UiO : Department of Informatics
University of Oslo

Do Work in Progress (WIP) - Limit in Agile Software Development Matter?

Truls Skeie

Master's Thesis Spring 2014

Do Work in Progress (WIP) - Limit in Agile Software Development Matter?

Truls Skeie

24th January 2014

Abstract

Contents

List of Figures

List of Tables

Preface

Part I

Introduction

Chapter 1

Background

In this master thesis the main topics will contain analyzing of the gathered data from Software Innovation (SI) in order to see if WIP - limit it agile methods matters. SI is a Scandinavian software company that delivers Enterprise Content Management applications.From 2008 to 2013 SI gathered information about each task that was develop. The main reason SI gathered the data was to see if Kanban was more sufficient than Scrum for their use. For interested reader, the case study can be found in the article "Quantifying the Effect of Using Kanban versus Scrum: A Case Study" (Sjøberg, Johnsen and Solberg, 2012). In this thesis the data will be used to determine if WIP-limit matters in agile methods.

In the this chapter there will be a brief introduction of Scrum, Kanban, affiliated tools and to Software Innovation.

Research in the field of Work In Progress (WIP) and if WIP matters in software development, lacks proper research. But some similar research has been done, Giulio Concas and Hongyu Zhang for instance has done research on the difference between limit WIP and unlimited WIP (Concas et al., 2013) and David Anderson, Giulio Concas, Maria Ilaria Lunesu, and Michele Marchesi also highlighted the difference between limit WIP and unlimited process in the article 'Studying Lean-Kanban Approach Using Software Process Simulation' (Anderson et al., 2011)

How to find the best WIP in software development (SD) for a given interval and context also lacks proper research, but in manufacture business some research has been done. Taho Yanga, Hsin-Pin Fub, Kuang-Yi Yanga stated that WIP could be defined as; $WIP = cycletime * throughput\ rate$ in manufacture business (Taho Yanga and Yanga, 2007).

1.1 Kanban

”We can define Kanban software process as a WIP limited pull system visualized by the Kanban board” (Anderson et al., 2011).

Kanban system focus on;

- Continuous flow of work
- No fixed iterations or sprints
- Work is delivered when it’s done
- Teams only work on few tasks at the time specified by the WIP limit
- Make constant flow of released tasks

(ibid.).

Toyota production system introduced Kanban as a scheduling system for lean and just-in-time (JIT) production during late 1940’s and early 1950’s in order to catch up with the American car industry. The Kanban method combined with the lean approach was a success for Toyota. The success was noticed the software development industry among other. In the last ten years more and more software development companies have started to implement agile methods such as Scrum and Kanban (Conboy, 2009), (Ohno, 2001).

More and more software projects adapt to Kanban, and this is one of the reasons why this thesis will focuses on Kanban and WIP. Kanban is one of the agile method in the wind these days, and is used with Lean Software development which is one of the fastest growing approaches in software development (Anderson et al., 2011) One of the most important people in Kanban software development, David Anderson also referred to as ”father of Kanban in the software development industry” (Gupta, 2013) and author of the book "Kanban: Successful Evolutionary Change for Your Technology Business" once stated ”If you think that there was Capability Maturity Model Integration, there was Rational Unified Process, there was Extreme Programming and there was Scrum, Kanban is the next thing in that succession.” (Leonardo Campos, 2013)

The Kanban method splits one big problem into many small pieces of problems. When the small pieces are defined by the team, the problems are put up on the Kanban-board to visualize the problems and see potential bottlenecks. When people started to understand Kanban, they easily discovered where the bottlenecks where (Shinkle, 2009).

One of the main difference between Scrum and Kanban is estimation, in simulation of software maintenance process, with and without a work-in-process limit (Concas et al., 2013) estimation was defined to be the main source of waste. In their research, they find out, if they let the developers work with small tasks at time and not be interrupted, they will be more effective. The developers in this case was interrupted when they was assigned to estimate tasks. The research groups decided to implement Lean-Kanban, which includes minimizing waste, which meant estimation for this case. After implementing Lean-Kanban the teams increased the ability to perform work, lower the lead time and meet the production dates.

1.2 Lean-Kanban

The Lean approach was introduced between 1948 and 1975 in manufacturing work in Japan. It was designed to find and eliminate waste, so the manufacturing could deliver value to the costumer more efficiently. In 2003 Mary and Tom Poppendieck first introduced Lean thinking to software development, they published 'Principles of Lean Thinking' (M. Poppendieck and T. Poppendieck, 2003). Poppendieck stated that an important tool to manage work flow is the concept of pull-systems, which means tasks are put in production only when a costumer asks for it (M. Poppendieck and T. Poppendieck, 2009). In the recent years, Kanban has been introduced more and more to software development, and is becoming one of the keys to Lean practice in the field (Anderson et al., 2011).

1.3 Scrum

The Scrum framework is the source of much of the thinking behind principles and values of the Agile Manifesto. Values as "Individuals and interactions over processes and tools", "Working software over comprehensive documentation", "Customer collaboration over contract negotiation" and "Responding to change over following a plan" relates directly to Scrum. (Alliance, 2012).

