USING NESMA FUNCTION POINT ANALYSIS IN AN AGILE CONTEXT

ROEL VAN RIJSWIJCK
(0512362)

Radboud
Universiteit
Nijmegen

Supervisor: prof. dr. M.C.J.D. van Eekelen

Institute for Computing and Information Sciences
Faculty of Science
Radboud University Nijmegen

August 2013 – Thesis number: 188 IK

ABSTRACT

The software development paradigm shift from waterfall-like methods to Agile development brings us a lot of benefits. The control of scope, estimations and productivity measurements once useful for incremental development, like Function Point Analysis (FPA) should be able to cope with the paradigm shift as well. While some sizing experts attempt to solve it by introducing new sizing methods, FPA still is widely used as the de facto standard for functional size measurement. By combining principles used in the application of COSMIC Full function points, as well as the early sizing and enhancement project sizing as proposed by the Netherlands Software Metrics Association (NESMA), an attempt is made to use FPA in an Agile context. After the principles of the Agile paradigm and functional software sizing are discussed, a new combined approach called NESMA Agile FPA (NAFPA) is presented as a method for project control and estimations in Agile development. This method is applied in a case study to give an indication of its usability in an agile context. This shows the possibility to apply NESMA FPA in Agile environments, but cannot give a definite answer on whether it is wise to do so. Furthermore it will show some problems that are part of the method, and directs to possible solutions that could help in the usage of NAFPA in software development projects.

Thesis title:	Using NESMA Function Point Analysis in an Agile context
Supervisor:	Prof. dr. M.C.J.D. van Eekelen
Second reader:	Prof. dr. ir. Joost Visser
Institute:	Institute for Computing and Information Sciences
Faculty:	Faculty of Science
University:	Radboud University Nijmegen (NL)
Case company:	ISDC, Cluj-Napoca, Romania
External Supervisors:	Ramona Muntean (Measurements and Best Practices) Ovidiu Suta (Quality Assurance Manager)

Not everything that can be counted, counts
Not everything that counts can be counted.

—William Bruce Cameron, 1968[1]

ACKNOWLEDGMENTS

In this document you will find the master thesis I performed at ISDC in Cluj-Napoca, Romania. It was an interesting journey in the world of Function Point Analysis, but also a challenging one that demanded concentration and quite some work. This would not have been possible without the help of some people. In this small personal note I would like to thank them for their great help and the great time I had because of them.

First and foremost there is my main thesis supervisor: Marko, who in our (Skype) meetings, always managed to help me out when I was stuck and get back to my focus. My second reader Joost. Ramona, my main supervisor at ISDC, who became a good friend and made sure that I got everything I needed to fulfill this project and that I had a great time while doing so.

Furthermore at ISDC: first there were Cor, Theo and Anca, who were willing to help me out by offering me a place in their great company. Ovidiu, my second supervisor that helped me with setting up the research . Ionel, who became an amazing FPA mentor and a friend. Carmen, who was a big help in performing the case study and my lunch break buddies Otilia and Attila.

Last but not least: Judit, who supported me along the way and 'moved' me to Romania.

1 It is a common misconception that this quote is originally from Albert Einstein as there are no known records of him ever have written or said this. However it is for sure that this quote was published by William Bruce Cameron in his 1963 text "Informal Sociology: A Casual Introduction to Sociological Thinking" (information obtained from: http://quoteinvestigator.com/2010/05/26/everything-counts-einstein/)

CONTENTS

INTRODUCTION

1 INTRODUCTION

1.1 INTRODUCTION: RECAP OF THE RESEARCH PROJECT

This Master Thesis was started on request of ISDC, a professional near-shoring software development company, that wanted to investigate a uniform size measurement for their developed software. This should be used for estimation and benchmarking purposes. The research was originally proposed to compare the standard in SCRUM, expert judgment with SCRUM poker, with functional sizing methods COSMIC and NESMA FPA. The latter was already used at ISDC. While starting the investigation it became clear that in contradiction to COSMIC and expert judgment with SCRUM poker, NESMA FPA did not provide us with a way to handle Agile projects by default. Therefore the research switched from the main comparison of the three methods to the development of this method. Now that this is explained, the remainder of the document will focus on the research performed in the end. All the activities performed in light of the initial research question will be omitted. In this way the reader is presented with a coherent document. Traces will still be found on COSMIC FPP, due to the fact that principles and techniques proposed specifically for COSMIC FPP are presented as possibilities to overcome difficulties in the NESMA sizing method with regard to Agile environments. COSMIC FPP will be a means to an end and not longer be part of the goal. Later on COSMIC will still be presented as a possible other path to follow, to achieve the same goal. Perhaps in an even better way. Not only will this thesis give an answer to the question on how to use NESMA FPA in an agile context, it will also analyze if it might be worthwhile to use NESMA FPA in this context, and what could be concluded from the information obtained after applying it. Due to the nature of the project we used for the case study, which was sometimes referred to as 'kamikaze' project, a clear comparison between the method and SCRUM poker could not be made. Therefore the thesis shifted again its focus from a strict comparison between the method proposed and Expert Judgment with SCRUM poker to showing the possibility of using functional size in a similar way. Finally, it will also show what additional information can be obtained from using NESMA FPA together with SCRUM.

1.2 MOTIVATION

Estimating software projects is as difficult as it is important. Every client and supplier would like to know what a certain project is going to cost before starting development. Wrong estimations and IT projects getting way over budget are common, making the develop-

ment of methods to improve estimations a crucial research area. This has been topic of research since the 1960s already [22]. Even with all this knowledge on good estimation methods a widely used approach to estimating the effort of developing a software project is still expert judgment. One or more experts are asked to estimate an entire project in hours needed, based on their experience and knowledge with help from certain tools and methods. One of the most commonly used being, the Work Breakdown Structure (WBS) in which the project is divided in small tasks, each estimated seperately.

SCRUM provides an own way of estimating effort. In Scrum poker developers decide how much functionality (story points for user stories) they will finish in a certain amount of time (Sprint) based on the relative sizes of the tasks. This is merely a short-term planning ritual repeated for each sprint. For the entire project and estimate done before development starts, often expert judgment is still the defacto standard.

While the costs of software projects are mostly concentrated on hours worked, the size of the software to be developed could be a good measure for a baseline. This is not surprising. By comparison, when you are planning on building a house, you would like to know how big that house will be before estimating your budget. Furthermore, when a project takes more hours than estimated, a comparison should be made to a baseline to determine if the extra hours are a result of more functionality being delivered, or inefficient development. This is hard to do when the scope is quantified in monetary or hourly terms. However, different measures can be used to define the size of a project. Lines of Code (LoC) are often used to define the size of a software solution. However, LoC differ between programming languages and developer styles, making it hard to compare projects. Besides this, it is also not the ideal way to express efficiency and productivity, as relating the amount of lines of codes to the productivity might negatively influence the programming done in a project.

Function points could be a better measurement to this extent. They provide a method to count functionality as a measure of size. After all, when looking from the perspective of the client, the functionality offered is what matters, not the lines of code to achieve that functionality. In theory this metric should be uniform for all IT projects worldwide, while in practice, the repeatability of the measurements and their theortical validation is disputed, as was pointed out by Meli [21]. Function points are counted on the requirements for the project, which makes them difficult to measure with incomplete requirements. In Agile development incomplete requirementes are a rule rather than an exception. Changes occuring during the process are also hard to take into account while estimating. Furthermore, traditional FPA methods like IFPUG and NESMA are developed to cope with traditional administrative application development, which differs from new applications that are demanded of the industry at the moment. These are products like web portals, business intelligence solutions and mobile applications, for example.

COSMIC Full function points were introduced as a way to overcome these difficulties. They are considered to be wider applicable and easier to count. They are regarded as the second generation in functional software sizing. FPA as proposed by NESMA and IFPUG is still widely accepted as the default way to measure functional size. COSMIC FFP is slowly increasing its userbase. Investigating whether FPA as proposed by NESMA could be used in a way to handle these difficulties, may provide companies with a way to continue functional sizing without changing to COSMIC yet, or at all. COSMIC FFP and FPA are different in nature, which means they can not be used in a company at the same time, at least, if the goal is to compare those differently sized projects.

Using a combination of SCRUM with FPA is not entirely new.

JeffSutherland et al. [28, 13]used velocity in function points developed by man/month to prove that distributed scrum can be successful, they could not use story points to this extent, because these were not comparable across teams. They argue that function points, although not perfect, are still the best measurement to compare software across teams. Jeff Sutherland is one of the originators of SCRUM.

Recently an article appeared on the combination of FPA with Scrum in a dutch software magazine by Onvlee and van Solingen [25]also they argued that there was added value in combining the two approaches. Furthermore COSMIC released a using COSMIC in agile development guide, showing how Scrum was viewed through the eyes of Software sizing experts.

Wide-band Delphi and software sizing estimation in early phases, or incomplete requirements, which will be covered later, do not exclude each other. When estimating you probably end up to a point where discussions are helpful in making choices in assumptions. Experts debating on this and reaching consensus would probably help the estimation process. The added value is that as a result they will reach a conclusions which is quantifiable and easily translatable to effort, given the fact that historical information will give an input estimate for an output. This is the way it should be, since the output is unknown, but the general productivity in the company is known, it is best to estimate the output, not the input. If the input gets bigger than estimated, it is hard to defend that the output did as well, or that it was a matter of lower productivity. If the output gets bigger than estimated, it is easy to defend that this lead to an increase in input. If the output is the same as estimated, it is easy to argue that the effort increased by lower productivity.

These are all reasons why size could be a necessity in development. This thesis will focus on estimations. Not only the estimate done at the beginning, also at points during the development and will focus on accuracy as well as the effort to do the estimations. All the facts stated above show that this is something worth investigating.

To answer this, a new way of performing FPA based upon NESMA is introduced: NESMA Agile FPA (NAFPA). This uses principles introduced by COSMIC and is a natural evolution of the concepts already introduced by NESMA. NAFPA is tested in a case study by

applying it to a software development project done in SCRUM. This will lead to certain measurements, difficulties and opportunities for improvement that will be examined.

1.3 RESEARCH QUESTION

This thesis will therefore answer the following question:

To what extent is using NESMA FPA in Agile environments an advantage compared to using Expert judgment?

To come to a good answer the following research questions should be answered:

- Q1: What are the difficulties in estimating software size with regard to Agile development (with emphasis on SCRUM)?
- Q2: How can NESMA FPA be used to handle these difficulties?
- Q3: When comparing the technique proposed in Q2 with expert judgment, which one provides a company with better estimates?
- Q4: How much effort does it take to use the methods in Q3?
- Q5: What can we learn about the project that we cannot find out by using SCRUM?

2 BACKGROUND

The background of the project will be on Software Development Management and more specifically on the sides of Software Estimation, Software Sizing and therefore Software Metrics. On the other side the context of these measurements will in Agile Development and more specifically SCRUM. The relation between the terms discussed in this thesis are made clear in figure 1, most of these terms will be explained in this chapter. Furthermore we will discuss the cone of uncertainty, one of the basic principles in estimating software projects. This cone shows in which stages of the software development process which deviations can be expected from the actual value when estimating.

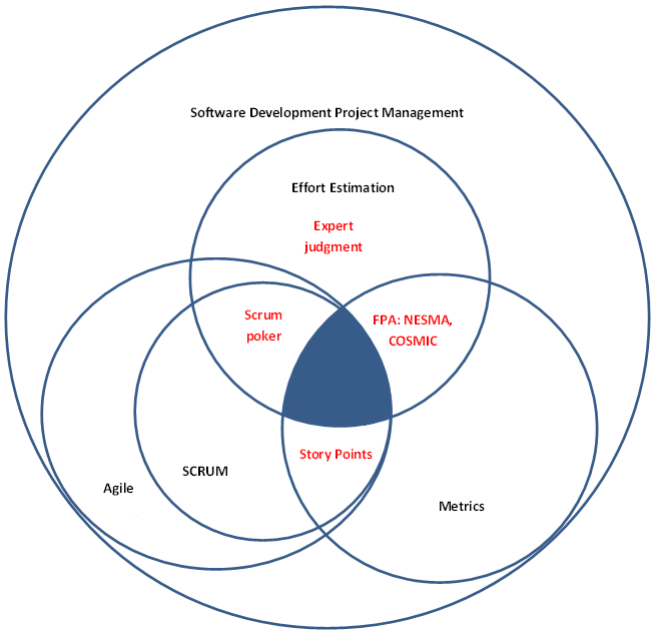

Figure 1: The background of the research as visualized in a combination of a Venn and Euler diagram.

Figure 1 Shows the background of the research, easily visualized as a combination of an Venn and Euler diagram. Most of these terms will be further explained in this section.

2.1 FUNCTION POINT ANALYSIS

To take a software size measure independent of the Lines of Code and the amount of hours, Albrecht thought of Function Points Analysis in 1979 while working at IBM [14].

In the course of time, different methods for Function point analysis evolved, which can be seen in figure 2. All five endnodes of the graph obtained ISO certification.

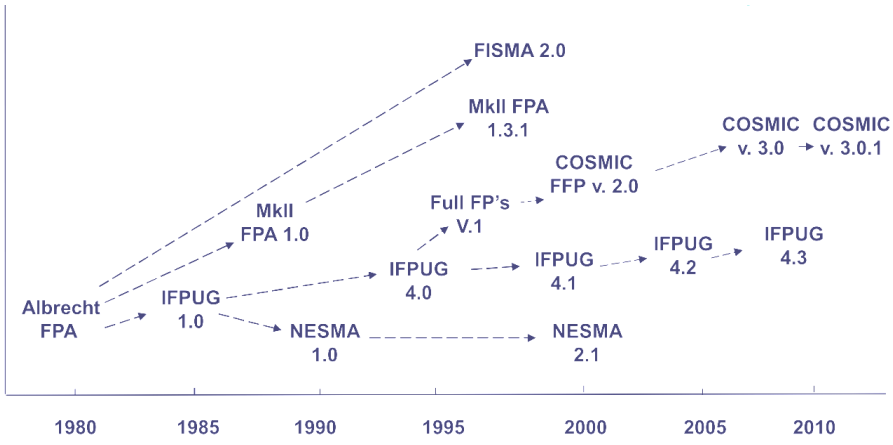

Figure 2: The evolution of FPA methods, over time (Validated and taken from the presentation 'Do We Really Need To Choose One Functional Size Measurement Method?' by Gencel as presented on the *UKSMA / COSMIC International Conference on Software Metrics & Estimating* held on 27-28 october 2011)

It is important to note that the difference between FISMA, IFPUG and NESMA is so small that the values of all three can be easily compared. Furthermore, MKII is lacks widely acceptance. Even its founder, mr. Charles Symons at present time, in later years was leading COSMICON which developed the COSMIC method [1] COSMIC, being relatively new is gaining ground. The data on its growth is limited since it sets itself apart from the other methods by making all documentation freely available. The downloads on the website show an increase. COSMIC adressess shortcomings of the traditional methods in a new way of looking at functional size and is therefore regarded as the second generation of functional size measurement methods. We have chosen one of the three comparable methods, NESMA, based on the experience with this at the company where the case study is taken place, as the basis of our new method.

2.2 ESTIMATION

When estimating a project, the focus is normally on the costs and effort. To determine this, a starting point is needed. This is often the outcome of a project. It is clear that the expected outcome of a project is related to the effort that needs to be done to complete the project. To estimate and measure the outcome a baseline is required. An idea for this baseline is the software size. When looking at some traditional methods, the estimations are done theoretically with the input as baseline, since hours and story points are about the effort needed to complete a task. However it makes more sense to base the project progress on expected output. When taking into account the output as well as the context of the project(variables like team maturity, team co-

1 Information gotten from the COSMICON website http://www.cosmicon.com/presidentChairmanV3.asp . retrieved 19th of August 2013

hesion and number of reusable components) we get to a realistic and argumented estimate of the required input which is required to that goal. Therefore when talking about estimation it is important to go from the goal to a methodical way from the expect result of a project to the input required to get there. One way to do this is the Work-Breakdown Structure. In this method a piece of work is broken down into small tasks that are easier to estimate than the project as a whole. Besides looking at the output and reasoning what could be the input for this, look for analogies is another popular way of estimating. This either done by benchmarking or experience. Although yielding quite ok results, this is outside the scope of this research, partially due to not having enough benchmarking info available in the environment of the case study.

