Cha05] Robert Charette. Why software fails. `http://spectrum.ieee.org/computing/software/why-software-fails/5`, April 2005. Accessed 21/04/2011.

[Coh04] Mike Cohn. *User stories applied: for agile software development.* Addison-Wesley, March 2004.

[DB01] Chris Cowdery Don Bradley. Small business: Causes of bankruptcy, 2001.

[DC03] Mark Denne and Jane Cleland-Huang. *Software by Numbers: Low-Risk, High-Return Development.* Prentice Hall PTR, October 2003.

[EV10] J. Laurenz Eveleens and Chris Verhoef. The rise and fall of the chaos report figures. *IEEE Software*, 27:30--36, 2010.

[Hen10] Karhatsu Henri. Building a self-organizing software development team: Multiple case study. Masters thesis, University of Helsinki, Helsinki, May 2010.

[HRW00] Martin Höst, Björn Regnell, and Claes Wohlin. Using students as subjects a comparative study ofstudents and professionals in lead-time impact assessment. *Empirical Softw. Engg.*, 5:201--214, November 2000.

[IKA10] Marko Ikonen, Petri Kettunen, and Pekka Abrahamsson. Exploring the sources of waste in kanban software development projects. In *Software Engineering and Advanced Applications, Euromicro Conference*, volume 0, pages 376--381, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[Iko10] Marko Ikonen. Leadership in kanban software development projects: A quasi-controlled experiment. In Pekka Abrahamsson and Nilay Oza, editors, *Lean Enterprise Software and Systems*, volume 65, pages 85--98. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Kni10] Henrik Kniberg. *Kanban and Scrum - making the most of both*. Lulu.com, 2010.

[Kot96] John P. Kotter. *Leading Change*. Harvard Business Press, January 1996.

[Lab] Shore Labs. Kanban tool. `http://kanbantool.com/`. Accessed 08/05/2011.

[McC04] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, June 2004.

[MT10] Tom McCall and Ben Tudor. Gartner says worldwide IT services revenue declined 5.3 percent in 2009. http://www.gartner.com/it/page.jsp?id=1363713, May 2010.

[Ohn88] Taiichi Ohno. *Toyota production system: beyond large-scale production*. Productivity Press, 1988.

[Pin09] Daniel Pink. Daniel pink on the surprising science of motivation. `http://youtu.be/rrkrvAUbU9Y`, August 2009. Accessed 01/05/2011.

[Pol] Lunar Logic Polska. Kanbanery. `http://kanbanery.com/`. Accessed 08/05/2011.

[PP03] Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley, 2003.

[Rey08] John C Reynolds. Some thoughts on teaching programming and programming languages. *ACM SIGPLAN Notices*, 43:108110, November 2008. ACM ID: 1480852.

[RM00] M. L Russ and J. D McGregor. A software development process for small projects. *Software, IEEE*, 17(5):96--101, October 2000.

[Rol11] Lucas D. Rola. Kanban for small software projects - project blog. `http://scrumban.co.cc/`, February 2011. Accessed 02/05/2011.

[Roo10] Stefan Roock. Lead time and cycle time. `http://stefanroock.wordpress.com/2010/03/02/kanban-definition-of-lead-time-and-cycle-time/`, March 2010. Accessed 02/05/2011.

[Roy98] Walker Royce. *Software project management : a unified framework*. Addison-Wesley, Reading Mass. [u.a.], 1998.

[RR02] Noelie Rodrguez and Alan Ryave. *Systematic self-observation*. SAGE, January 2002.

[SB01] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[SDLM01] Aaron J. Shenhar, Dov Dvir, Ofer Levy, and Alan C. Maltz. Project success: A multidimensional strategic concept. *Long Range Planning*, 34(6):699 -- 725, 2001.

[SG05] Andrew Stellman and Jennifer Greene. *Applied software project management*. O'Reilly Media, Inc., November 2005.