Scrum have three main roles, the Product Owner, the Scrum Master and the members of the development team. The Product owner in collaboration with the Scrum Master decides which work to be prioritized in the backlog. A backlog represents the tasks to be done in order to complete the project. The Scrum Master acts like a team leader and helps the team and organization to take best advantages of Scrum. The development team works on tasks specific for the sprint there in (ibid.).

Sprint is a time-boxed interval over a given time. The Scrum framework

suggests the duration of sprints to be from one to four weeks. Before each sprint, a sprint planning meeting is conducted, with all the team members attending. A Sprint planning meeting is held so the team can discuss tasks from the backlog and come to an agreement of which tasks to be put in the minimal backlog. (Alliance, 2012).

In each sprint a minimal backlog is created so the developer knows which tasks to work on. The Product Owner and the team members discuss and decide which tasks from the backlog to be added to the minimal backlog. After the minimal backlog is full, the Product Owner and the team members discuss each task in order to get a better and a shared understanding of what is required in order to complete the task. One of the main principles in Scrum is that it requires that a new feature is ready for release after a sprint. The feature should be a visible part of the product in order to get feedback from end-users. So all the tasks in the minimal backlog combined should be a visible of the product. (ibid.).

1.4 Kanban Board

”The Kanban board makes it clear to all the team members the exact status of progress, blockages, bottlenecks and they also signal possible future issues to prepare for”(Middleton and Joyce, 2012).

The Kanban board is one of many important tools in Kanban. It’s used to control the WIP, increase the information flow with visualization and spot bottlenecks (Concas et al., 2013). A Kanban is board is illustrated in figure 1.1. Each column has an intuitive name in order to describe itself so the developers easily can track where each task is. In figure 1.1 each column is named "Backlog", "In progress" and "Done". Each column can have a WIP-limit to specify how many works in progress there are allowed in the column (Middleton and Joyce, 2012). In figure 1.1 the WIP-limit is stated under the column name. The backlog columns have a WIP-limit of 4, In progress has 5 and obviously done doesn’t have a WIP-limit.

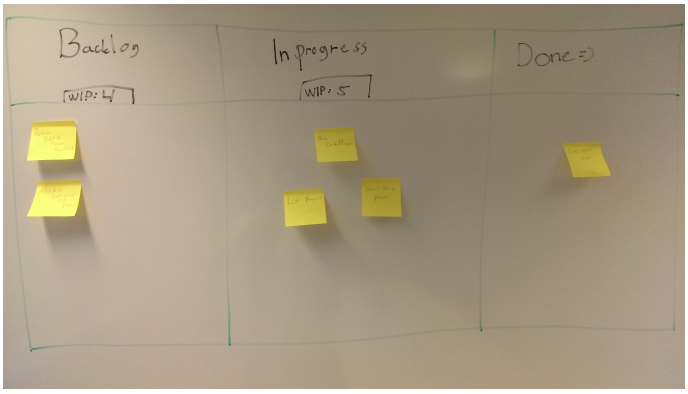

Figure 1.1: Example of a Kanban board

1.5 Lead time

”Lead time is the total elapsed time from when a customer requests software to when the finished software is released to the customer” (ibid.).

The citation by Joyce above is close to definition of what lead time is. This definition could be useful for consultancy companies, but for SI, who is a big in-house development company with few releases each year this definition is unsuitable. Quantifying the effect of Using Kanban versus Scrum stated two reason why this is unsuitable for SI:

"First, the amount of time a work item remains in the backlog queue before it’s put on the board is a function of priority, not whether the company uses Scrum, Kanban, or other development methods. Furthermore, companies that develop and sell products to many customers might propose new features themselves and put them on the backlog before any customers request them. Second, given a policy of two or three releases a year, the result of a work item isn’t delivered to the customer immediately after it’s finished” (Sjøberg, Johnsen and Solberg, 2012).

Lead time is an essential ingredient when you look for the optimal WIP. Often in project, lead time is split into pieces, so every task has its own lead-time; this gives the development teams the advantages to experiment with different WIP’s in order to see the different lead-times and then measure which WIP that suits this project the best.

1.6 Just-In-Time

"Just-In-Time is based on delivering only the necessary products, to the necessary time and the necessary quantity." (Lai, Lee and Ip, 2003).

Just-In-Time was introduced 30 years ago by Toyota Motor in combination with Lean. JIT has been developed to increase productivity through waste reduction and increasing the value added on the production processes. In one of the books by Mary and Tom Poppendieck the JIT principle is explained (M. Poppendieck and T. Poppendieck, 2006). To explain, illustrate and visualize the JIT principle Mary and Tom Poppendieck uses the picture 1.2 (Lai, Lee and Ip, 2003) (M. Poppendieck and T. Poppendieck, 2006).