Regardless of the what estimation method is used, one cannot go besides the fact that the longer you are away from finishing a project, the harder it is to estimate how long it will take. To make this explicit, the cone of uncertainty will be used to give some handles on how estimations done during different stages in the development process normally differ from the costs of a project in the end.

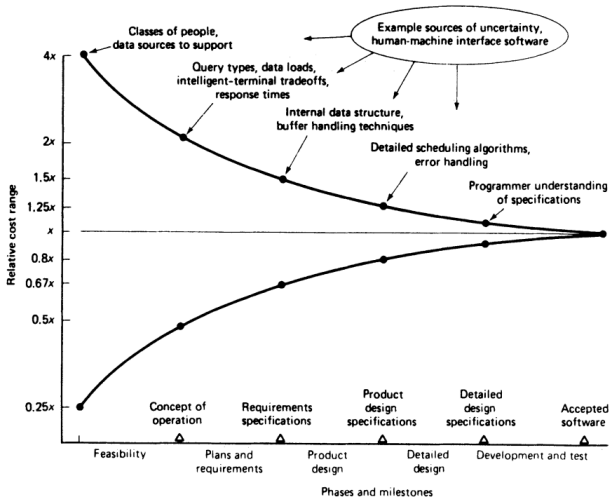

Figure 3: The cone of uncertainty as specified by Boehm.Boehm [3]

2.3 THE CONE OF UNCERTAINTY

During a software development project estimating the total effort it takes to complete the project gets easier with time. Boehm [3] made this explicit in what he called, the *cone of uncertainty* which is the relative graph shown in figure 3. It shows the relation between the estimated value and the actual value in the end on chronological moments during a software development project. At the beginning this can be expected to be around one fourth or four times the actual value.

In this research estimations will be made a different times within this scale. The cone of uncertainty makes sure there is a baseline to compare the measurements at different times to.

As can be seen, estimating the software development project at an extremely early stage can lead to an actual value of the project which is four times higher or one fourth of the original estimate. However practice shows that the first case is more prevalent. Looking at it from another angle, probably overestimations, will not generate as much attention as underestimated projects, or perhaps this information does not even leave the supplying party in the project.

The cone of uncertainty is tailored for the waterfall method,still we can distill some points on there that are similar for Agile Development. The feasability study and planning are for example still done at the beginning of the project, it is the detailed design and the development and test phases that are interative instead of continuous. How to handle this will be further explained in section 15.1 on page 80.

2.4 SOFTWARE SIZING

In section2.2 the fact that one should look at the output of a given project and quanitfiy this without directly looking at the potential input in hours was explained. To this extend function point analysis (FPA) can be used.

Function points analysis is a method to determine a value known as the *software functional size*. In this section a definition of functional size will be given which will serve as a backbone for the rest of the thesis. Furthermore the dinstinction between functional size, complexity and effort is explored.

Functional size is an approach to determine the software size. Software size is a vital component in cost accounting methods like COCOMO as proposed by Boehm et al. [2]. A size of something often serves like a baseline for the costs of things. One could think of the size of a cable, the size of a sheet of metal and the surface size of a piece of real estate. From the beginning when thinking of software size, what comes to mind is line of codes (LOC) or the kilo equivalent (KLOC). This is easy to measure and is understandable to everyone. There is something intrinsically wrong with using this metric as a measurement of the size. Different programming languages can express the same system behaviour in a different amount of lines. This could be solved by making a database which can transpose the average length of programs in different programming languages. One could imagine that Ruby equals the Java Code by a factor 0.6 for example, since Ruby is known for the fact that less typing has to be done compared to JAVA. A worse disadvantage to this respect is the simple fact that the same behaviour can be programmed in fewer or more statements in the same programming language. This also means that a developer who writes more bloated code could get praised for being more productive than his colleague who wrote the same functionality in the same time with less LOC. Furthermore LOC is something that one would not imagine of being of interest to any client or a project manager of a software project. Just as one would not be interested in number of bricks that are used while building a property. These

latter arguments are obviously the main argument that LOCs should not be considered when talking about software size. This argumentation also shows what is in the end important: The output of the software created, which is the functionality it offers to the user. To this extent functional size was created as an alternative to LOC.

Recently also 'business value' is proposed as a measure that could be used as an important leverage for contract negotiations and software projects. This should be considered a shared responsibility between the supplier and the client and could be difficult to make measurable as it is context specific, something which function points are considered to be to a considerably lesser extent.

2.4.1 *Functional size vs. effort*

In this thesis the step will be made from functional size to effort. In the end functional size will be the metric that will give an indication of the effort that should be done to complete a software development project. This means that the size will be converted to effort using some parameters, hence the name: parametric estimation. It is important to note that size is not the same as effort, nor is it directly related. This is a distinction that is sometimes forgottten, since software sizing is sometimes blindly used for estimation purposes. To get from size to error, there are more parameters at play.

Besides this there is another mixup commonly made. That is the distinction between size and complexity. Where complexity tells us something about the inner working or contextual environment of a piece of software of a certain size.

Productivity also plays a part in this, which in its turn has its own dependencies. Productivity can easily be measured in hours per function points and the hours per function points rate can be adjusted to include complexity. In this thesis, no further action will be taken with regard to the complexity, the focus will be on size measurement of functionality. It is important to note the difference and the fact that it exists.

For the case we will abstract from all this and assume in the hours per function point and the complexity and risks associated with the project are taken into account.

2.4.2 *Product size vs. Developed size*

Another important distinction to make when talking about software size is the difference between **product size** and **developed size**. Product size is the size of the piece of software at a given moment, or the estimate of the size in the future. Developed size is the size that was actual developed, so this includes all the changes and deletions in the process. Effort is stronger related to developed size than to product size. If during development the software functionalities are added and later deleted and other functionalities are changed, this will lead to an increase in developed size, but to less extent to an increase in

the software size. Considering a piece of software with the same size with less changes and deletions, you cannot compare the two projects on productivity without looking at the developed size first.

The terms used for the same concepts are different among methods. While COSMIC uses product size vs. developed size. NESMA talks about software size vs. project size. The COSMIC terminology is preferred, because project size could be confusing. It might lead readers to believe that it has everything from requirements analysis, the travel hours for the team to go to the client and the cost of coffee also in it. These are aspects of a project that are not taken into account in the scope of this research.

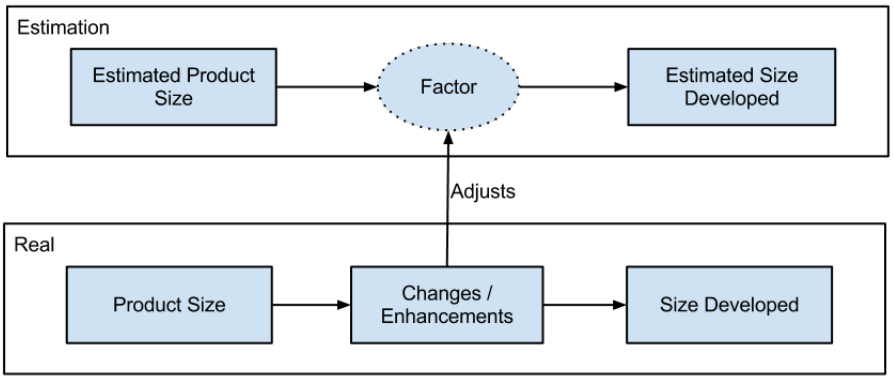

Figure 4: Simplified overview of the correlation between product size and size developed in reality as well as in estimation

In figure 4 the clear difference between the estimated product size, the estimated developed size and the real product and developed size is displayed. Both have a concept in the middle that increases the size developed as opposed to the software size. In reality these are all the changes and enhancements done during the project. In estimation this could simply be a factor, which on its turn can be calculated from different parameters. A continuous improving estimation program for new projects should adjust the factor all the time and come up with parameters that influence this, like client, team, distance. This is in no respect different from an usual risk analysis. This is considered beyond the scope of this research as well, but might prove an important starting point for further research.

2.4.3 *NESMA*

NESMA (Nederlandse Software Metrieken Associatie) is an association that unites the users of function points in the Netherlands. NESMA developed an own method of function point analysis, which is a branch from the IFPUG FPA method. The changes from IFPUG FPA are so subtle that the values obtained for both methods should lead to almost the same values and be comparable.

NESMA functional sizing is done by looking at the user transactions as well as at the data functions of an applications from the user

point of view, where a user can be a person as well as another application, comparable to the notion of 'actor' as it is used in UML.

A function is determined as follows:

> *The five types of components of which an application exists, as seen from the perspective of FPA. These components determine the amount of functionality an application provides to the user.*

These five types of components are divided in two groups:

- Data function types
- Transactional function types

The data functions can have two types:

- Internal logical Files (ILF): A logical group of permanent data seen from the perspective of the user that is used and maintained by the application.
- External Interface Files (EIF): A logical grouped of permanent data seen from the perspective of the user that is used by the application, but maintained by another application.

The user transactions can be of three types that are recognizable by the user :

- External Inputs (EI): A unique function in which data or control information is entered into an application from outside that application.
- External Outputs (EO): a unique output that crosses the application boundary.
- External Inquiries (EQ) :a unique input/output combination in which the application ditributes an output fully determined in size without further data processing, as a result of the input.

The size in function points of these functions is determined by their complexity. Complexity can be 'Low' 'Average' or 'High' and should be determined differently for user transactions as well as data functions

For user transactions the following concepts are used to determine the complexity:

- Data Element Types (DET): the number of attributes that are associated with the user transactions, or are part of the ILF.
- File type referenced (FTR): the number of ILF and EIF that are referenced in the user transaction.

For determining the complexity of the data functions the following components add to their functional complexity:

- Date Element Types (DET): attributes or columns of the ILF or EIF that are used in the application

- Record Types (RET): a user recognizable subgroup of data within an ILF or IEF, or a reference to another ILF / EIF.

Each different function has a different number of functions associated with each different complexity level. These should be counted for all the functions identified and the total will be the functional size in the end. This total function point count is the functional size of the application.

A schematic overview is shown in figure 5:

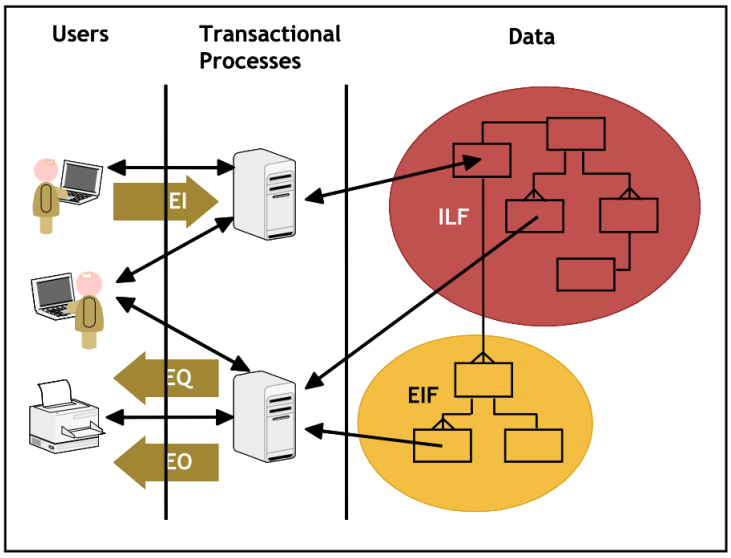

Figure 5: A schematic overview of FPA, taken from Heeringen [20]

Using nesma for incomplete requirements

To use NESMA FPA in the early phase of the software development process, **indicative function point analysis** (FPA_i) and estimated FPA were developed. FPA_i uses indication techniques to estimate the final size of a project. For this there are a number of strategies, depending on the availability of the information. A process model could be used to count transactions and estimate the associated data. Or a data-model might be used to count the data and estimate the transactions.

The other particular approach to early sizing is especially well known **'Estimated FPA'.** In this aproach all functionalities are counted. The transactions are rated as average and the datafunctions as low. In the function points community it is known as the *'Dutch Method'*. These methods will be further explained in the first part of the actual research. In that part we will introduce the concept underlying early sizing gradually with the explanation of how this relates to SCRUM.

We will discuss these two sizing methods here.

The real high level requirements can be sized using FPA_i. In the waterfall method the stage in the project which is suitable for FPA_i is detailed in figure 6.

FPA_i is performed differently depending on the documentation available. The different kinds of information needed for FPA_i are

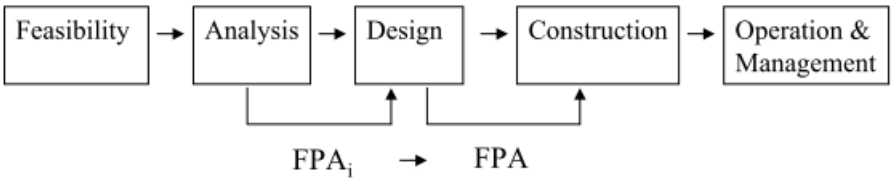

Figure 6: The place of FPA_i in case of waterfall project

defined in the 'FPA in the early phases of the application life cycle' NESMA [23] manual as:

A. User requirements

B. A list of user data groups (objects)

C. A process model

D. Application boundaries

Furthermore NESMA advises a benchmarking list, which because of unavailability in the case, will not be covered, as discussed earlier.

ESTIMATED FPA In the manual for the FPAi method elementary processes are mentioned as a minimum requirement for the use of *estimated FPA* instead of *indicative FPA*. When looking at the definition we see:

> Elementary process: a process is an elementary process when two conditions are met:
> • The process fulfills an independent role as far as the user is concerned and handles the processing of information to completion;
> • The application is left in a consistent state after completion of the process.

These elementary processes when identified are either part of a data functions (EIF or ILF) or part of one of the transaction functions (EI, EO or EQ). After identfying these, the analyst simply gives the average complexity to the transactions and the low complexity to the transactions. It is not needed to look at the DETs FTRs and RETs, making the chance smaller that the estimation is done incorrectly.

INDICATIVE FPA For FPA_i the guide to sizing in early stages offers different approaches depending on the available documentation as described above. One of the requirements is the availability of a process model, which will lead in the end to a situation in which we can use estimated FPA. To complete an incomplete process model a CRUD (Create, Read, Update, Delete) matrix can be constructed from the processes and the data-groups. If necessary, it can be completed with different processes. This is a suitable method to make sure a complete set of user stories will be taken into account. It looks more suitable for the early stages in a waterfall project for which the

requirements should be checked before starting development. Therefore this technique will be abstracted from in this research. We assume that the backlog should be complete enough and if not, that the method should be able to handle this incompleteness, without any additional tooling or methods. The usage of additional tools will be considered as outside of the scope of this research, but be discussed at the end as possible improvements.

Without taking analogy based ideas into account, what remains is the counting of the user requirements. Depending on the granularity this could be done in two ways. First there is the 'Quick Scan' when the requirements are not that detailed.

The functional requirements are sized like T-Shirts by the individual project members (Small, Average, Large). Then using expert judgment estimations (or historical data, or benchmark data) a range should be retrieved of the number of functions for each category. NESMA provides us in the manual with the following overview, given as an example in table 1.

size of requirements	**Number of functions**		
	Minimum	**Expected**	**Maximum**
Low	0	1	2
Average	3	4	5
High	6	15	25

Table 1: Example of requirements size estimation to a rang of number of functions, according to NESMA, these figures are often used in an administrative environment

The size for each functionality might be translated to FP in the end by using a company standard for Function Points.

This estimation is fast but inaccurate. Another approach is the 'detailed analysis' of the early requirements. Using this one would actually reason about the requirements available.

This is done in the following way:

- Make a minimum, expected and maximum possible count for the requirement, based upon the estimated outcome of the description.
- Count the total, this will be a range.

The following example is taken from the manual as well, it clarifies how this could be possible.