[SS05] Catherine Soanes and Angus Stevenson. *Oxford Dictionary of English (Dictionary)*. Oxford University Press, August 2005.

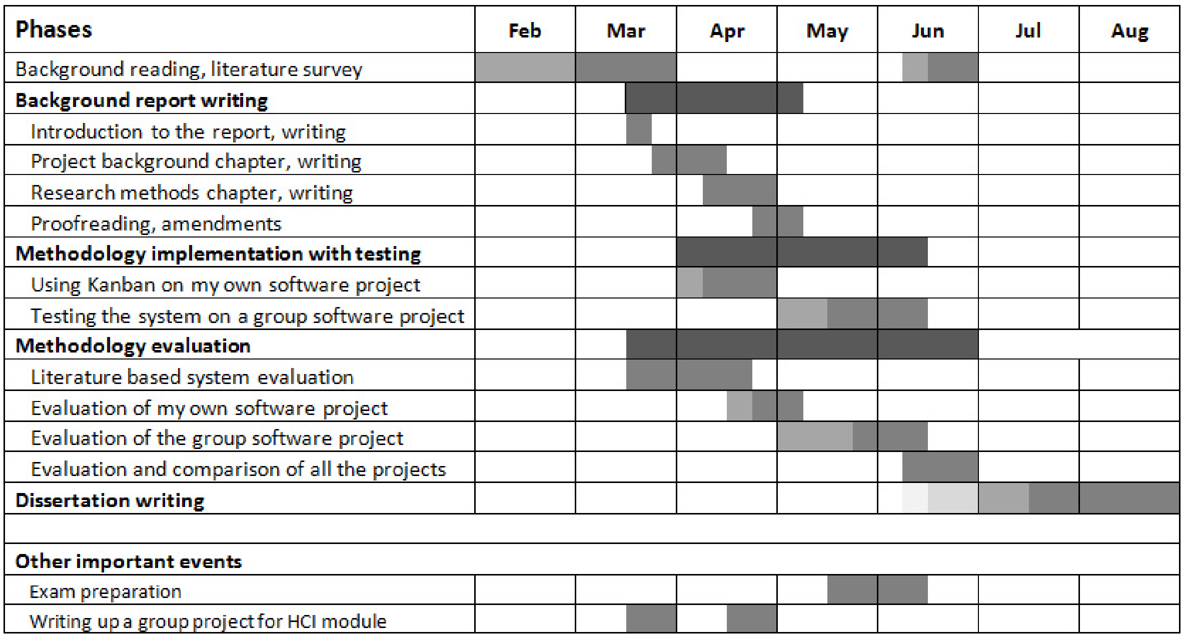
[Tar] TargetProcess. Teams board. `http://targetprocess.com/Product/agile_tour/teamsboard.aspx`. Accessed 08/05/2011.

[Tro06] William M.K. Trochim. Web centre for social research methods. `http://www.socialresearchmethods.net/`, November 2006. Accessed 02/05/2011.

[Yin03] Robert K. Yin. *Case Study Research: Design and Methods, Third Edition, Applied Social Research Methods Series, Vol 5*. Sage Publications, Inc, 3rd edition, December 2003.

Appendix A

Gantt Chart of the project



Phases	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**
Background reading, literature survey	▨	▨			▨		
Background report writing		▨	▨	▨			
Introduction to the report, writing		▨					
Project background chapter, writing		▨	▨				
Research methods chapter, writing			▨				
Proofreading, amendments			▨	▨			
Methodology implementation with testing			▨	▨	▨		
Using Kanban on my own software project			▨				
Testing the system on a group software project				▨	▨		
Methodology evaluation		▨	▨	▨	▨		
Literature based system evaluation		▨	▨				
Evaluation of my own software project			▨	▨			
Evaluation of the group software project				▨	▨		
Evaluation and comparison of all the projects					▨		
Dissertation writing					▨	▨	▨
Other important events							
Exam preparation				▨	▨		
Writing up a group project for HCI module		▨	▨				

Table A.1: Gantt Chart of the project