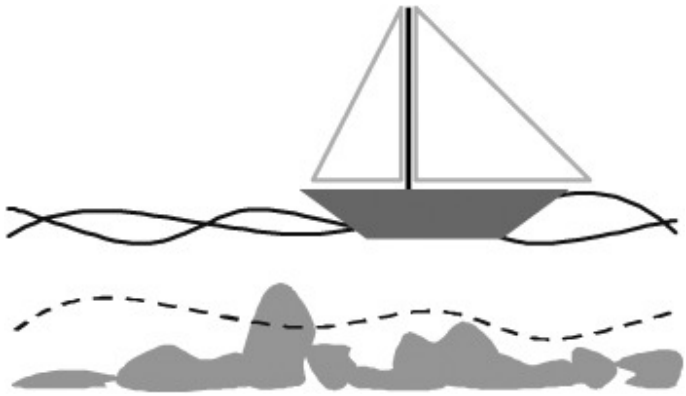

Figure 1.2: JIT example

In the picture 1.2 the stream reflects the inventory. Under the stream, there are rocks located in different sizes. The rocks illustrates waste and problems that can occur. If the stream level is lowered, the rocks are more visualized. At this point you have to clear out rocks in order to make the boat continue it's journey, or it will crash into the rocks. After the rocks are cleaned out, you can lower the stream level again and continue until it's just pebbles left

If we lower the stream, problem and waste will become visible. But why do Lean want to lower inventory in order to make problems and waste occur? Because when problem and waste occurs, you are able to fix the problem and remove the waste. Fixing the problem and removing the waste can have several benefits such as, your process could be optimized and you are on step closer to have zero problems and zero waste. (Lai, Lee and Ip, 2003) (M. Poppendieck and T. Poppendieck, 2006).

1.7 Throughput

"The output of a production process (machine, workstation, line plant) per unit time (e.g., parts per hour) is defined as the systems throughput or sometimes throughput rate" (Adams and Smoak, 1990)

The concept of throughput in order to measure how productive teams,

people or companies is similar in both software development and manufacturing. On the other hand, in software development each task is more abstract, tasks can have different solutions depending on how the team or developer approaches the task and the tasks is usually only done once.

In manufacturing, each task usually has one solution and when the solution is found the physical item is mass produced. Adam said "Throughput in plant, line or workstation, is defined as the average quantity of good parts produced per unit time" (ibid.), which gives a good example of the relationship between tasks in manufacturing and software development. In manufacturing the part either fits its purpose (good) or not (defective), in software development a tasks can fit a purpose, but the purpose may be wrong.

In software development throughput is measured in number of finished delivered tasks per day, week, month, quarter or year. A key factor in successfully measuring throughput is to specify the size of each task. If the size is not specified before, each tasks has to require almost the same amount of work.

In software development throughput is measured in number of finished delivered tasks per day, week, month, quarter or year. A key factor in successfully measuring throughput is to specify the size of each task. If the size is not specified a developer x can have throughput of 1, but another developer y could have throughput of 3 and they still have done the same amount of work and this will give wrong results if the throughput is measured. So its recommend to specify the amount of work for each task in advanced.

1.7.0.1 Example of throughput measurement

This is a simple example to illustrate throughput and different task sizes. Team x had a throughput of eighteen tasks after the first quarter, twenty after the second, fifteen after the third and twelve after the last quarter and they used Scrum the first two quarters and Kanban the last two as illustrated by table 1.1. It will look like team x benefits most from Scrum. But if the task during the Kanban time was twice the size of Scrum, Kanban would suite team x the best. In order to to get valid result from throughput measurement the size of tasks has to be agreed.

Quarter	Throughput	Framework
1	18	Scrum
2	20	Scrum
3	15	Kanban
4	12	Kanban

Table 1.1: Throughput

1.8 Chrun

1.9 WIP

”WIP-limits seem to be the worst understood part of the Kanban system. When used properly, it exposes bottlenecks and reduces lead time for individual work items. Used improperly, it can starve developers for work or result in too many people working on the same work items.” (Shinkle, 2009)

WIP-limit is a tool in Kanban to reduce overhead by limit task-switching for each developer and visualize bottlenecks. One of the best way to explain WIP and the impact of WIP-limit is to use cars and roads as analogy. All roads have it maximum capacity width number of cars . When this limit is reached, traffic jam occurs and the throughput of cars decreases and lead time increases. The same can be said about software development teams, a software team has a maximum number of tasks they can perform, if the team is pushed over the maximum limit, the throughput of tasks decreases and lead time increases.

WIP-limit helps team to reduce overhead, decrease lead time and increase throughput (Mark L. Spearman and J.Hopp, 1990). But in order to get the best throughput and lead time, a software team needs to experiment with WIP in order to find their WIP-limit. When first implementing Kanban, Shinkle explains that the users don’t care about the WIP, but rather the visibility of Kanban through the Kanban board. When the user get more experience with Kanban, they start to attempt the principles of WIP-limit (Shinkle, 2009).

Shinkle defined novice and more experience kanban-users with a descriptive analogy (ibid.): ”Think about a typical person wanting to bake a cake. They go to the store, purchase a boxed cake mix, and follow the directions as described on the back of the box. They have little to no knowledge about how to alter the recipe nor do they have a desire to do so. Their goal is simply to bake a cake.”

"An advanced beginner understands how to apply some context to the instructions or rules on the back of the box. They can make minor adjustments for things like altitude, pan size, oven conditions, etc. They are still following the basic recipe, but can make minor adjustments likely based on previous experiences." According to Shinkle, principles like WIP-limit will be adopted when the user has some experience with Kanban.