User requirement: *Maintain customer data* Will get the following values:

- Minimum: Create customer data in an internal user data
- Expected: Create, update and delete customer data in an internal user data group

- Maximum: Create, update and delete customer data and link contact persons to these customers, using a data group 'Customers and contact persons'.

The user data group 'Contact person' is not included in this count as it is assumed it will be counted elsewhere in the application or project.

It depends on the situation and the information available as well as on the personal preference of the analyst which approach will be taken.

Using NESMA FPA for enhancement projects

Besides early requirements NESMA has developed a method to estimate the effort of enhancement projects. These are projects that build on, or alter an existing piece of software. The way of counting these projects is different, since the final size of the product depends on the previous size of the project and changes to existing functionalities need a different way of counting.

For example, when there is a simple output available for the user in the old product, but this is changed in a very complex output, it is impossible to count this as a simple EO. After all, there is already an output. Pretending there was no output before would not reflect reality as altering is not the same as creating. Previously the output was low complexity, now it is high complexity. So the amount of function point grew. This growth should and could be counted, after all, when estimating or evaluating effort, this effort should be taken into account.

If we take a simple output that displays 10 DETs, but this output after alteration shows 12 DETs. This is a small change. However 10 DETs are different than they were before. This could mean the number of FP stays the same. A developer will spend time on alteration, so counting 0 is not an option.

To estimate such projects *Enhancement Function points* were developed. It is clear that these enhancements make the distinction between *developed size* and *product size* as was discussed in 2.4. The developed size or to be developed size in this respect is the Enhancement Function Point count and the product size (or for an enhancement project, the **increase** in product size) is the normal function point count or at least the difference between the new count and the count of the original software.

The increase in product size can be deducted from the enhancement function point count, but only if a previous count is done on the project. The previous count is considered the Base count (FP_{base}) Enhancement function points are calculated differently than normal function points.

The following rules for enhancement function points are found in the NESMA manual for enhancement projects. For adding, deleting and changing existing functionality:

Adding functions

For adding a function, the following rule applies:

$EFP_{added} = FP_{added}$

Which also means that when a new user story is developed, the EFP method and normal FPA can be used interchangeably. One difficulty is that NESMA in their guide states that the calculated effort for enhancements in hours as compared to normal development might be different. This is interesting, since the EFP method already has a way of making the distinction (counting it separately, with impact factors). Furthermore this will only generate problems from the perspective of the added functionality. After all, the changed and deleted function points will be marked as such. The totally new developed versus the added functionalities will not be distincted. This does not make a lot of sense, since developing new pieces of software should take about the same amount of time. The reason behind this is that normally with enhancement projects the team also takes time to understand what is going on in the rest of the program. Nevertheless in this research this is considered not relevant.

Deleting functions

For deletions the following Enhancement effort is calculated:

$EFP_{deleted} = 0.4FP_{deleted}$

This is quite straightforward and understandable. It does make more sense to actually count a deletion less, because it sometimes is a matter of just deleting the call to a particular function.\

Changing functions

For the changed functionalities the sizing of the developed size is less straightforward.

For the data types first the number of DETs that are added / changed or deleted are counted ($\triangle DET$) Then the number of DETs in the original data function (previous baseline) is taken ($\prec DET$)

$DET_{changed} = \frac{\triangle DET}{\prec DET}$

Then an impact factor is defined by looking at the following table:

$DET_{changed}$	$\leq 1/3$	$\leq 2/3$	≤ 1	> 1
Impact factor ($I_{changed}$)	0.25	0.50	0.75	1

Table 2: Data function impact factor

The same kind of impactfactor is determined for the transactional functions:

$DET_{changed} = \frac{\triangle DET}{\prec DET}$

Now the same is done for the FTRs:

$FTR_{changed} = \frac{\triangle FTR}{\prec FTR}$

With those two percentages the following table can be used to determine the Impact factor:

Impact factor ($I_{changed}$)	$DET_{changed}$		
$FTR_{changed}$	$\leq 2/3$	≤ 1	> 1
$\leq 1/3$	0.25	0.50	0.75
$\leq 2/3$	0.50	0.75	1.00
≤ 1	0.75	1.00	1.25
> 1	1.00	1.25	1.50

Table 3: Transactional function impact factor

When the impact factors are determined, the EFP of changed processes (transactional as well as data) can be easily calculated by:

$EFP_{changed} = FP_{changed} * I_{changed}$

Where $FP_{changed}$ is the size of the transaction function after the change is done.

Total count

The result of the enhancement function point count can be determined in the following way:

$EFP_{total} = \sum EFP_{changed} + \sum EFP_{deleted} + \sum EFP_{added}$

As stated before, it is also possible to determine FP_{new} , the new functional size of the values counted. For this we need some extra information:

$FP_{before-change}$ Which is the size of a changed functionality before the change.

The new size of the software is then determined in the following way:

$FP_{new} = FP_{base} + \sum FP_{added} + \sum FP_{changed} - \sum FP_{before-change} - \sum FP_{deleted}$

The enhancement function points count shows that next to handling requirements in early stages of software development it is possible to size a software project that enhances a previous piece of software.

2.4.4 *COSMIC FPP*

COSMIC Full Function Points (CFFP) is a sizing method that is considered the second generation of Functional Size Measurement. It measures the size of software differently from the NESMA methods. Comparing between the function points counted is still heavily under debate, but since there are fundamental differences between the two methods this is considered not advisable and outside the scope of this thesis. If productivity should be measured across teams, the same sizing method should be used.

COSMIC takes a different approach to the size of a piece of software, not the data is important but only the transactions between the different components of the system. The assumption is that the amount of different data should be proportional to the amount of transactions. We will see here that this is actually the case. In this sec-

tion, the COSMIC FPP will be explained briefly. For a more detailed overview of the method, the COSMIC measurement manual should be used. In this research, the COSMIC method will be not used as a comparison to the NESMA sizing, but principles that were thought of to combine this method with Agile Development will be used to fit the NESMA standard to Agile practices. For this it is important that the method is briefly touched upon in the research background.

Each COSMIC measurement has three different phases that lead to a final count of the functional size. These will be explained one by one.

Measurement strategy phase

In the measurement strategy phase the following things are defined:

- ***The purpose of the measurement***: It is important to know why the count is done and on what. Things like, counting requirements, or counting a readymade application are defined here. This will define the artifacts that will be used to do the count.
- ***The scope of the measurement*****:** This defines the set of user requirements that will be counted. The scope mainly refers to the scope within an individual piece of software, not the different individual pieces of software that should be counted. An important part of the scope is the idenfication of different *layers.* This is basically the division of the piece of software into an hierarchical collection of different components. This is done by decomposing the parts of the software system. The different levels that can be identified are considered *Levels of Decomposition.* Each level of decomposition has 1 or more components in it. The functional user requirements should be all in the same level.
- ***The functional users are identified***: a functional user is defined according to the COSMIC Measurement Manual as *'A type of user that is sender or an intended recipient of data in the functional user requirements of a piece of software'*. With determining the functional users, combined with the functional user requirements, the boundary of the measurement is easily determined.
- ***The level of granularity is determined:*** The level of granularity should be determined. As can be imagined at the beginning of a software project, the level of granularity for the input to the measurement is probably greater than at the end. The level of granularity in this case could be interpreted as a flexible way of the different ways of leveraging between Indicative and estimated Function Point Analysis.

Mapping Phase

In the mapping phase, while respecting the decisions made in the previous phase, the following aspects of the application are determined:

- ***The functional processes are identified:*** The functional processes are all the processes that can be identified on the lowest level
- ***The data groups are identified:*** The data groups go hand in hand with *objects of interest*, which is any entity in the functional user requirements. A data group is a piece of non-redundant data.

Measurement Phase

In the measurement phase the transactions within the functional processes that were defined earlier are counted. This is done by looking at the data movements that occur on data groups.

COSMIC distinguishes four different data movements:

- An Entry (E) to the system where a functional users enters data across the boundary of the application
- An Exit (X) where the system outputs data outside the boundary of the application to a functional user.
- A Read (R) where a data is being read from a data group on persistent storage.
- A Write (W) where a data is being stored to a data group on persistent storage.

The size of the persistent storage is therefore abstracted from and is implied with the number of transactions and the data groups.

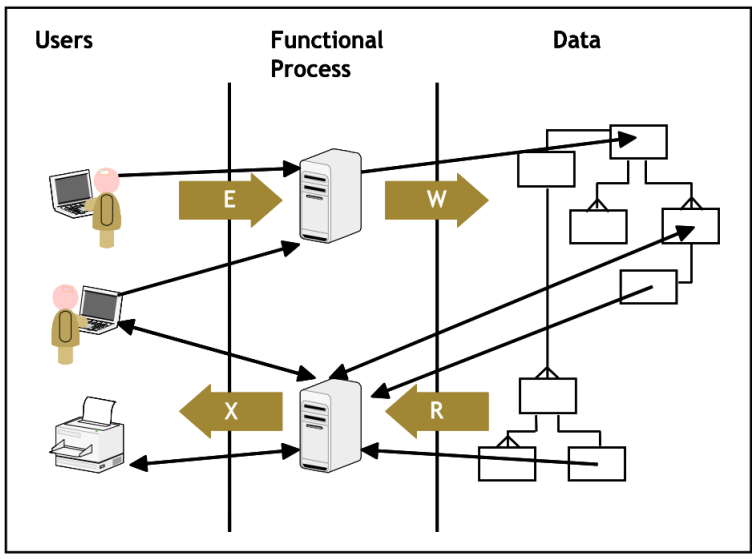

Figure 7: A schematic overview of the COSMIC framework, taken from Heeringen [20]

Each different transaction identified is 1 csu (COSMIC size unit) the total number csu counted is the final size of the transaction.

2.4.5 *Discussions and criticisms regarding FPA and COSMIC*

It would be only fair to discuss criticisms on the methods proposed. COSMIC addresses the critiques that according to COSMICON are disadvantages of NESMA. These are:

- maximum size of functions is limited
- mostly applicable to traditional administrative applications
- not suitable for new software development methods.

COSMIC was developed to handle these difficulties better than normal FPA did. However COSMIC itself is not without criticisms either. Due to its finer granularity, it might be more work to apply it.

In general there is

When looking at the people in the Function Point community we see that often these are professors, or at least PhDs. It is indeed true that conferences organized by the community often have more of a professional goal to it than a scientific one and the methods have a disclaimer that they are not scientifically proven or developed in an academic context, but research has been done to prove applicability in an professional setting.

2.5 EXPERT JUDGMENT

Expert judgment is another way to estimate the total size of a project. An expert is asked to look at the project definition to provide an estimate of the total amount of time and developers this project will take. Assumption is that estimates could easily be made by people with a lot of previous experience. Analogy based estimation will be implicitly made by the expert and disaggregation is often used in the form of a 'Work Breakdown Structure' to divide the entire project into different tasks that can be estimated easier. Even though it seems this method is prone to bias and other human factor problems, it is in fact shown that it often leads to quite acceptable estimates, sometimes even better than analytical methods, as was shown by Brwer et al. [4].

2.6 SCRUM POKER / STORY POINTS

When using SCRUM to aid the Software Development process, the estimates are done by splitting the entire project in user stories.. To estimate this, story points are given to the userstories that will be developed in the sprint. Storypoints are a relative size measure to compare the individual user stories among each other. The principle behind this is that in general it is easier to tell that an object A is twice as big as object B, but to estimate the exact size of object A is more difficult. This makes sure there is no real baseline for the sizes of the individual user stories. Storypoints are incomparable across teams, but conversions to hours can be made, by taken the total amount of hours in a sprint and seeing how many story points were done in that

sprint. When looking at how story points are actual made a strange conclusion can be drawn. Even though people refer to it as a relative size measure, it is actually a relative effort estimation. The relative effort estimation that is given to a particular piece of functionality at the beginning is not revised or verified with any baseline even after development is done.

To establish the storypoints for a userstory, planning poker is played at the start of each iteration (called Sprint). Planning poker is played to determine the total amount of hours needed to do the tasks in the sprint An arbitrary piece of software is chosen. All the planning poker players agree on the fact that this is a piece of software they can identify with. Then the planning poker players all turn a card which has a sequence of numbers on them (often the fibonacci numbers). These numbers indicate how many times the userstory is bigger and/or more complex. No distinction is made between complexity and size and the estimation done has the title 'size-complexity'. Size and complexity are difficult to get a grip on and confusing them leads developers to answer the question: "how many times longer does the userstory take to develop as the 1 storypoint piece of software?" This makes story points a measurement of expected effort, not size nor complexity. Still if the group is agreeing on a certain amount of story points and this way of measuring is more accurate as estimations than the both function point methods, all the disadvantages of story points can be compensated by the advantages they bring.

3 RESEARCH METHOD

To answer the research question proposed in 1.3 the environment in which the research takes place will be taken into account. This is a near-shoring company with different projects that are managed using SCRUM. Examples are taken from this field of practice as well as from the literature available on function point analysis.

The research question will be answered in two parts: The first part in which the method is introduced from the literature and thought of by the authors. The second part in which the method will be tested in the case study and conclusions can be drawn regarding the possibility and the efficiency of such method. Furthermore flaws can be found which will lead us to present some improvements.

3.1 PART 1: INTRODUCING A METHOD FOR NESMA IN AGILE ENVIRONMENTS

In part 1 problems will be identified that arise when using estimation techniques, adhering to Q1. A scientific literature study will be used to this extent as well as examples from practice, either from sources in the blogosphere, whitepapers or the interviews with the stakeholder in the context of the case.

These problems will be investigated on how to solve them with the methods provided by NESMA and the principles more common in methods like COSMIC. or how new research approaches these problems using the methods , like asked in Q2. When taking a selection of the possible solutions to sizing Agile projects with NESMA, a complete approach can be constructed, called NESMA Agile FPA (NAFPA). This can be tested for its applicability in a case study in Part 2.

3.2 PART 2: APPLYING THE METHOD IN AN AGILE ENVIRONMENT

Part2 will answer the third research question by using the approach determined in step 1, to test the following hypotheses:

- H_1: The accuracy of the estimations (independent of what is measured) is higher for NAFPA than for expert judgment .
- H_2: The effort of doing the estimations is lower for NAFPA than it is for expert judgment with Scrum Poker.
- H_3: The accuracy of the estimations in hours is higher for NAFPA than for expert judgment .

These hypotheses will be tested by doing a case study, the context of which will we elaborated upon in the next section.

Measurement will be done for some particular moments in the cone of uncertainty. The measurements will be done by the same analyst, to make sure the actual measurements are taken into account and not the measurerer. The sample will not be big enough to result in conclusive results, but it will be exploratory of the use of a combination of the methods and the difference. This is not mutually exclusive. Estimating time per sprint using story points can have certain benefits, like ownership of the developer of the user story and improvement in team coherence, that function point analysis can not offer. So as was already asserted. They can be used next to each other.

Afterward a qualitative analysis will be done which will answer the following questions:

1. Effort of Estimation How big is the effort to use this method of estimating the project? Effort in this case will be two folded:
    a) Time taken to measure. The time taken to measure the project will be used to give a conclusive estimation of the effort of estimation. Since costs for a software project are mostly based upon hours worked, including the analysts, this will be one of the major concerns.
    b) Ease of measuring : How easy is it to measure the project? Even though the speed of measurement and the ease of measurement probably are correlated. It could be that the task takes little time, but is perceived by the analysts as an uninteresting task to perform.
2. General accuracy : How accurate were the estimations in their prediction?
3. Accuracy in Hours . How accurate was the prediction when looking at the amount of hours? For this the function point will be calculated in hours using the company average for similar projects.
4. What extra information do we get from using NAFPA?

3.3 CASE STUDY

The case study is performed at ISDC, a company specializing in near-shored software development, which has its main office in Hilversum, The Netherlands. The case study was performed in Cluj-Napoca, were most of the software development takes place. Projects are mostly done using SCRUM at ISDC. A beginning is made implementing NESMA FPA in the organizational processes as well. Therefore a FPA team has recently been set-up to start working with function points . For the moment FPA measurements were made for some real projects and some employees were given the task to count function points in projects recently finished as means of practice. Furthermore the first Pre-Sales counts were done, which would establish the selling price. The fact that the case is a company which is moving

towards implementing FPA, makes this research applicable to companies that would like to start with implementing FPA as well. or have a bit of experience with it already. A disadvantage however is the lack of expertise in function points the company currently has. This makes sure the measurements in 'effort of estimation' will not be applicable to companies with years of experience in sizing, this will be probably be less. Furthermore the data available on how many hours per function point will be less representative for this specific case than that data at company well acquainted with FPA. We assume that the accuracy of the size estimates when calculated to hours will improve over time and the results in this case should be considered pessimistic.

The case project was considered to be a critical project that would result in more future projects if done correctly. The specifics of the project will be discussed in section 10.1.

Part I

DEVELOPING NESMA AGILE FPA

In this part of the thesis a method will be presented that will answer the research questions as proposed in Part 1. By looking at principles from COSMIC FPP, SCRUM and NESMA FPA, we will propose a method that could be suitable for controlling the size of a Agile software development project, managed in Scrum while keeping track of the size of the project using the principles presented in NESMA FPA.

4

DIFFICULTIES WHEN ESTIMATING AGILE AND MODERN SOFTWARE DEVELOPMENT PROJECTS

Due to its nature Agile development is harder to estimate than normal waterfall development methods. In this chapter we will map the concepts underlying agile development to the exact difficulties it poses for estimating size and effort. The main problem is the change in focus from the requirements to communication and from a static plan to adapting to change. As efficient as these developments are for the outcome of the project and the relationship with the customer, the harder they make it to do estimations and track project progress in a quantified manner. These changes will be divided in two main issues:

- The fact that the requirements specification from the start could be merely high-level.
- The fact that requirements are bound to change

These will be specified in detail and the difficulties they pose for estimating using FPA are discussed. Besides these difficulties caused by the nature of Agile development, there are also difficulties that are not due to Agile development, but due to the changing industry or general software development. First, there is the handling of non-functional requirements. These obviously have an influence on the project effort, but how to make this concrete? The complexity of the world of software is increasing all the time and all the different situations cannot be covered in this thesis, but some well known ones will be discussed on how to estimate and track their size.

4.1 DIFFICULTIES SPECIFIC TO AGILE DEVELOPMENT

4.1.1 *Only high level requirements*

When looking at the main four principles of the Agile Manifesto Fowler and Highsmith [19]one of them is:

> *Working software* over comprehensive documentation

One of the consequences of this shift in paradigm, is that the requirements of a project in scrum might not yet be fully known from the start. To make estimations based upon information available is harder when not all the details are known. The requirements that are available and their level of detail differ between projects, although the common similarity is that they can be considered "high-level". Which means that they specify globally what the stakeholders expect and perhaps what is the problem that will be solved. When looking further at the requirements the basis of the requirement in Scrum are the user stories, which are a base requirement before starting a sprint.

User stories have the main purpose of conversation starters with the product owner, during development. This is SCRUM in theory; in practice often in a grooming session the user stories are discussed between the team members to make sure everyone understands the goal the user story tries to achieve. The methods for estimating should be able to handle these high-level requirements well, but FPA was developed to score full requirements. The lack of documentation which was a basic part of the waterfall method FPA should be handled in order to work with FPA estimations in this environment.

4.1.2 *Requirement changes*

When looking at the agile manifesto again another one of the four general principles is:

> *Responding to change* over following a plan.

This is a new foundation Agile has brought the software development world and is considered as an ideal. Embracing changes that occur anyway would be a more pragmatic approach. This is not the ideal situation when trying to estimate what will be the total effort of the project in the end, nor to determine its final size. Before the start of each sprint, the backlog is prioritized and updated. Perhaps the product owner would like to have functionality changed, removed or added. These changes are reflected in the software size. The effort of development is also influenced in the same way, but not equally. When a previously developed functionality is removed or changed in a way that decreases the size, the total effort increases, albeit to a lesser extent, because the deletion comes with extra work. The functional size however, decreases. Considering software size, this will bring us to a situation where there should be more emphasis on the **size of the developed software (effort**) and the **size of the software delivered (software size)** as was discussed in the first part of this thesis. When determining some kind of change factor in early estimations we can already give an indication of the size of the developed software. This is based on the functionality of the software to be delivered as well as some parameters to be considered like historical data, client specific data, the outcome of a risk analysis, etc.

Changes during the project pose a lot of interesting questions one could argue about, since changes occur on the border between the client, the supplier and their initial contract. Changes occurring could have different causes, either the client was unclear, the client changed their mind, or on the other hand the supplier was not understanding, the supplier did not provide the requested functionality or simply proposed a solution that did not work. How to handle this in contracts depends on the type of project or company and is outside of the scope of this research. However, for all contracts the functional size of the developed software should be reflected in a good way to help make the changes clear. Bug fixing and rework by testing or other quality checks should be excluded, since these are accounted for by the developing team itself and most probably will be reflected in the

same way during future development. This should be reflected well in the productivity measurement and therefore it cannot be taken as in increase in size. It should be possible to track the growth in functional size to actual development and early estimations, to make clear whether early estimations were based on wrong assumptions from the client or the suppliers side as well. Handling changes in functional sizes as a bonus makes sure that software development could perhaps easier adhere to another agile principle:

> *Customer collaboration* over contract negotiation.

In the sense that the costs for the development will be distributed between client and supplier in a fair and traceable manner. However to evaluate software size as a tool for contractual obligations, would have to be done in another research. This research will not be primarily concerned with this, but acknowledges its potential.

4.2 NON-FUNCTIONAL REQUIREMENTS

Non-Functional requirements (NFRs), are not directly seen as traceable to obvious functionalities in the system. A NFR as *"the system should be available 99% of the time"* might as well be a case of buying the right hardware and setting up a secure environment. Taken them into the software size is debatable, since it is not directly contributing tot the size. Nevertheless, some NFR like robustness, security and usability do contribute to the time taken for a project and the number of development hours. How to handle these NFRs in the estimation process? One way is to handle the NFRs by benchmarking, by adding parameters to the NFRs available and counting some extra effort if applicable. This alternative will not be explored in this thesis.

Another way to go about this is to dispute the fact that NFRs do not lead to functionalities in the end. In a secure environment for example this security will lead to backup functionalities, interactions with other systems, input validation and authentication forms. All of which should be counted as function points. This is true to some extent, but it is not applicable to all non-functional requirements. It also shows that the danger of counting things double arises when explicitly counting NFRs. The latter approach is the one that will be explored and will be used in the method.

4.3 MODERN SOLUTIONS

FPA was aimed at administrative applications developed using the waterfall method. The difficulties that arose when changing software development management to its Agile form are discussed. The fact that IT projects in modern day are more than just administrative applications is not. This will be covered in this chapter. Based on the current state of IT in general and the context in which this research is situated we identify the following types of projects that could make software sizing and estimations more complex than for administrative development:

- Application integration, which is done mostly in the background and therefore has little 'user functionalities'
- Business intelligence, which has the same problem as application integration, as well as the difficulty of having time-consuming calculations in it.
- Web Portal / Mobile development. With web-portal development the problem is that some parts are already delivered by the package implementation. In this research we refer to it as web portal development, but the same can be said for ERP implementations or any other software development which is partly a product implementation.

Because they have common characteristics they will be discussed in this part at the same time. Another paradigm shift which has not been discussed is the movement from software to more standardized 'solution' projects. There create hybrids from the types of projects discussed above. Known technology is installed as package implementation, integrated and perhaps tailored with custom development. This makes sure that all the different approaches to size modern solutions should be able to be used together as well. This creates challenges that should be catered for by the sizing method, making sure that the different solutions are not interfering with the basis of the method and therefore can be combined. We propose that this can only be the case when the solutions are not that different from sizing the applications the methods were actually designed for.

The examples of modern solutions do not cover everything, as the IT industry is highly dynamic. But still it would given an indication of the suitability of the methods to be used company wide, at least in the provided context by the case company.

5 EXPERT JUDGMENT AND SCRUM

The difficulties of using Scrum with regard to sizing, estimations, unclear requirements, changes during the project and the different kinds of development are handled by principles used in the Scrum method itself. How this works will be explained here.

5.1 ESTIMATIONS

Expert judgment does not require the requirements to be fully specified. It is assumed that the expert knows the estimates or retrieves them by using a WBS or talks with the clients. Together with the historical data, which can be stored digitally, on paper as well as in the head of the expert is combines with this to make an estimate of the amount of hours needed to complete the task.

During SCRUM the requirements will be introduced and clarified by communicating by the stakeholder.

EXTRAPOLATION OF VELOCITY After the velocity (how many story points per sprint) is calculated, the remaining user-points can be divided over a N-number of sprints. This will serve as the base of the estimation. This happens in the following way:

First a project is well on the way. Three sprints have been completed. User-stories are available with story points based on a small piece of software. Every story in the product backlog has a certain amount of story points and we estimated a total amount of story points for the rest of the project.

- The velocity in the previous sprints was: 20 story points per sprint.
- We know that we have 400 story points to go
- That will be 20 more 2 week sprints.
- The total amount of time reserved to finish this project is therefore 40 weeks.

When doing a couple of sprints, the velocity will slowly increase. Velocity is determined as the number of hours taken to complete a story-point.

According to a recent article in a Dutch IT industry magazine Onvlee and van Solingen [25]his extrapolation could be done as well with Function Points, however little actual research was found explaining this or proving its convenience. In the case this will be tried arbitrarily, by comparing the estimate of the sprints done to the total project and assuming this will be the case for the other sprints as well.

5.2 ACCOUNTING FOR CHANGE

Experts will probably take into account number of changes requested by the specific client in earlier project, if applicable, or just add some percentages on top of the original estimations making sure the changes are incorporated in the estimation process. This could be prone to bias and estimation errors, because it is basically based on a guesstimate.

In later stages however, the estimation in between, are performed at the start of each sprint. Since sprints are usually started with a discussion with the product owner, with whom the team will discuss all the changes , requirement changes are therefore taking into account. When extrapolating on the velocity, taking into account the changes done in the sprint up until now and calculate this, based on the information available.

Furthermore storypoints are considered a measure of size-complexity. This means that implicitly the participants in a SCRUM poker session will take the changes into account if they see this coming.

5.3 NON-FUNCTIONAL REQUIREMENTS

Expert judgment definitely includes the non functional requirements as well. A highly critical system will be estimated by an expert as taking longer than one that does not require the same high level of defined NFRs. In scrum poker the non-functionals will lead to user stories that will be developed separately from the or will be taken into the scrum anyway. Although this is still topic of debate as well. One approach is to add the non-functional requirements on the user stories as an extra line.

Handling non-functional requirements in Scrum is advised to be done as it is in the actual situation as well. Non-functional requirements normally span the entire software to some or lesser extent. For example the usability constraint could be expressed in a user story in the following way:

> As a library user
> I want to search for books by title
> **With speed and ease of use**
> So that I can find all books with similar titles**.**

Note that this is performance and usability NFR which applies to this user story. Now that we have shown how to handle the NFR in the context of user stories, it is clear that these are handled automatically in the way of estimating as well. This user story will probably receive a higher number of story points than a user story without these extra constraints in wide-band Delphi as the NFR gives the estimator the idea that it will take longer to meet this requirement than one without this extra requirement.

Since the expert judgment is done by either one expert or a group of experts, probably no issues will arise when doing the expert judgment on a given project which is part of the 'modern solutions' discussed earlier. Scrum teams are normally composed of people having previous experience in the field , making sure that the new demands in software are covered for scrum poker as well. This flexibility of both methods is obviously one of their strongest advantages. Therefore, except for the shortcoming discussed earlier, expert judgment with scrum-poker should be able to handle all difficulties illustrated above. The way this is done is through its extreme flexibility, which is created by the expert software developers.

6 USING FPA COMBINED WITH SCRUM

When looking at the application of sizing methods and the Scrum project methodology we see that the COSMIC method has come up with ways to handle the difficulties proposed in chapter 4. Although NESMA did not come up with potential solutions, we can use the strategies proposed by Rule [26] and COSMIC [18] to improve the way NESMA can handle this. When looking at the principles COSMIC proposed, we see that some of these principles, could be used for counting in FPA as well. These principles will be introduced here and combined with the NESMA method. This will gradually show that also NESMA can be applied in Agile environments.

First we will see that NESMA FPA can be used to estimate the final size when not all the requirements are known. The size can always be estimated based on the information available. Then it will be made clear that it is also possible to estimate the size of user stories using known principles. This results in separate parts of the application that can be sized using distinctive methods, based on the detail of information known about these parts. We will see that using principles proposed by Rule [26] these could be sorted n a clear and convenient way. This will provide us with a technique to handle the different levels of granularity in requirements and to keep track of the size using the backlog.

After showing this, the changes will be handled. It will become clear that also changes and the discrepancy between work done and work delivered could be handled using principles from NESMA. It will be shown that the combination of early sizing, functional sorting and EFPA will provide analysts with the right tools to keep track of size without losing connection to the Agile principles.

Then, a method will be proposed to help the analyst with sizing the non-functional requirements. Finally the modern solutions will be discussed.

6.1 THE BEGINNING OF A PROJECT: REQUIREMENTS BEFORE THE START.

Although user stories tend to be very useful at the beginning of a sprint, obviously there will be some other sorts of high level requirements that are even available before creating user stories to put the requirements in. This is normally in the part of the project which is referred to as *Pre-Sales* or *Sprint 0*. These could be the feasibility study of the project, the process diagrams provided by the stakeholders or descriptions on the goal of the project. These requirements could even be from a coarser granularity than is required for FPA and therefore require a different way of sizing. As was introduced in 2.4.3 NESMA provides us with different way of estimating software size in a spe-

cial manual for early sizing. In that manual, methods are provided to count function points in different levels of detail:

- Detailed FPA: Which can be done when the requirements are detailed enough to be interpreted for the NESMA method, this is the default method of NESMA FPA.
- Estimated FPA: Which is applicable when the requirements are generally known, but not in enough detail to determine the complexity of the functions.
- Indicative FPA (FPA_i): as discussed in the previous chapters, which could be used to make a rough estimation of the future size.

When the requirements go from being high level to detailed enough for detailed FPA, the corresponding method can be used to get an idea of the final size.

First we will explain how to size requirements on the level of user stories. Next, we will show how to handle different requirements.

6.2 START OF A SPRINT: SIZING USER STORIES

User stories are created for the purpose of acting as a trigger to conversation with the product owner. This means among other things that a user story when perceived as a requirements can only give a high level overview of what the system should do, the details of the functionality after all, still need to be discussed. Nevertheless, COSMIC shows that it is possible to estimate an almost detailed function point count to a user story.

A user story should have the following structure, according to Cohn [7]:

> As a **[stakeholder role]**
> I want to **[perform an action]**
> **[With some frequency and / or quality characteristics]**
> So that... **[description of value or benefit achieved]**

Grant rule Rule [26] suggests to redefine this in COSMIC terms to make it compatible with COSMIC to:

> As a **[functional user]**
> I want to **[respond to an event]**
> **[With some frequency and / or quality characteristics]**
> So that... **[useful output, or outcome, produced]**

This is merely a subtle difference that might be confusing to users of the scrum method and does not contribute to the COSMIC method to great extent. Therefore it will not be incorporated here. It does show an interesting correspondence with the COMIC method and it shows that a user story could be used to estimate the final size of the user story. This can be illustrated with example shown in figure 8

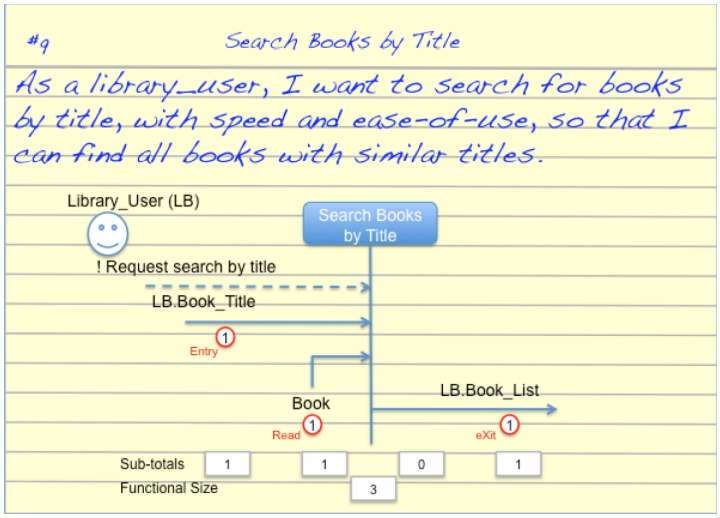

Figure 8: An example of a user story and the corresponding COSMIC FFP count , taken from Rule [26]

which shows us how the user story shown earlier can be counted as 3 COSMIC size units (CSU).