Setting the WIP-limit has a number of benefits according to Hopp, Spearman and Suri. They stated that it reduces flow times, reduces variation and improves quality (Mark L. Spearman and J.Hopp, 1990). However Srinivasan, Ebbing and Swearing said that setting the WIP is not easy. They suggest that WIP are just set, and then observe throughput, and adjust after that (Srinivasan, Ebbing and Swearingen, 2003). The principle of Kanban also says you should limit WIP. So what should the limit be? The principles of Kanban tells us to experiment (Kniberg, 2010). Lean Software Management and the Impact of Kanban on Software Project Work suggest that WIP should be minimized as well. The study suggest to minimize WIP-limit to keep high quality (Ikonen et al., 2011) and to create continuous flow and bring problems to the surface (Middleton and Joyce, 2012). The conclusion of present study is to keep the WIP-limit low and experiment by slowing increase the WIP-limit until the throughput decreased and lead time increased, then you know that the previous WIP-limit was the perfect one.

Several people have their opinion on how to best measure the WIP, but most of them agree upon that's there is no clear rule of how to measure WIP in software development. Lukasz proposes to use The Effectiveness as an indicator measured at the end of each cycle. The effectiveness is a formula that takes the number of bugs found (ai) and the number of bugs found by external people (e.g. lawyers, accountants, coaches, consultants, translators, internal and external service providers ,etc.)(ei), and minus ai and ei, then divide the result by ai and multiply it by 100% as shown in 1.1 (Sienkiewicz, 2012)

$$Ei = \frac{ai - ei}{ai} * 100\% \tag{1.1}$$

1.9.0.2 Benefits with WIP-limit according to research

1. The tasks should be prioritized from high to low, in order to keep the best WIP. The article also states that high WIP will keep people task switching and not be able to fully concentrate on each work in progress (Ikonen et al., 2011).

2. There's stated when using short-cycle times and Kanban board to limit WIP, the software development team's learning is increased'

(Middleton and Joyce, 2012)

3. The team in Lean Software Management study realized the where bottlenecks. So the team started to determine WIP by their constraints. The team quickly realized that they had fewer quality assurance/testing staff and business analysts than software developers. This reflected the bottlenecks and the constraints, so the team adjusted WIP to how much work they could handle; this gave the team more experience in dealing with WIP and increased productivity (ibid.).

As notice, there's no clear rule on how to deal with WIP and how to determine WIP, although WIP is a crucial tool in order to Kanban sufficient

1.10 Limit WIP vs. Unlimited WIP

Simulation of software maintenance process, with and without a work- in-process limit did a research on how the throughput and how developers experience WIP limit and unlimited WIP(Concas et al., 2013).

One of the result from this paper was at the end of a simulation, the average of closed issued was 4145 when the WIP was limited and 3853 when the limit was not limited (about 7% less). The paper concludes their finds like; developers are more focused on fixing few issues, because the number of issues they can work on is limited. Because the limit of WIP, the developers are more likely to continue on the issue from the day before, rather than starting on another issue, this reduce overhead. When developers start on a new issue, they need to use time to familiarize themselves with the code and the issue. That could create unnecessary overhead if some developer already has done it, but that developer is now working on a another issue. The study also showed that limit WIP can improve throughput and work ef?ciency. (ibid.).

1.11 Software Innovation(SI)

Software Innovation is a Scandinavian software company. SI develops and delivers market-leading Enterprise Content Management applications that helps organizations improve and increase efficiency in document management, case handling and technical document control.

Software Innovation has approximately 300 employees, and has offices in Oslo, Copenhagen and Stockholm and a development center in Bangalore (*Software Innovation* 2013).

Part II

The project

Chapter 2

Research Questions

In this thesis the overall research question in this thesis is to study the effects of WIP limits in particular for SI. Some of the goals are:

- See if the exist an optimal WIP-limit for a given context.
- How to best find the optimal WIP-limit
- Which various parameters taking in to account in order to optimize WIP.

Chapter 3

Research Methods

In this section information about how my program will measure WIP, Items in backlog and throughput per date, per month, per quarter and per year will be explained. First off, there will be describe how the program do the measure in words, then the algorithm will be explained using algorithm in Pseudocode (JD, 2013)

The set from SI only contains dates when tasks was created, added to backlog, pulled from backlog or finished. In order to measure average per month, quarter and year the program needs to know WIP and items in backlog for each day. In order to do so, the program will generate the remaining days.

Table 3.1 show an excerpt from the dataset, as the reader may notice, the date 2008-10-09 is not in the set, so in this case, the program has to create 2008-10-09 and add it to the set

Created Date
2008-10-07
2008-10-07
2008-10-07
2008-10-08
2008-10-08
2008-10-10

Table 3.1: Excerpt from the dataset

The Date standard is specified as yyyy-mm-dd.
When I write iterate it means looping through the data with a for each-loop.
Quarter of a year is defined as January, February and March (Q1); April, May and June (Q2); July, August and September.

(Q3); and October, November and December (Q4) according to Investopedia (Investopedia, 2013)

3.1 Analyzed Data

In this thesis the plan is to measure the data in one big step, where I measure all the teams and data in some given interval. I would also measure the data in small steps, when SI used Scrum, when SI was in the transforming phase and when they had changed to Kanban. Also I will measure WIP values for a given day, date, week, month, quarter and year. This will give me data to compare between teams in a specific interval.