This is a simple example and in practice not every user story is that easy. In this example the user story is equal to only one functional process. When taking the rules of user stories by Cohn [7] into account we might even argue that a user story should be only one functional process. This however is an utopian view on the agile process and therefore risky as an assumption. In reality in projects user stories might not be so strict in their granularity. A user story consisting of more than one functional processes turns up in professional and academic literature. inDesharnais et al. [8] for example. Therefore it is assumed that this will in practice be the case on numerous occasions. This is a simple matter of repeating the same exercise as in figure 8 more than one time. Given all this, we should aim to do the same using NESMA. As in both cases they are functional size measurements, they should be able so give a measurement on the same level. Since NESMA looks at different aspects of the application and for example, takes the data separately, we should make some assumptions to get an indication of the function points:

A. Library user is an authenticated user and therefore has data associated with it

B. The books data-group is not developed explicitly to satisfy this user story and is already available (and is therefore developed in another user story)

C. The books data-group as well as the user data-group are internal logical files in the program (ILF) and not an external logical files (ELF)

D. The list of books has maximum 5 values associated with it.

With these assumptions in mind, the user story is a prime example of an External Output. Since it references 2 ILF we get the following

table. One might imagine that most of these assumptions should normally be known to some extent beforehand, by looking at the entire project.

Type	FTR	DET	Complexity	Function Points
EO	2	<=5	Low	4

Table 4: NESMA detailed count with assumptions of the example user story (figure 8)

Since the user story only references 2 ILF and because of assumption D The user story is 4 story points.

The fact that these assumptions can be made all the time correctly in SCRUM is unlikely, therefore if possible some will have to be eliminated. This is possible when using estimated FPA.

If we use estimated FPA the count gets easier. For estimated FPA, it is not necessary to know the details of the function in this level.

We identified that the user story is a prime example of an external output. In estimated FPA, all the transactional functions are rated average. So this would be rated at average. Therefore the number of function points for this part of the software is 5, as can be seen from the following table:

Type	Complexity	Function Points
EO	Average	5

Table 5: NESMA estimated count of the example user story (figure 8)

Just as a user story can have different functional processes, a user story might have different elementary processes. But next to the transaction elementary processes (EO, EI, EQ) the data has to be counted as well.

This could be approached in the following ways:

- The creation/modification of the Data is put in separate user stories
- The data has already been created in another project.
- The data will be created inside in that user story, leading to a combination of data and user interface changes in one user story, which is fine, as long as the same data is not partially made in another user story.

If the latter is the case, a Work Breakdown Structure is often used to make a distinction between the two. In this case it is easy to size the user story. If the split is not so clear , this sizing will be difficult. In that case it is wise to enforce the splitting of data and functionality based tasks in the WBS. This means that even before an actual count is made in FPA, the way of working in planning and starting up this project should already take the counting guidelines implicitly into account. While this could be considered a disadvantage, splitting data

oriented tasks from transaction oriented tasks, similar to methods like Model View Controller (MVC) and other approaches, might be needed in any case.

When looking at the book example from figure 8 again, given that idea, the userstory will be valued like this:

- Minimum: The library user wants to simply get a list of book titles whenever he types in a book title.
- Expected: The library user wants to have basic information with the book.
- Maximum: The library user wants to get a list of books with a lot of detailed information regarding the books furthermore mistakes should be corrected by suggestion and possible book titles should be given when the user gives the first letter of the title.

The difference here is that not the amount of functions are counted, but the size of the separate functions are estimated. Originally this should be expressed in a range.

6.2.1 *The other way around: assessing user story quality using FPA:*

Now that the possibility to extract FPA counts from user stories is explained. This could also be used in the reverse direction. If it is not possible to deduct a count from the user story, it could mean that the user story is either not clear, or on a wrong level of granularity. This is explained in Desharnais et al. [8] in which it several levels of possible sizing are identified that can be used to assess the right level of detail of an user story to put it into a sprint. Only when detailed sizing is possible a requirement is good to be used for development and testing purposes. Note that this idea could be contrary to the principle of requirements in Agile, where they serve merely the purpose of communicating with a stakeholder. It might be good practice to establish the FPA detailed level as a baseline just before starting development. This is normally done in the SCRUM practice called *grooming*. Where among other activities, the stakeholder together with the team go through each requirement to make sure everyone involved understands the goal and what should be done.

In case a user story is considered not detailed enough, some different actions could be taken:

A. The user story will be split.

B. The user story will be clarified in this grooming session.

C. The user story will be placed in a group of user stories that have not been clarified yet. For this see the next section about the sorting of user stories in a product backlog.

This is outside the scope of this research but it is worth mentioning.

We have shown that estimated sizing is possible even if the requirements are not fully detailed. As one can imagine the requirements in an Agile project will by definition not have a guarantee that the requirements have the same level of granularity at the same moment. This asks for a combined approach to make the estimations. In their manual for handling software size in Agile Development COSMIC [18], provides us with a way to handle this combination. Simply put, user stories are sorted in three sections, called functional areas, based on the knowledge available:

A. High detailed user stories, where the measurement of Function points to be developed could be done in detail.

B. Less detailed user stories, where the number of Function points can be estimated as an average size

C. Low detailed (not even user stories), where the size can be estimated by analogy, for analogy however a lot of historical data is required, which in the context of this research is not available. Another high level sizing method should be used, while analogy will be a viable option for the future.

COSMIC does not take user stories into account. The COSMIC concept is centralized around functional processes. According to the consortium these should replace user stories. They serve the same purpose: the division of the project in manageable chunks. With this approach a product backlog could be created as seen in figure 9.

Functional Area	Functional Process	Size (CFP)	Priority/ iteration	Done
F.A 1	FP1	8	1	
	FP2	10	1	
High detail, Function size measured!	FP3	8	1	
	FP4	6	1	
	FP5	4	2	
	FP 6	4	2	
	FP 7	8	2	
	FP8	8	2	
	FP9	8	2	
	FP10	6	3	
	FP11	10	3	
F.A. 2	FP12	8	3	
	FP13	8	3	
Less detail estimated the number of fp with an average size	FP14	8	4	
	FP15	8	4	
	FP16	8	4	
	FP17	8	4	
	FP18	8	5	
F.A. 3				
		120	5-8	
Low detail, Size estimated by analogy				

Figure 9: A product backlog dividing functional processes in functional areas and estimating them. As taken from COSMIC [18]

Since ruling out user stories and their story points is too radical and could compromise the agile process as it is supposed to be, the functional processes should be created from the user stories, and be used in co-occurrence with them, not replace them. Therefore in table

10 a more suitable representation of this backlog was created by the authors which combines the use of user stories with the concept of functional processes and COSMIC sizes in Csu.. Furthermore, story points can still be constructed from the user stories and the rest of the project remains unchanged. Certainly at the beginning function points should be used as a tool for quality assurance and project control, which is best done without putting impediments on the people and their current way of working.

Functional Area	User Story	Functional Process	Size (CFP)	Priority/iteration	Done
F.A 1	US1	FP1	8	1	
		FP2	10	1	
High detail,	US2	FP3	8	1	
Function size	US3	FP4	6	1	
measured!	US4	FP5	4	2	
		FP 6	4	2	
	US5	FP 7	8	2	
	US6	FP8	8	2	
	US7	FP9	8	2	
	US8	FP10	6	3	
		FP11	10	3	
F.A. 2	US9	FP12	8	3	
	US10	FP13	8	3	
Less detail	US11	FP14	8	4	
estimated size	US12	FP15	8	4	
estimated using		FP16	8	4	
estimated approach	US13	FP17	8	4	
		FP18	8	5	
F.A. 3					
			120	5-8	
Low detail, Size					
estimated using					
indicative approach					

Figure 10: Example of a product backlog with user stories estimated in COSMIC FFP, as adapted from figure 9 by the authors

When looking at this backlog it is obvious that the three functional areas correspond to different methods of FFP analysis. While the user stories in F.A. 1 one can be sized in full detail, the user stories in functional area 2 should be estimated. In functional area 3, the remaining part of the software to be developed is not even put in user stories yet. An indicative method should be used instead of analogy. Conveniently using these three levels of detail the same approach can be used as is explained in 2.4.3. Where the three different ways to size requirements in early stages using NESMA are presented. The same product backlog could be constructed to make the count for NESMA easier. When looking at the projects done in SCRUM, the user stories in the backlog in F.A. 2 meet at least the requirements to perform estimated function points analysis. For F.A. 3 the indicative method could be used to obtain at least some indication on what is to come. In this way it is easy keep track of the growth of the application and the difference in change needed in the database and needed in the end application.

Now we can establish that to some extent NESMA can handle the same concepts as COSMIC, albeit in a perhaps more cumbersome way. Therefore a product backlog alike overview could be created. Here

the division in data and transaction development should be handled. To this extent two approached can be taken:

- To take them separately, so creating a product backlog for each
- To take them together and make the distinction in the backlog itself.

This choice is open for the detailed count as well as for the estimated count. For the indicative count, this choice is not so easy. In the indicative approach, a total estimate is given based on the information available. To make a distinction would complicate the matter, since the size of that data could be derived from the transaction functions or vice versa. Therefore we argue that it is best to keep them together. Separate overviews may be made derived from the data in the product backlog, but in the center there should be a product backlog that has the entire count.

A product backlog could be created which is similar to the one used for COSMIC, as can be seen in figure 11. In this example it is simply shown how a backlog could be constructed. In reality, descriptions and references should be added, tailored towards the specific development project.

Functional Area	Userstory	Function Type	DETs	RETs / FTRs	Complexity	FPs	Priority	Done
F.A 1	US1	EIF	15	6	Average	7	1	
		ILF	5	7	Low	7	1	
High detail,		EO	3	5	Low	4	1	
Function size	US2	EQ	4	6	HIgh	6	1	
measured!	US3	ILF	15	26	High	15	2	
		ILF	20	25	High	15	2	
Functional Area	**Userstory**	**Type**	**DETs**	**RETs / FTRs**	**Complexity**	**FPs**	**Priority**	**Done**
F.A. 2	US4	EIF			Low	5	3	
	US5	ILF			Low	7	3	
Less detail		EO			Average	5	3	
sized	US6	EI			Average	4	4	
using		EI			Average	4	4	
estimated approach		EQ			Average	4	4	
Functional Area	**Type**	**Amount**	**DETs**	**RETs / FTRs**	**Complexity**	**FPs**	**Priority**	**Done**
F.A. 3	ILF	5				175		
	EIF	8				120		
Low detail, Size								
estimated using								
indicative approach								

Figure 11: The setup of a product backlog that could be used for NESMA counting for the transactional functions

In this figure already one of the FPA weaknesses becomes clear. When looking at user story 3 (US3) it is shown that the total size of that user story is 30 FP. With an average of 5 hours per FP, this means 150 hours of work. We will probably see that the task takes less time than planned and is more due to the fact that data is counted more heavily than transactions in NESMA.

The velocity in that sprint, when calculated, becomes high. It could be 3 hours per function point when many data functions were developed. This would give a skewed image of the productivity and is therefore a good reason why the splitting of data and transactional function is problematic. FPA should not be used for micromanagement. The velocity across teams could be evaluated in the

end nonetheless, since the high FP count for the data is compensated by a lower one for the transactional functions. The tracking of the project progress might show some deviations as well for the same reason. The relative big size difference between data and transactions and the fact that data is a prerequisite to transaction development, could make sure that the project might seem as being further ahead than it actually is at the beginning.

6.4 DURING SPRINT: CHANGING REQUIREMENTS

As discussed before, the changing nature of the requirements and functionalities in agile development is one of its main advantages in user satisfaction as well as its disadvantage for making estimations. Even though the idea of changes in software is not new, to treat them during the process as normal instead of treating them as deviations is. Conveniently for treating them afterward, NESMA already supplied its applicants with a method to take changes into account: Enhancement Function Points (EFP) as was discussed in 2.4.3. They refer to the changes as enhancements after development is done. This means that software is delivered to the client, which after using it for some time, comes up with requested changes to the functionality. For 'enhancement' projects like this, the size of the enhancement could be estimated based on the requirements. This is comparable to the way the iterations are handled in Scrum. In fact after each sprint software is supposed to be delivered to the client, which in its turn is assessing it and comes up with possible changes.Therefore we argue that the usage of enhancement function points will provide a good overview of the developed software and quantifies the size of the changes done requested by the client after each sprint.

Enhancement function points will be introduced in the way we see it fit in the method as proposed so far. In this way the incomplete and changing nature of the information available can be tackled at once. The size of the developed software therefore is the EFP, the size of the developed software will be determined from the EFP, so EFP in this case is the leading measurement after sprint 1. The difficulty here is that the data-functions need to be separated from the transactional functions again. This is hard to do in a user story. However in the case situation as well as often in reality, a user story, when discussed with the client, leads to a WBS that will split the changes to the database task and the changes to the functionality task. If this WBS is available, it is easy to count the changes done.

Now the counting can be done similar to a sprint backlog, adhering to the EFP method as well as to the general FPA method. This is shown in figure 12. In this example only the white columns need to be filled in, to determine what happened in that sprint with regard to the size. All the other information is taken from information provided in the product backlog, previous sprint backlogs or the NESMA rules of counting function points.

Sprint 3

					RET / FTR				DET							
Userstory	Function	Add/Chg /Del	Old Type	Chg to Type	Add	Chg	Del		Add	Chg	Del		Function Complexity Rating	New Function Points	Impact Factor	EFP
US3	F7	Change	EO	EO	5	6		15	4		2	14	High	7	1	7
US3	F8	Change	EO	EO		7		20	3			22	High	7	0.5	3.5
US3	F9	Change	EO	EO			1	0	3	7		5	Low	4	1.25	5
US5	F15	Change	EIF	EIF		6		1		4	1	1	Low	5	1	5
US5	F16	Change	EIF	EIF		4		5	6			10	Low	5	1	5
US5	F17	Change	EO	EO			2	8		2		8	High	7	0.25	1.75
US6	F18	Change	EI	EI	4		2	22		4		19	High	6	0.25	1.5
US6	F19	Del	EQ	EQ				0				0		0	0	1.2
US6	F20	Change	ILF	EIF	0	1	0	5	0	0	0	4	Low	5	0.4	2
US7	F21	Add		ILF	6			6	9			9	Low	7	0	7
US7	F22	Add		EO	8			8	10			10	High	7	0	7
US7	F23	Add		EI	12			12	6			6	High	6	0	6
US8	F24	Add		EQ	5			5	8			8	High	6	0	6
US8	F25	Add		EIF	4			4	12			12	Low	5	0	5

Figure 12: An example of a sprint backlog for a sprint that could be used for NESMA counting, only the white fields have to be filled in, to do the count

This could be used as the an important tool in dealing with the client as well.

Sprint 3

FUNCTIONAL SIZE	
Sprint 2 final size	51.0
FP added	31.0
FP changed before	48.0
FP changed after	46.0
FP deleted	3.0
Increase in Functional size	26.0
Current functional size	77.0
EFFORT	
EFP added	31.0
EFP changed	30.8
EFP deleted	1.2
Developed size	63.0

Figure 13: The following small report on what happened in the sprint depicted in 12 can easily be made from the count.

When we combine this information from different sprints,we can make the following overview as well of all the sprints occurred until now. This provides us with an overview which shows us the development of the change factor and the velocity in one easy overview as can be seen in figure 14.

This will give us already some important information, like velocity in FP/Sprint and the ratio between the functional size delivered and the functional size developed, for each sprint. Note that the same might be done for each user story to see the outliers.

Sprint	Increase in functional size	Functional size (Cumulativel)	Developed size (velocity in FP/Sprint)	Developed size (vel.) (Cumulative)	Change factor
1	21	21	21.0	21.0	0.00
2	30	51	45.0	66.0	0.29
3	26	77	63.0	129.0	0.68
4	32	109	40.2	169.2	0.55
5	31	140	42.0	211.2	0.51
Average	28		42.2		

Figure 14: An example of an overview of the progress made in different sprints during development that could be made using NAFPA

When translating this back to the product backlog, an overview can be made that puts the estimated size next to the developed size in the

Functional Area	Userstory	Product size estimated	Product size Done	Developed size	Planned	Done
F.A 1	US1	18	18	18	1	1
	US2	6	6	6	1	1
	US3	18	18	33.5	1	2
	US4	6	6	6	1	2
	US5	15	17	28.8	2	3
	US6	16	11	20.7	2	3
	US7	16	20	5	2	3
	US8	10	11	11	2	3
	US9	5			3	
	US10	4			3	
F.A. 2	US11	14			4	
	US12	12			4	
F.A. 3		80				
Total		220	107	129		

Figure 15: The changes from 12 taken into the product backlog, for the NESMA count

functional areas for each user story, as can been seen in figure 15 This will give the current state of the project in one overview. The user stories that turned out bigger than expected in terms of functionality, or with a much higher developed than delivered size can be easily seen. Ideally one would like to have an information system that links all the information regarding size in a click-through overview. This would make sure that all the information is easily traced back to small parts of the system. For now, it is sufficient to show that his can easily be done.

In figure 15 most of the values in the project can be taken from the vales obtained.

- We see that 107 of the 220 FPs are completed, but this is not the entire truth. We see that that part of the 107 FPs was actually estimated to be 95 FP. So in terms of added functionality we have 15 FPs, this is an increase of about 12.6%. Which could be considered scope creep.
- Furthermore the actual percentage of which the project is done in terms of estimated FP is therefore $\frac{95}{220} = 43.2\%$
- When assuming that the same increase in size will occur, the estimated size on delivery will have 12.6% extra which will be 248 FP
- When losing that assumption the estimated size on delivery will be at increased with 15 FP to 235 FP.
- The ration between product size and developed size (quantification of changes needed to developed to FP) is $\frac{129}{102}$ About 20.5%
- The total effort for this project could therefore be estimated as 248 increased with 20.5% which is 299 Effort FP, of which 129 are done.
- Depending on the velocity in EFP/Sprint, the number of remaining sprints can be easily calculated, therefore the deadline

can be checked on feasibility and measures can be taken to either delay or make sure the deadline is met.

The values obtained in this examples are not taken from a real example, they are merely used to illustrate how the count could be done.

This count can also be done for more sprints than just one, since it might not fit an organization to have people counting function points that often. Fact is that changes are easier to track in a short time-span than in a long one, but this is depending on the resources available. If a detailed change log is kept of all the changes made in the development process, the count can be done taken that log as the main source for the development done.

It is important to have one approach and example approach that will be used in the case. This will be done by introducing three different situations and take a weighted average of this.

The three different situation, or assumptions, are the following:

1. The 'wrong estimate' or 'ahead of schedule' assumption, in which the scope remains unchanged, regardless of what happened in the sprint.

2. The 'incident' assumption in which we assume that the functionality developed more was due to implicit client changes or underestimation of the function points before. We also assume that the other estimates are still correct but the extra developed functionalities are added to the scope.

3. The 'representative for project' assumption in which we assume that there was scope creep and we expect that this will happen as well for the end-functionality. Considering the overall knowledge regarding IT projects, this could be the most realistic one.

Weights can be given to these three different alternatives. It might be wise to emphasize on the third alternative, knowing that extra work and budget overruns are more common than software projects finished earlier.

7 FPA AND NON-FUNCTIONAL REQUIREMENTS

As was discussed in 4.2 and 5.3 Non-Functional Requirements (NFRs) should be added as tests in the user stories. Due to the flexible nature of the agile method, this might not always be the case. NFRs can also be put in separate user stories. Furthermore they are often introduced upfront and should be counted separately in doing estimations.

To handle non-functional requirements in a functional context, the NFR framework was developed. This serves as a base for solutions in software sizing that were developed recently (SNAP for IFPUG and NFSM with COSMIC). NFSM will be briefly discussed to give the solutions a context and then the specific methods are introduced. Unfortunately information on SNAP is not publicly available and is rather new, therefore it is not taken into account as part of this research. That is the main reason that in this part, inspiration will be taken from COSMIC literature again. The NFR framework and its combination with COSMIC will be briefly introduced and then the concepts will be fitted to the way in which to size functionality in NESMA.

7.1 NFR FRAMEWORK

The NFR framework as presented by Chung and Prado Leite [6] is a way to make non-functional requirements more explicit throughout the development process. This framework proposes to go from Non-functionals to sizable chunks by making a divisions in goals (soft-goals) that depend on the satisfaction of the NFR. In the end-nodes of this network, some functional requirements can be discovered. These functional requirements can be sized. When developing the NFSM (Non-functional Size Measurement) method Kassab et al. [11] build on this framework.

The NFR method exploits the idea that a Non-functional requirement like 'Performance' comes with certain functionalities that need to be added to the system. The method is aimed to discover these extra functionalities by means of identifying soft-goals. . There a three different kind of soft-goals:

- NFR soft-goals, which are a high level non-functional description (like 'availability'), which can be decomposed into NFR sub-softgoals
- Operationalizing soft-goals, which can be operationalized (like 'indexing')
- Claim soft-goals, which make a claim that can be checked. (like 'Gold card accounts are important')

From this it is obvious that operationalizing soft-goals can be easily perceived as a sort of functional requirements. They do not necessarily capture functionality visible to the user, but supply us with a requirement that can be measured. The inter-dependencies between the three can be made apparent by using a Soft-goal Interdependency Graph (SIG). An example of a SIG can be seen in figure 16. Claim soft-goals are nothing but some extra information that can be omitted from the sizing. The core of what could be used from the method for sizing is merely a decomposition of NFR soft-goals, to sub-goals to operationalizing soft-goals, in order to use a FSM to increase the measured size in a representative and logical way.

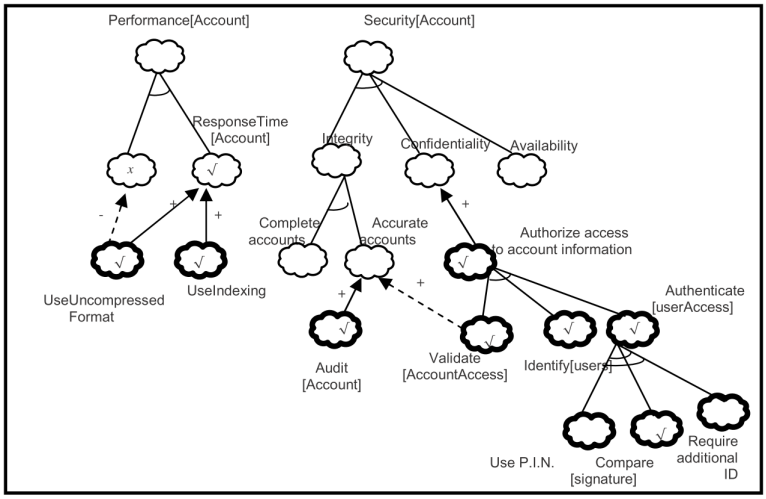

Figure 16: An example of a Soft-goal Interdependency Graph (SIG) as presented in the NFR method. Taken from Kassab et al. [11]

NON-FUNCTIONALS AS A SEPARATE KIND OF REQUIREMENTS Inevitably, non-functional requirements are not just a group of functional requirements, therefore some aspects of non-functional requirements will not be possible to break down to functional requirements. This part of the non-functional requirements are either used to guide architectural choices, or they do not contribute to size, but to complexity. Regardless of this discussion, in this section the way in which NESMA could use this framework will be discussed as at least one possibility to handle this correctly.

7.2 COSMIC NFSM

The operationalizing soft-goals identified by the NFR method could be sized using COSMIC, according to Kassab et al. [11]. Besides the operationalizing soft-goals identified, they argue that each NFR soft-goal could come with two operationalizing soft-goals, because some non-functional requirements need to be controlled. For this one could imagine that at least availability monitoring and availability quantification (measure some variables to obtain a value that represents the

constraint). This logical consequence of having those two control sub-processes, can be seen in figure 17 .

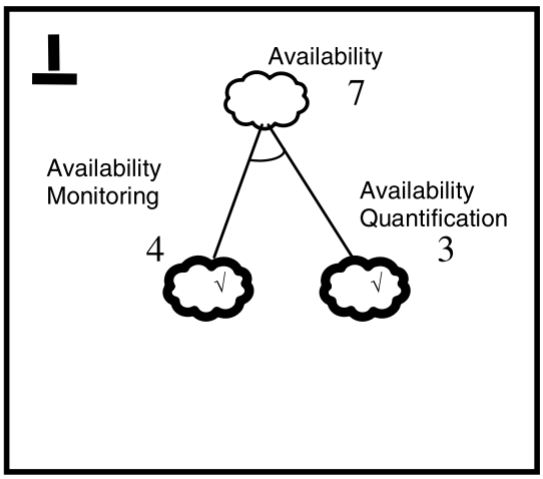

Figure 17: Functional requirements implied by the non-functional requirement 'availability', taken from Kassab et al. [11]

Now that we have obtained the two sub-processes which size can be measured. We see that the non-functional requirements contributes to the total size of the application. The measurement is taken from Kassab et al. [11] and verified by the authors.

When we size every component like this, we can fill in the model in figure 16, in the way as portrayed in figure 18.

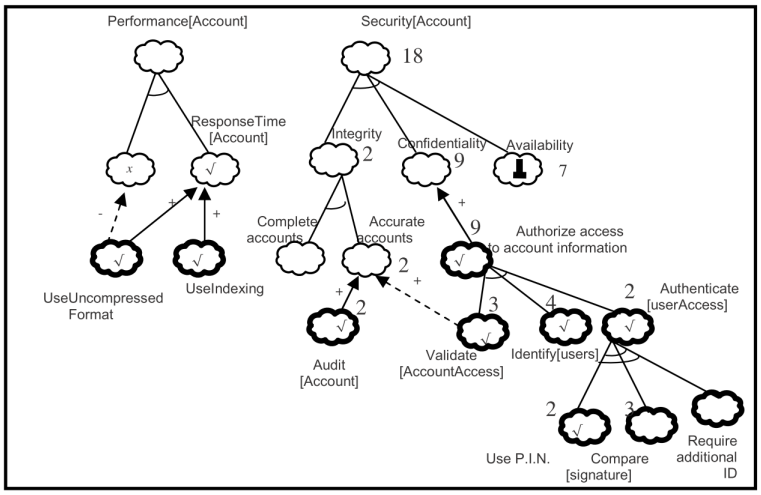

Figure 18: NFR From figure 16 filled in with corresponding CFP

We see that the total size of the NFR for security is 18 CFP, this is an contribution in total project size which should not be overlooked when estimating the effort for the project.

7.3 USING THE SAME PRINCIPLE FOR NESMA

NESMA did not take non-functional requirements into account yet. IFPUG recently developed SNAP, which should be able to make es-

timations more accurate by taking the non-functional requirements into account. The information made publicly and academically available on SNAP is limited. Therefore for the moment it is not possible to discuss this in detail and this research will focus on using the NFR for NESMA as well. When looking at he COSMIC FPP method, the same principles could be used for NESMA, just as it was the case for the high level requirements. So by making use of the NFR framework, some measurable parts of the NFR will be given that we can measure with NESMA. In the example below it will be explained how this is possible.

When looking at the operationalizing soft-goals above we can try to use NESMA FPA to size the operational soft-goals as well. For this we will start from the point where we identified the same processes as before. The only difference would be the sizing on itself. We again have the two processes, discussed earlier:

- 1.1 The availability quantification
- 1.2 Availability monitoring.

Again, we have the difficulty that NESMA method obliges us to consider the data separate from the transactional processes. This makes the sizing a bit more difficult. We assume that the failure history is a small ILF, the target availability level is an EQ. Furthermore the update of the current availability level is EI. The complexity of all these things is low. Therefore we will get the following table:

Process ID	Process description	Function	Description	Data group	Complexity	FP
1.1	Availability quantification	ILF	failure history	-	Low	7
		EI	Write Current Availability Level	Failure history	Low	3
Functional Size in FP:						10
1.2	Availability monitoring	EQ	Read status and send it	Availability status	Low	3
Functional size in FP:						3
Total Functional size in FP:						13

Table 6: size measurement of the availability non-functional requirement

When we size every component like this, we can fill in the model in the SIG as displayed in figure 18, in a similar way.

Since we assume another table needs to be created, which adds heavily to the FPA count in the end, the availability NFR adds 13 FP to the total. Again some assumptions had to be made regarding the counting and especially regarding the data functions. It is unknown whether the failure history is an ILF, or perhaps an EIF. One could imagine that if in the context of this case there are a lot of different systems, a general solution is used to track availability of all systems, therefore the ILF becomes an EIF, making sure the count goes from 13 to 11, if the complexity remains low.

An external solution taking care of availability might have larger needs regarding the data, adding to the complexity. This all depends on choices made by the architect or product owner on how to handle the non-functionals.

Regardless of what is the case, all the assumptions made during the development process should be documented well so the entire project progress in terms of estimated effort vs actual effort is transparent.

8 MODERN DAY SOLUTIONS

When looking at the modern solutions detailed in chapter 4.3, difficulties can be found. After all, FPA was developed with administrative applications in mind. All of the different applications domains discussed in 4.3, have certain characteristics that might make them hard to size. In this chapter some guidelines will be shared on how to cover these particularities in functional sizing.

8.1 APPLICATION INTEGRATION

Application integration is different from default development projects, because it has little interaction with the user. In functional size measurement this can be handled by looking at different applications as different users. A user does not have to be a person, it could also be another application. When integrating application A and application B, both of them could be the user viewpoints, interfacing with the other application. On the other hand, one might choose to count the integration as the application that takes data from all EIFs. EIF being the data in both applications. Both would make sure an accurate count can be done. Counting the functionalities should be done in a way that no functionality is counted double, the analyst should be very careful with application integration in this respect. NESMA does not have a specific guide for this at the moment,There are guidelines for the usage of EIFs and for the usage of transactions to another application. This should be decided upon in the prelude to the actual counting.

8.2 BUSINESS INTELLIGENCE

For business intelligence there is something similar going on, since business intelligence / data-warehousing emphasizes less on user functionality and more on internal combining and interpreting data. NESMA published manuals and articles related to BI and data-warehouses. Since this is available, but will not be used in the case. This will only be covered briefly.

In discussions on forums as well in real life, the fact that business intelligence has difficult calculations are considered to be missing in FPA. However complex calculations are a matter of complexity. This complexity has to do with a different factor that influences the effort needed in the end. Furthermore is the fact that data-warehouse project often have difficult calculations in them that take a lot of time, could be quite general for all data-warehouse projects. This makes sure that the development of a function point could take more hours in a business intelligence project than it does in a project for normal

Activity	Effort in Days / FP
As is	0.4
Configured	0.7
Customized	1.7
New	2.1

Table 7: An example of a table to determine effort from the FP count for different activities

administrative software. This is normal, just as a house, albeit smaller in size could take longer to build than a warehouse, of much bigger size, but with a lot less complexity.

8.3 WEB PORTAL PACKAGE IMPLEMENTATION / MOBILE DEVELOPMENT

For web portal development there is a problem for estimation and sizing. Needless to say the project could be sized, but the question is whether it is necessary. Installing a portal like MS Sharepoint comes with a lot of functionalities that do not need to be developed., there was one that does seem logically sound, but might be less pragmatic for actual use.

An Australian consultancy company published a white-paper CHARISMATEK [15] with the following approach. The whole project should be sized, but the effort related to the function point count could be estimated differently. According to the method, functionalities in package implementations are delivered to the customer in four different ways

- **As is**: fully supported by the package
- **Configured**: the functionality can be configuring from built-in configuration tools.
- **Customized**: Source code level changes should be made to the package in order to deliver the functionalities
- **New**: Functionality is not available in the system and needs to be developed for the users.