I will reuse some data from the paper from Quantifying the Effect of Using Kanban vs. Scrum: A Case Study (Sjøberg, Johnsen and Solberg, 2012) Mainly I will look at churn and lead-time measured in the paper and compare them with WIP and throughput between teams. Hopefully the data measured will give me some sense of what WIP limit does for SI and which factors helping determine if WIP-limit matters in agile software development

Hopefully I would gather or get some other underlying data, so I will be able to compare my work, on the set from SI to the other data.

3.2 Information about the dataset

The set from SI contains almost thirty columns of different data. I will not examine every column in my thesis, but the columns I will use will be given a brief introduction to here.

3.2.1 The columns

Column	Description
Created Date	When a task is put in backlog
Date From	When a given task is taken from the backlog
Date to	When a task is done. Done is defined by SI to be ready for release.
Lead Time	The amount of time elapsed from the date the task was created until the tasks has finished
Process Type	States the process used by the team which contains Kanban or Scrum
Team	States the team who has been working on the task.

Table 3.2: Information about the columns from the SI dataset

3.2.2 Algorithms for calculating WIP, backlog and throughput

Variable	Description	Columns from SI
Wip-Limit per day	The number of items in progress on the given day	Date From and Date To.
Throughput	Number of tasks finished on a given day	Date To
Backlog	Number of items in backlog on a given day	Created Date and Date From
Hashmap	Hash table algorithm works by associating keys and their values in one-to-one mapping and storing them in a hashmap (Adi and Salomo, 2010)	

Table 3.3: Description of variable and which columns from the SI set that is used to measure the variable

3.2.3 WIP-limit per day

In order to describe the algorithm of measuring WIP limit per day, I will do it stepwise. A detailed example on how the algorithm works is listed in the 3.2.7.

Key - Date	Value - Integer
2008-10-07	2
2008-10-08	0
2008-10-09	0
2008-10-10	2
2008-10-11	3
2008-10-12	0
2008-10-13	0
2008-10-14	2
2008-10-15	0
2008-10-16	0
2008-10-17	0
2008-10-18	0
2008-10-19	0

Table 3.4: skrive noe spennede

3.2.4 Step 1: Gather all dates into a Hashmap

First step of WIP measurement is adding every date in the date from column into a hashmap. Hashmaps contains key-value pairs, which will be respectively Date and Integer in this algorithm. The date will contain the 'date from' column and the counter is number of occurrence of the date also known as WIP for that date.

Pseudocode step 1:

```
for date IN date_from_column:
WIP = nr_of_date_ occurrence(date)
Hashmap.put(date, WIP)

nr_of_date_ occurrence(Date date)
FOR d IN date_from_column DO
   if d EQUALS date DO
     nr_of_date_occurrence++
return nr_of_date_occurrence

```

3.2.5 Step 2: Gather the remaining days

Since the 'date from' column do not contain all dates, the algorithm will create the remaining dates. In order to create the remaining dates, the program takes the first date and the last date from the hashmap created in previous (3.2.4). Then the program checks if all the dates between the

first date and the last date are in the hashmap. If the dates are not in the hashmap, the program will put the date into the hashmap.

Pseudocode step 2

```
First_date //points to the first date in the hashmap
Last_date //points to the last date in the the hashmap
Next_date //points to the next date
Next_date = First_date // Next_date assigned before iteration
while Next_date NOT EQUALS Last_Date
  New_date = Next_Date + 1 //Finds the date the next date
  AddToHashMap(New_date)
  Next_date = New_date

addToHashMap(Date d)
  if d NOT CONTAINS IN Hashmap
    Hashmap.put(date, 0)

```

3.2.6 Step 3 Measure WIP

Now that all the dates are in the Hashmap, WIP per day will be measured. Iteration through the Hashmap is necessary. Through the iteration each date and WIP is extracted from the Hashmap and the new WIP is measured based on how many tasks that has been finished, illustrated by the pseudo code.

Pseudocode step 3 The hashmap used in this pseudo code is the hashmap generated in section 3.2.5.

```
lastWIP =  0
CurrentWIP = 0
for date AND WIP IN Hashmap
  CurrentWIP = WIP
  Nr_of_finishedDates = Occurrence_of_date(date)
  WIP_measured = CurrentWIP - Nr_of_finishedDates + lastWIP
  newHashMap.put(date, WIP_measured)
  currentWIP = WIP_measured

Occurrence_of_date(Date date)
  for d in The Date To column
    if date AFTER d DO
      Nr_of_dates_to_decrement++
Return Nr_of_dates_to_decrement

```

3.2.7 Example

Figure 3.1 shows tasks id in the y-axis and dates in the x-axis. The green line indicates the duration of the task. This table indicates how many tasks in progress (WIP). On 2008-10-12, tasks 3, 5 and 6 are in progress, which means WIP is three for 2008-10-12.

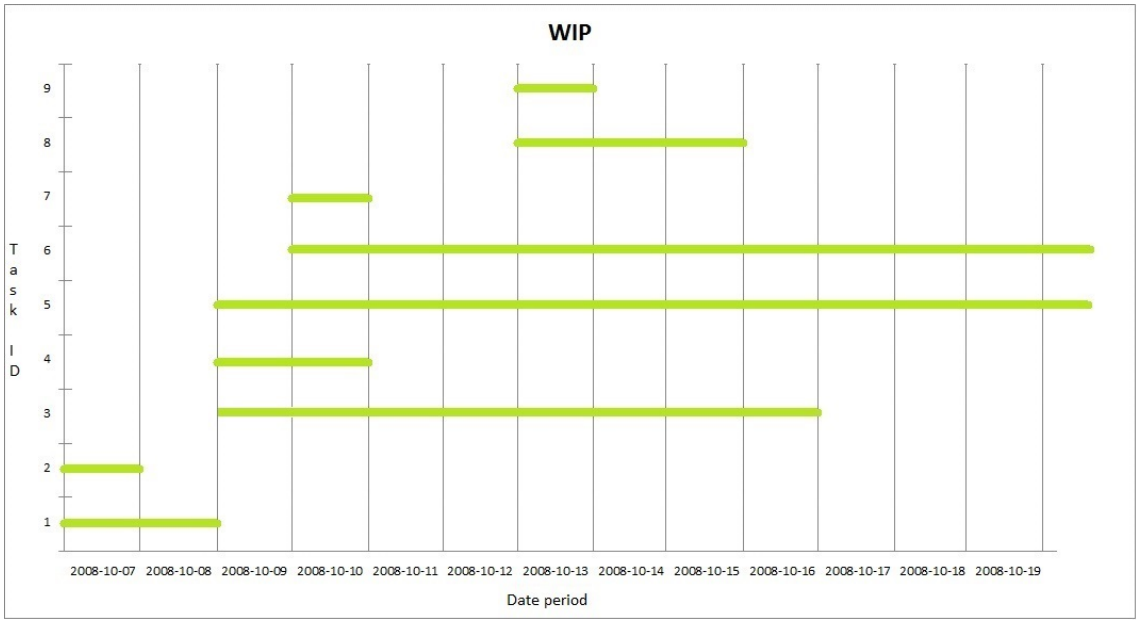

Figure 3.1: Illustrating a WIP timeline

Task ID	Date From	Date To
1	2008-10-07	2008-10-08
2	2008-10-07	2008-10-07
3	2008-10-09	2008-10-16
4	2008-10-09	2008-10-10
5	2008-10-09	2008-11-04
6	2008-10-10	2008-10-05
7	2008-10-10	2008-10-10
8	2008-10-13	2008-10-15
9	2008-10-13	2008-10-15

Table 3.5: Showing Task ID, Date From and Date to

Task ID	Date From	Date To
10	2008-10-07	10
11	2008-10-07	43
12	2008-10-09	65
13	2008-10-09	30
14	2008-10-09	25
15	2008-10-10	85
16	2008-10-10	57
17	2008-10-13	24

Table 3.6: Showing Task ID, Date From and Date to

In the example dates from figure 3.1 and from table 3.5 (skal gel ha med table tre her go) will be used to illustrate how the algorithm measure WIP

3.2.7.1 First step

First the algorithm will do a look-up on the date 2008-10-07 referred with task ID one and two in WIP Table 2 and measure the current WIP counter, which is two, because its two tasks in progress on this date. Then, the program will put the date and the current WIP into a hashmap. Next, the algorithm will see if any task was done in the last period, since task one and two is the two first tasks to be measured in this example, there's no task finished in the period and there's no WIP from other tasks, so the current WIP at 2008-10-07 is two illustrated by figure 3.1.

3.2.7.2 Second step

The program will do a look-up on the date 2008-10-09 referred with task ID three, four and five in table 3.5 and measure the current WIP for this date, which is three. Then, the program will put the date and the current WIP into a hashmap. Next, the program will see that two tasks were done in the period of 2008-10-07 to 2008-10-09. So the program will measure that three new tasks were started at 2008-10-09, two where finished and the WIP from last date measure where two, so the calculation of WIP at 2008-10-09 is 3-2+2, which gives a WIP of three on 2008-10-09 illustrated by figure 3.1.

3.2.7.3 Third step

The program will do a look-up on the date 2008-10-10 referred with task ID six and seven in table 3.5 and measure the current WIP for this date, which is two. Then, the program will put the date and the current WIP into a

hashmap. Next, the program will see that no task where done in the period 2008-10-09 to 2008-10-10 and the currently WIP is three, so this gives a new WIP of 5 illustrated by figure 3.1.

3.2.7.4 Fourth step

The program will do a look-up on the date 2008-10-13 referred with task ID eight and nine in table 3.5 and measure the current WIP for this date, which is two. Then, the program will put the date and the current WIP into a hashmap. Next, the program will look at the period between 2008-10-10 and 2008-10-13, and see that task four and seven was done in this period. The WIP at 2008-10-13 will be $2 - 2 + 5 = 5$ as illustrated by figure 3.1. As illustrated by the example, WIPs are not decrement until the finished date is passed, even though the task is done on the date, it's also worked on the same date, therefore I have chosen to decrement the WIP after each date is passed.

3.2.8 Bug

Each task in the data set is label as either feature or bug as shown in figure 3.7. When the program reads in the data file, each task label as bug are saved in a data structure as shown in listing 3.1.