It is clear that not everyone of these activities, while contributing in the same way to the functional size, do not take the same effort when implementing them. The effort again is depending on the expertise available at the company and should be derived from historical data in the company itself. Therefore the following table is proposed:

In this way sizing and estimation can still be done for portal development. Sizing the entire installation of a portal like MS Sharepoint

would probably be a waste of time, so it depends on the equilibrium between effort invested in estimation and the value of the estimation in the end. An alternative could be to only count the configuration and development parts, and making an expert judgment estimation on the unconfigured parts of the solution. furthermore it might be possible to get the exact count for a sharepoint installation, or part of its standard modules because public data is available on this. This is unlikely, yet not impossible. Another problem with package implementation as discussed by Chaturvedi et al. [16] is that a part of the requirements might be covered by existing functionality in the application, although not all the functionality implemented by the package is actually required by the client, but it comes automatically with installing it. So the delivered functionality will increase without any reason. It is not extra work requested by the client just as it is not accidental extra functionality. Chaturvedi et al. [16] proposed package points as an alternative. They argue that function points are not suitable for package implementation due to the difficulties raised above and it therefore needs its own method. The example above, even if not very pragmatic, shows that this is not totally necessary. The extra functionality should not be taken into consideration, at all. Not as final delivered functionality, not as requested functionality. The most important lesson to take from all these small ideas is that some sort of functional sizing can be done, but some creativity should come into place. It is worth noting that the proposed solutions are only available in some white-papers of relatively small companies working in the field of software cost estimation and are in no way academically verified.

Mobile development

Due to the fact that mobile development for small applications, for platforms like iOS and Google Android has been focus of attention only recently, not much information was found regarding estimation and sizing. However functional sizing should not be a big problem, the screens in the applications are outputs often using writes to and from a database. In NESMA these are often EI, EO and EQ communicating with an EIF. One ought to be careful. When decomposing mobile development and looking at merely the development of a phone application, the danger is that the object to be sized is too small to be used for this kind of estimating, as software sizing is not recommended for small projects (< 100 FP) due to its coarser granularity.

9

PART 1: RECAP

Using a selection of the techniques to overcome the problems as discussed in the previous chapters we explored solutions in which NESMA FPA can be used to estimate before and during developing when using scrum. We consider this specific application of the mix of methods NESMA Agile FPA (NAFPA). NAFPA could perhaps not only be used for estimation and benchmarking purposes, but even serves as a meaningful baseline for other purposes, like project control and productivity measurements. Using techniques advocated by NESMA itself for many years as well as some new approaches from the COSMIC method, we have shown that the following activities are theoretically possible using NAFPA:

- Sizing high level requirements up to the level of user stories and beyond.
- Keeping track of the changes during development and at the end.
- Making estimations at the beginning as well as during the process.
- Separate developed functional size from product size.
- Calculate velocity (productivity) in way that it can be compared across projects and even companies.
- Handle the non-functional requirements.
- Handle the modern software artifacts.

In the next chapters NAFPA will be tested by applying it in a real software development project.

Part II

APPLYING NESMA AGILE FPA

In this part of the thesis we will validate the NESMA Agile FPA method by performing a case study on a software project developed in an Agile development. Each sprint will be counted to see the development and adjust the scope. Afterwards the project will be analyzed using the values obtained.

10 PROJECT AND APPROACH

10.1 CASE PROJECT

The project that was used as a case study for validating our results, is a portal application developed in JAVA meant for Employee Health Management (EHM) in large organizations. The web application framework LifeRay will be used as the backbone for development. Furthermore, different JAVA portlets with standard functionalities developed by ISDC in the past will be introduced to save time. The project was done from March 28th 2012 until October 1st 2012 and was supposed to last 3 sprints. In the final stage (User Acceptance Test) changes were still being implemented. Therefore this was counted as a virtual 'sprint 4' . The counting and validation was done from December 2012 until March 2013. It would have been preferable to test these assumptions in a live environment, but time restraints and other practical issues did not permit this. The following assumptions were made and verified to make sure it can still serve as a valid test environment for the proposed approach:

- The beginning of the project was well documented and could serve as enough input for a detailed estimate.
- The changes occurring during development were well kept in an issue tracking system.
- There was a project member (requirements engineer) available to test assumptions and ask questions that was highly involved in the software development project.
- The project was counted chronologically, which means first the high level estimate was done and then the sprints were counted in chronological order.

Some changes with regard to the proposed guidelines had to be implemented which will be discussed in the next chapter.

10.2 APPROACH

The approach to be followed is the one discussed in the previous chapters. As usual for each method, a method provides a guideline that should be tailored to the context. This case was not so different. For each project a counting approach should be thought of. In the case company this is usually done at the beginning of the project in agreement with the team members and a QA officer.

In this case, the following was decided:

A. **As the requirements documentation was of high quality and detail, the estimation would be done using the detailed count.**

This is unfortunate because this means that the possibility of using the functional areas as discussed in section 6.3 cannot be tested in this thesis. This part of the method is therefore discussed in future research.

B. **As it was a portal application, some functionality would come out of the box.** For portal applications an approach was discussed to divide the work in a way that would justify the difference in hours. This could be done by dividing the functionality in three segments: 'Modification', 'Development' and 'Configuration' as was discussed in 8.3. Because the historical data did not allow a clear distinction between modification and development, this was combined in one segment. This left only two: Out-of-the-box (OOB) or Development 'DEV'. Please note that as an alternative, the hours allocated to a single Function Point could also be adjusted.This would make sure to make sure the fact that parts of the application should require less programming is taken into account in the whole. This is a more raw approach, which is less time-consuming, but does not show which specific parts would take less time than others. Also when comparing different project the extent to which re-usable components are used will not be taken separately. On the level in which we specify the size of each user-story or as we will see beneath the modules, this is mandatory. A user-story about LifeRay standard functionality should not be treated the same as a user-story about a brand new module to be developed.

C. **User-stories were not counted as such, but a more coarse-grained approach was taken to separate different 'modules'** The way the stories where put did not allow us for a separation in user-stories as proposed in section 6.2 instead we looked at each functional process separately and divided the project in more coarse grained 'modules'. This was chosen because even though in the case company there is a lot of understanding about scrum, this is not always the case for the clients. The 'stories' in the end respected more the different tasks resulting from the Work-Breakdown structure and referred to the requirements document than it respected the actual format for user-stories. This made sure that no clear user-stories in the form proposed were available in this case. The modules were constructed based on grouping on functionalities and the different 'objects' in the requirements. The method developed in the previous part was done based on the general specification of a Scrum project and as with every method should be flexible enough to be adapted to different contexts.

With these project specific modifications in the method, the estimates and tracking of the project progress was started.

11 ESTIMATIONS AT START OF PROJECT

11.1 PRE-SALES

First a count was made of the Pre-sales, as would be expected to be the case when a whole estimation process is done using software sizing. Therefore an estimated count is done on the high-level requirements as received from the customer. The goal of the measurement is the estimate the effort that should be done and the final size of the product on completion of the project. This count was based on the following very detailed documentation:

- The technical documentation
- The functional specification
- A mock-up of the functionality requested
- An overview of the screens requested.

This Pre-sales count resulted in an estimate of the the following values:

Kind	FP DATA	FP TRANSACTION	Total
DEV	49	486	**492**
OOB	57	138	**226**
Total:	**106**	**624**	**730**

Table 8: The Pre-Sales count of our case project, divided in Development hours (DEV) and Out-of-the-box hours (OOB)

An amount of hours could be attributed to the counted FPs for estimation purposes. We chose to this respect:

1. 5 hours development for Development hours (about the company average for JAVA development, which usually has some ready-made components as well)
2. 1 hour development for OOB hours (time taken to configure, this was a guess without any real foundation, as this is the first time in this context an estimate is performed in this way)

This would mean 2870 hours of development pre-estimated. The hours for testing, requirements analysis and all other factors are derived from the development hours estimated. This is company, project and team specific and is therefore outside of the scope of this investigation, which will focus merely on the development hours.

After the Pre-Sales count a planning was made to divide the functionality over the sprints.

The planning was already done, as a past project was used. Even though the information from the count may have lead to another division of tasks, the planning already made was taken to see how this related to the functionality. This resulted in the following figures, which can be seen in table 9.

Sprint	type	FP DATA	FP TRANSACTION	Total FP	Total Hours
1	DEV	29	146	175	875
	OOB	29	23	52	52
tot.		**58**	**169**	**227**	**927**
2	DEV	20	205	225	1125
	OOB	14	4	18	18
tot.		**34**	**209**	**243**	**1142**
3	DEV	0	135	135	675
	OOB	14	111	125	125
tot.		**14**	**246**	**260**	**800**
Total		**106**	**624**	**730**	**2870**

Table 9: The planning as made by the project team, divided, sized and with estimated hours according to the FP count.

In the table something strange is shown. In a normal project, one would expect the velocity in the sprints to increase slowly. Therefore also the amount of hours planned should be either relatively stable over the sprints climb. In this case, the functionality implemented is increasing over the sprints, but the amount of estimated hours is unstable. This is due to the choice we made to make the distinction between OOB and DEV functionalities and the fact that the team or team leaders did not take into account the same measures, as WBS and expert judgment was introduced as the input to the planning.

The first count serves as a similar idea of a quantified product backlog and it will be divided in 3 sprints accordingly. The planning in Function Points for the different sprints will be taken as the baseline and will be adjusted during the project.

12 SIZING DURING PROJECT

In this chapter the result is shown of the sizing done during the project. This theoretical means: after each sprint the functionality delivered is counted. In practice this was simulated. As timing issues did not allow a clear count done after each sprint in a live project. An old project was given and with help of documentation like the issue tracking done during the project and with help of a member of the team, a simulation was made of the different sprints and the activities performed during development.

Even though this would not reflect reality as much as to do the actual sizing of the software delivered, it gives us enough ground to explore whether it makes sense to size a project using the proposed method.

For each sprint we will perform the following steps:

- Look at the actual developed functionality compared to the estimated one.
- Look at the actual hours made and the estimated amount of hours.
- Determine the project progress
- re-estimate the remainder of the project.
- Determine the functional velocity of the sprint.

12.1 SPRINT 1

In sprint 1 a basic function point count was done, as we assume there is no developed code to change yet. The first sprint took place from March 28th to April 17th 2012. In total 860 hours were made during this interval.

The following FP values were counted:

Kind	FP DATA	FP TRANSACTION	Total
DEV	47	172	**219**
OOB	22	18	**40**
Total:	69	190	**259**

Table 10: Actual result in FP of Sprint 1

While the estimate was in fact:

Kind	FP DATA	FP TRANSACTION	Total
DEV	29	146	**175**
OOB	29	23	**52**
Total:	**58**	**169**	**227**

Table 11: Original estimate in FP of Sprint 1

This means that 14% more functionality was developed than planned. Furthermore there was more developing done than out of the box functionality than initial estimated.

Now the scope can be re-evaluated, the progress can be determined and we can evaluate the original estimate. Furthermore by looking at the hours we can calculate a value similar to 'velocity' as it is done in SCRUM.

Project progress

The results obtained from the first sprint would give us an indication of how far we are in the project. This can be quantified in a percentage of the work done compared to the work that should be done to finish the project. In terms of size, this is: FP developed / FP to be be developed.

FP to be developed was originally 730, but we can re-estimate this to take it into account. For this we use the proposed method and we assume that, as always, the truth is black nor white and lies somewhere in the middle. To makes this clear the three assumptions are made to re-evaluate the scope, as was proposed in 6.4.

1. The 'wrong estimate' or 'ahead of schedule' assumption. The scope remains unchanged: **730**.

2. The 'incident' assumption. This means that an extra 32 FP is added to the end-goal, which makes it **762.**

3. The 'representative for project' assumption in which we assume that there was a scope creep of $32/227 \approx 14\%$ and we expect that this will be done as well for the end-functionality. This means the end scope will grow to **832 FP.**

So we see that through these 3 methods the progress can vary between 31 and almost 40 percent and the end-scope will be between 730 and 832 FP. There are in general two approaches to this handling this uncertainty. One is to work with ranges. The range [730 - 832] would be the end-goal. However to make it easier, we will take a weighted average, with the weights: 0.2, 0.3 and 0.5 as this is a rough estimate on the likelihood on all three assumptions. In practice, these should be filled in by either the judgment of the team or some experts or a benchmarking system has to be used to validate these constants. The following values were calculated:

End goal estimation method	1	2	3	Weighted Avg. end goal
End goal FP	730	762	832	**791**
Developed	259	259	259	**259**
To be developed	471	503	573	**512**
progress in %	**39.6%**	**34.4%**	**31.1%**	**32.7%**

Table 12: Evaluating the end-goal in FP for the following sprints after sprint 1

If we take this weighted average of the end scope, which amounts to 791, we see an increase of 8.4%. This gives us a progress of 32.7% and the estimated remaining amount of FP to be developed is 532 FP. Given the fact that productivity tends to increase over sprints, this is a reasonable guess. This way of determining the end-goal will be repeated throughout the other sprints.

Adjusted estimate

This adjusted estimate is an increase of 8.4% on the end goal. This means that we can adjust our original estimate for the remaining sprints:

This is done by comparing the end-goal to the other estimates. It shows an increase of 29 FP, which we divide evenly among the estimates.

The adjust estimate can be seen below. In hours, using 5 hours/FP this means that 70 hours need to be probably made extra for sprint 2 and 75 hours for sprint 3. The results of this re-estimation can be seen in table 13.

Sprint	FP estimated	FP re-estimated	Hours re-estimated
2	243	257	1212
3	260	275	875
Total Todo	**503**	**532**	**2087**

Table 13: Re-estimation of the following two sprints after sprint 1

What happened can be easily displayed by using a burn-down chart, in which we clearly see the scope increase and the remaining functionalities still to be implemented.

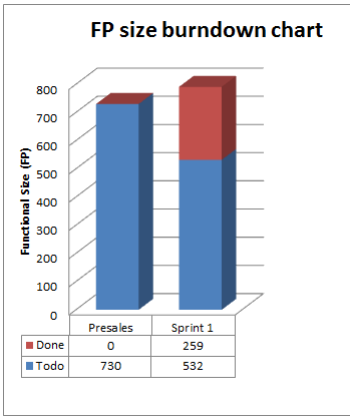

Figure 19: Burn-down chart after the first sprint

Functional Velocity

As is usual in sprints, the velocity should be determined for estimation and productivity benchmarking purposes. The amount of development hours used for this sprint was:

860 / 259 = 3.32 Hours per FP. Which were a bit less than the average counted at the company. Unfortunately the hours spent on each DEV / OOB were not available. This means that this would be solely used for estimating purposes.

12.2 SPRINT 2

Sprint 2 took place from April 18th until May 8th 2012. In Sprint 2 we counted the added functionality as enhancement on the first sprint. This lead to the following values.

Kind	DATA		TRANSACTION		Total FP	Total EFP
	FP	EFP	FP	EFP		
DEV	15	21.25	199	199	214	220.25
OOB	17	17	14	14	31	31
Total:	32	38.25			245	251.25

Table 14: Actual values from Sprint 2

The re-estimation of the scope for this sprint was **257 FP,** While the original estimate was:

Kind	FP DATA	FP TRANSACTION	Total
DEV	20	205	**175**
OOB	14	4	**52**
Total:	34	209	**227**

Table 15: Pre-Sales Estimated values Sprint 2

In which we assume that the functionality developed more was due to implicit client changes or underestimation of the function points before.

The total product size at the moment is 503 and the total developed size is 510.25 A 2% difference. The product size is 503 instead of the estimated 506, which is a difference of around -1%, and therefore too trivial to change anything. A 2% difference for the end goal would mean 801 EFP at the end.

This gives us a more interesting result, not only do we have a (small, even negative) difference in the number of function points developed vs estimated. When we compare this to the effort (Enhancement function points) we see that in the end more effort needs to be done to get to the end-goal, even though the scope decreased.