Task ID	Type
57970	Bug
57971	Bug
57972	Bug
57973	Bug
57974	Feature
57975	Feature
57976	Feature
57977	Feature
57978	Feature

Table 3.7: Example of how tasks are labeled

```
1  void findBug()
2    while inputFile != EOF
3      newLine = readLine()
4      if newLine.Type EQUALS 'Bug'
5        B = New Bug()
6        B.startDate = newLine.startDate
7        B.process = newLine.process
8        B.team = newLine.team
9        AddNewBug(B)
10
```

Listing 3.1: Pseudocode example of how bugs are found

3.2.9 Add Bug

When adding a new bug to the data structure, each bug is gathered into a data structure based on which team the bug belongs to, as the tests in lines 2, 7, 12 and 17 of the listing 3.2 shows. After the right team is found the program tries to add the bug to the corresponding data structure as illustrated in lines 3, 8, 13 and 18. If the date of newly arrived bug already contains in the data structure, a counter representing the date is incremented and the new bug is discarded as shown the method dateExists in lines 26 to 32.

Since the program knows which team the new bug belongs to (after the checks on line 3, 8, 13 and 18) a counter can represent each bug. In our analyze the only important for us is to know how many bugs there are on a given date and which team it belongs to.

```
1  void addBug(Bug b)
2    if b.team EQUALS "360"
3      if dateExists(b.date, 360.dataStructure) EQUALS false
4        // if date don't exists, then add the bug
5        360.dataStructure.add(b)
6
7    if b.team EQUALS "Neon"
8      if dateExists(b.date, Neon.dataStructure) EQUALS false
9        // if date don't exists, then add the bug
10       Neon.dataStructure.add(b)
11
12   if b.team EQUALS "Frontend"
13     if dateExists(b.date, Frontend.dataStructure) EQUALS false
14       // if date don't exists, then add the bug
15       Frontend.dataStructure.add(b)
16
17   if b.team EQUALS "Krypton"
18     if dateExists(b.date, Krypton.dataStructure) EQUALS false
19       // if date don't exists, then add the bug
20       Krypton.dataStructure.add(b)
21
22 void dateExists(Date d, DataStructure structure)
23   for Bug b in structure
24     if b.date EQUALS d
25       b.counter++
26       return true
27
28 return false
29
```

Listing 3.2: Pseudocode example of how bugs are added

3.2.10 Throughput

Finding the throughput per day is quite similar to how bugs are found (described in section 3.2.8). When reading in the data set a new throughput object is created for each line in the set. Then all throughput objects are sorted based on team association. When all throughput objects are sorted, the program measure throughput. The throughput measurement is similar to the dateExists method (lines 26 to 32) in the listing 3.2 stated in section 3.2.9.

The dateExists method starts of with a test, the same test is done for throughput, if the date of the throughput object is in the data structure the corresponding counter is incremented. If the date is not in the data structure, the new throughput object is added to the data structure. An excerpt of the code is listed in 3.3

```
1
2 void dateExists(Date d, DataStructure structure)
3   for Throughput t in structure
4     if t.date EQUALS d
5       t.counter++
6       return true
7
8 return false
```

Listing 3.3: Pseudocode example of how throughput is measured

3.3 Analyze the created data

From the data created i will analyze them using SPSS.

3.3.1 SPSS

IBM SPSS Statistics is a software product for managing data and calculating a wide variety of statistics. SPSS is built around the SPSS programming language.

3.3.2 Analyzed data per quarter

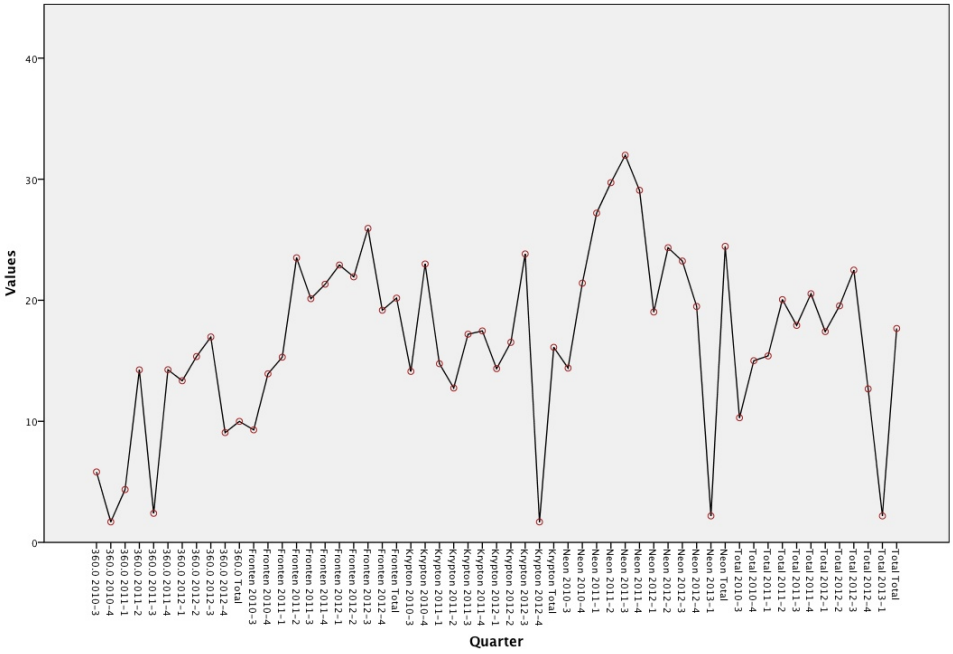

Figure 3.2: WIP for 360 per quarter

Chapter 4

Planning the project

Part III

Conclusion

Chapter 5

Results

Bibliography

Adams, M. and B. Smoak (1990). 'Managing manufacturing improvement using computer integrated manufacturing methods'. In: *Semiconductor Manufacturing Science Symposium, 1990. ISMSS 1990., IEEE/SEMI International*, pp. 9–13. DOI: 10.1109/ISMSS.1990.66111.

Adi, Erwin and Irene Salomo (2010). 'Detect and Sanitise Encoded Cross-Site Scripting and SQL Injection Attack Strings Using a HashMap'. In:

Alliance, Scrum (2012). 'Scrum, A description'. In:

Anderson, David et al. (2011). 'Studying Lean-Kanban Approach Using Software Process Simulation'. In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by Alberto Sillitti et al. Vol. 77. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, pp. 12–26. ISBN: 978-3-642-20676-4. DOI: 10.1007/978-3-642-20677-1_2.

Conboy, Kieran (2009). 'Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development'. In: *Information Systems Research* 20.3, pp. 329–354. DOI: 10.1287/isre.1090.0236.

Concas, Giulio et al. (2013). 'Simulation of software maintenance process, with and without a work-in-process limit'. In: *Journal of Software: Evolution and Process* 25.12, pp. 1225–1248. ISSN: 2047-7481. DOI: 10.1002/smr.1599.

Gupta, Vikram (May 2013). *InfoQ Interviews David J. Anderson at Lean Kanban 2013 Conference*. URL: http://www.infoq.com/articles/David_Anderson_Lean_Kanban_2013_Conference_Intervew (visited on 01/10/2013).

Ikonen, M. et al. (2011). 'On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation'. In: *Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on*, pp. 305–314. DOI: 10.1109/ICECCS.2011.37.

Investopedia (30th Nov. 2013). *Quarter - Q1, Q2, Q3, Q4*. URL: http://www.investopedia.com/terms/q/quarter.asp (visited on 30/11/2013).

JD (18th Dec. 2013). *Pseudocode Standard*. URL: http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html (visited on 18/12/2013).

Kniberg, Henrik (2010). *Kanban and Scrum - making the most of both*. lulu.com. ISBN: 0557138329.

Lai, C.L., W.B. Lee and W.H. Ip (2003). 'A study of system dynamics in just-in-time logistics'. In: 138, pp. 265–269.

Leonardo Campos Rafael Buzon, Eric Fer (Mar. 2013). *Kanban Pioneer: Interview with David J. Anderson*. URL: http://www.infoq.com/articles/David-Anderson-Kanban/ (visited on 30/09/2013).

Mark L. Spearman, David L. Woodruff and Wallace J.Hopp (1990). 'CONWIP: a pull alternative to kanban'. In: 28.5, pp. 879–894.

Middleton, P. and D. Joyce (2012). 'Lean Software Management: BBC Worldwide Case Study'. In: *Engineering Management, IEEE Transactions on* 59.1, pp. 20–32. ISSN: 0018-9391. DOI: 10.1109/TEM.2010.2081675.

Ohno, Taiichi (2001). *Toyota Production System on Compact Disc: Beyond Large-Scale Production*. Productivity Press. ISBN: 1563272679.

Poppendieck, Mary and Tom Poppendieck (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional. ISBN: 0321150783.

– (2006). *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional. ISBN: 0321437381.

– (2009). *Leading Lean Software Development: Results Are not the Point*. Addison-Wesley Professional. ISBN: 0321620704.

Shinkle, C.M. (2009). 'Applying the Dreyfus Model of Skill Acquisition to the Adoption of Kanban Systems at Software Engineering Professionals (SEP)'. In: *Agile Conference, 2009. AGILE '09.* Pp. 186–191. DOI: 10.1109/AGILE.2009.25.

Sienkiewicz, Lukasz (2012). 'Scrumban - the Kanban as an addition to Scrum software development method in a Network Organization'. In:

Sjøberg, D.I.K., A. Johnsen and J. Solberg (2012). 'Quantifying the Effect of Using Kanban versus Scrum: A Case Study'. In: *Software, IEEE* 29.5, pp. 47–53. ISSN: 0740-7459. DOI: 10.1109/MS.2012.110.

Software Innovation (Dec. 2013). URL: http://www.software-innovation.com/EN/COMPANY/pages/default_.aspx (visited on 12/12/2013).

Srinivasan, Mandyam M., Steven J. Ebbing and Alan T. Swearingen (2003). 'Woodward Aircraft Engine Systems Sets Work-in-Process Levels for High-Variety, Low-Volume Products'. In: *Interfaces* 33.4, pp. 61–69. DOI: 10.1287/inte.33.4.61.16377.

Taho Yanga, Hsin-Pin Fub and Kuang-Yi Yanga (2007). 'An evolutionary-simulation approach for the optimization of multi-constant work-in-process strategy'. In: 107, pp. 104–114.