Project progress

Regarding the project progress, again we can make the same adjusted estimated as performed after sprint 1.

Again we can make the same assumption as in the previous sprint, we see here what happens when the estimated functional increase was more than the actual one and how to handle the developed size vs. the product size.

1. The adjusted scope remains unchanged: 791. The EFP adds to 801, because the 10 extra FP were done, so they will be present as well in the end.

2. A deduction of 6 FP from the end-goal which amounts to 785. On the other hand, 10 EFP are added to then end goal EFP, for the same reasons as mentioned above.

3. The scope did not creep, but shrank with -6/500 = -1% and we expect that this will be done as well for the end-functionality. This means that the to be developed will 795-500=295 - 1% = 262. Besides this we assume that the EFP percentage of 10/241 = 4.15% will be valid for the last sprint as well. This means 262 + 4.15% = 277, which we will add to the already existing 510 = 787 EFP.

For the percentage of completion, we can make an approximate end-goal visible. However, we have two end-goals now, the product size as well as the developed size.

With other words, the result of this sprint have little impact on the project progress. However, given the average, we could adjust the estimate for the rest of the sprint again.

Given the 775 as end goal, still 272 FP has to be developed. Given this assumption on the EFP this would mean 282 EFP To be developed. This is an increase of the original 800 hours planned with 110 hours.

End goal estimation method	1	2	3	W. Avg
End goal FP	791	785	762	775
End goal EFP	801	795	787	792
Product size	500	500	500	500
Developed	510	510	510	510
% Done FP	63.2%	63.7%	65.6%	64.5%
% Done EFP	63.7%	64.2%	64.8%	64.4%

Table 16: End goal re-evaluation after Sprint 2

Sprint	FP estimated	FP re-estimated	EFP re-estimated	Hours re-estimated
3	260	275	282	910

Table 17: Re-estimation after Sprint 2, Sprint 3 is re-estimated

Adjusted estimate

Given these facts we can also adjust the estimate again and display the progress in a burn-down chart.

The hours for sprint 3 will increase with 35 hour due to the 7 extra FP on the EFP.

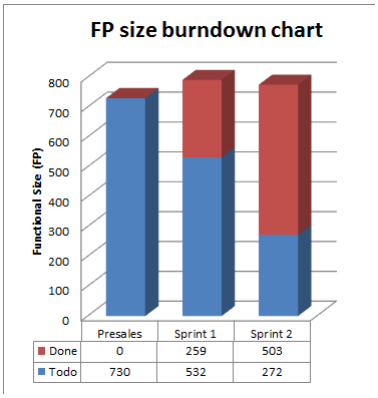

Figure 20: Burn-down chart after sprint 2

Velocity

The Velocity in this sprint was 1190.5 / 241 = 4.93 hours / FP and 4.74 hours / EFP. This is an increase of the previous sprint and given the OOB/DEV division made, should be about the velocity estimated in the planning stage.

12.3 SPRINT 3

The last planned sprint was done from May 9th until June 10th. Notice that this sprint took longer than the other sprints. This is remarkable as sprints normally span roughly the same period.

Kind	DATA		TRANSACTION		Total FP	Total EFP
	FP	EFP	FP	EFP		
DEV	7	14.5	51	51	58	66
OOB	14	14	209	209	223	223
Total:	21	28.5	260	260	281	289

Table 18: Actual values Sprint 3

The re-estimation gave us 275 FP and 282 EFP. The original estimate was:

Kind	FP DATA	FP TRANSACTION	Total
DEV	0	135	**135**
OOB	14	111	**125**
Total:	14	246	**260**

Table 19: Pre-Sales Estimated values Sprint 3

Re-estimation for this sprint seemed to have at least some effect.

In this Sprint some new functionalities were developed that were not in the original requirements. We know this because of proper change management in the project itself.

Even though the estimates and the development makes sense and we basically got to the latest estimated end-goal or even further, a lot of functionality was not implemented the way the customer saw fit. During the user acceptance test, quite some functionality was still implemented. In a normal 'live' situation this should have been handled by doing an additional enhancement count. However, unclarity in documentation and time issues did not allow this to happen. Therefore the last part will be considered as an extra sprint in which the changes that were unforeseen where implemented. Note that this means that even though the project until sprint 3 seemed to be on schedule, the unforeseen UAT which was originally intend to be used for bug fixes, makes sure that the end amount of functionality as well as hours got higher than estimated.

The burn-down chart could be displayed as well.

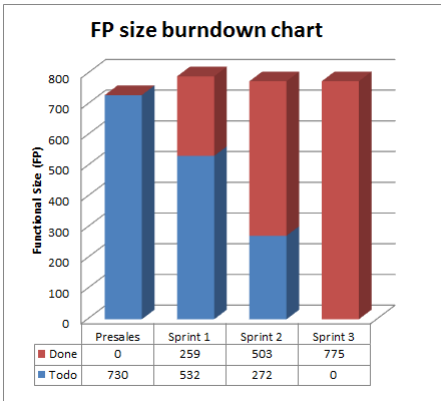

Figure 21: Burn-down chart after sprint 3

Velocity

The Velocity in this sprint was 1995 / 275 = 7.25 hours / FP and 1995/289 = 6.9 hours / EFP. This is an increase compared to previous sprints and can be attributed to an increase in bug fixes that needed to be done for the previous sprints, these take time away from the developers, but should not be taken as any functional change, as they are merely a quality issue.

12.4 USER ACCEPTANCE TEST (SPRINT 4)

The user acceptance test was the time when the development should be done, but it would be tested thoroughly before going live. This lasted from June 11th until October 1st 2012. The work done during this period were mainly bug fixes, which should not contribute at all to any change in functionality. Besides that also some changes were implemented, these will be counted here.

As this was anticipated before by the project team, this would normally mean that there should be another estimate done. Due to the fact that the project information provided was not clear enough on what the status was of these changing before starting the UAT, this could not be done. This was merely counted as the extra effort it took to complete this project. Furthermore as this project would be the base of several other enhancement projects, this would give us in the end the baseline for these future enhancement projects as well.

In the end, the added functionality in this phase of the project was:

Kind	**DATA**		**TRANSACTION**		**Total FP**	**Total EFP**
	FP	**EFP**	**FP**	**EFP**		
DEV	0	6.3	51	91	51	97.25
Total:					**826**	**896**

Table 20: End result of the last part of the project: the size growth during the User Acceptance Test

3099.75 Hours were spent in the UAT period, giving us a productivity of 60.8 hours / FP or 31.9 hours / EFP. These figures are extremely high, which is caused by the work done in the UAT mostly concerned bug fixes.

The final burn-down chart looks like this:

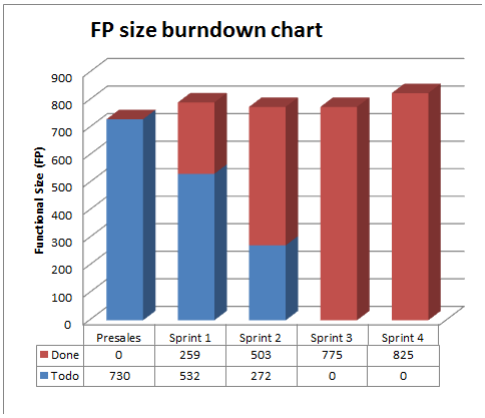

Figure 22: Burn-down chart after sprint 4

As stated before, the changes in the UAT, in the context of this research were not taken as estimations, but they were counted in the end, as they are a vital part of the project. This makes sure that the last two columns show nothing as To Do value, even though the Done part of the project is increasing.

13 PROJECT END

Now that the project is done we can analyse the results and the information obtained from the project. The rework in terms of functionality of the entire project was **8.5%** which is extremely low for a project that was perceived by its team members as chaotic and unpredictable and with a client that changed their mind quite often. On the other side, a bit of the chaos can be uncovered by comparing at the original estimate with the product size: **730** vs. **826** which is not a huge increase of functionality either. **730** vs. **896** for the effort together make an increase of **22.7%**, which is still acceptable given the fact that an increase could be considered usual.

Even though the SCRUM method and division in sprints were used during development this project could not be considered to be Agile in its total form:

- The requirements were very detailed from the start.
- Changes were introduced in the project and handled, but mostly implemented in the end.
- The software delivered at the end of each sprint still had a lot of rework to be done and bugs to be fixed.

It is for this reason that the tracking and scoping was not as successful as expected.

13.1 ANALYSIS USING NESMA FPA

In the end of the project it is possible to analyze what happened in terms of size increase and effort during the project, how this related to the first estimations and how the productivity can be defined in terms of added functionality. Furthermore we could adjust our estimates during the project and take into account an average productivity to adjust the estimate in hours.

13.1.1 *Analysis on size growth.*

A lot of different yet resembling figures have been collected over the course of the sprints. While these were handled during the sprints, at the end of the project they should be used for evaluation and benchmark purposes, the latter being only applicable when more projects are done or will be done using similar sizing techniques.

The size growth can be analyzed and displayed in several ways. Some will be explored here.

For benchmarking the following figures should be important:

- The product size vs. developed size, to relate in an increase in hours with the increase in changes.
- The estimated size vs, the actual or developed size in hours as well as EFP.
- The 'velocity' in general of this team.

We will see that not all of these metrics show what we would expect.

13.1.2 *Analysis on module level*

Now that we did several counts, we can analyze the difference in size in the module level, this will give us an overview of what parts of the software were specifically difficult to predict or develop. This will explain the size difference in an understandable way and gives an quantification to use when discussing with the client which parts were specifically bound to changes or bigger / smaller than initially anticipated. This can prove useful in deliberation, as a background for the analysis or for benchmarking purposes.

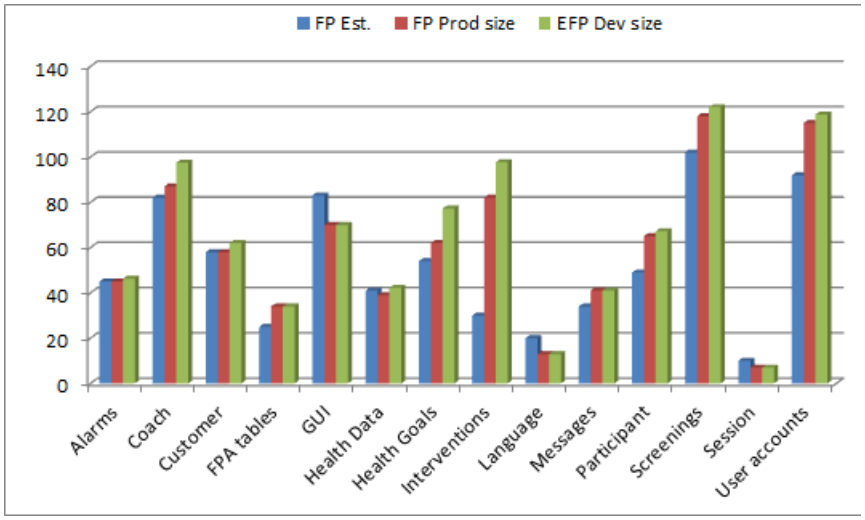

Figure 23: Module-level analysis on estimated size, product size and developed size

Now we can clearly see what influenced the growth. Interventions for example, got a lot bigger than it was estimated to be, this was because intervention packages entered the product, and these were not explicitly specified before. Seeing these differences does not imply that we have a clear reason for the differences occurring here, as there are several reasons why a change can occur the way it did:

- There was a mistake made in the estimation, counting might not have been done in exactly the same way.
- Some functionalities got attributed to a different module while counting them post-sprint. (see language and session for example)

- Architectural choices leading to a lower count: EIF instead of ILF, external module which takes away functionalities from the counting scope.
- Functionalities were added in grooming sessions or talks to the client.

The only way to make this distinction is having change management and decent issue tracking in place as well as awareness in the project team about these aspects when in conversation with a client, so no functional changes are made disguised as bugs or misunderstood requirements. This overview could give a reason for the project manager to investigate things further in case some growth in modules was overlooked.

13.1.3 *analysis on productivity*

When we look at productivity as discussed above we have the following figures, obtained from measuring the product size, developed size and the hours made in the sprints:

Sprint	FP	EFP	Hours
1	259	259	763.25
2	241	251	1130.25
3	275	289	1807.25
UAT	51	97.25	3099.75
Total	**826**	**896.25**	**6800.5**

Table 21: Analysis on the FP developed and the hours made during the three sprints and the UAT

This gives us the following values for the overall productivity. :

- The overall productivity in FP is 8.2 hour/FP
- The overall productivity in EFP is 7.6 hour / FP

When looking at the total average this is a big higher than the average in the company for JAVA projects with re-used components, which is around 5 FP, as pointed out before. According to some websites the industry average for JAVA supplied by ISBSG[1] is 10.6 [2] so from that respect it is not too bad at all and more importantly for this research, it seems to be a valid result.

When looking at how the productivity developed over the course of the project, it is best to do this in FP or EFP per hour. Normally velocity is also put in hours / FP. This would lead to a strange visualization. If one would go from 3 hours / FP to 6 hours / FP. Visualizing

1 ISBSG is the biggest repository on function size available, it can be found at: www.isbsg.com

2 Taken from: http://www.drdobbs.com/jvm/the-comparative-productivity-of-programm/240005881 retrieved: 17/5/2013

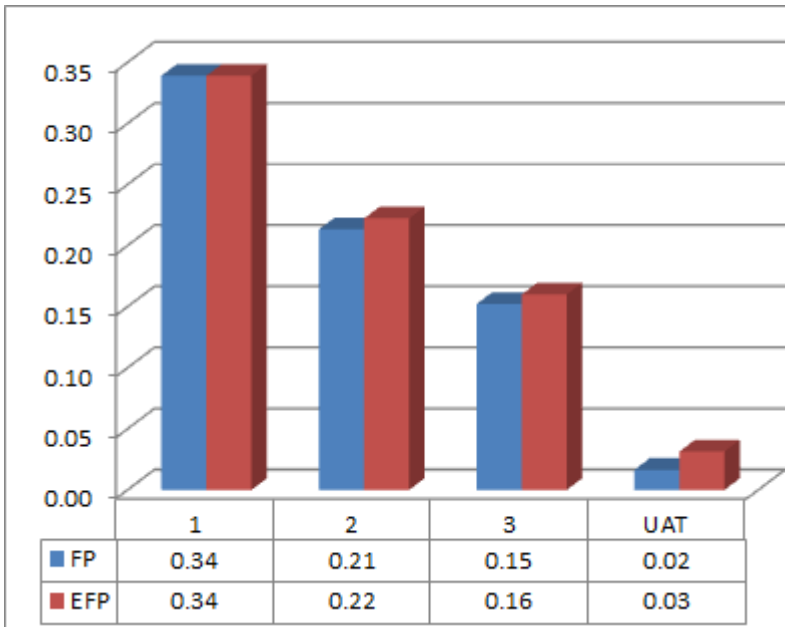

Figure 24: Functional productivity in FP/Hours and EFP/Hours for three sprints and the UAT

it would show an increase. In terms of velocity or productivity this can be misleading as the productivity is decreasing, not increasing. When calculating the productivity figures we see the the following as visualized in figure 24.

This is a peculiar result, as 'velocity' as proposed by the SCRUM method, should make sure that productivity should be increasing and moving towards a point of 'hyperproductivity'. When looking at more detailed data from the project, the reason becomes apparent: The more software developed in previous sprints, the more bug fixes that can occur in the following sprint. This is done to the point of the UAT, where all the changes and remaining bugs are resolved. This leads to a lower productivity in FP, as no functionality is added or really changed in the process, but a large number of hours are spent on development. To some extent this could be covered by FPA as well, if we count the bug fixes as one or a few DETs, FTRs or RETs changed without any impact on the functionality, we would still get some EFP value for it. This would mean that bug fixes are counted as part of productivity and normal development. This intrinsically means that providing low quality software could improve productivity, which is not recommended. It also shows one big advantage of the method. The power to make clear what in terms of extra hours could be due to the supplier (mistakes in earlier development) and what is due to the client changing their mind. This is not different from the usual change management, but could be used to check this change management or quantify it to greater extent.

14

RECAP

A case study was on a project in the range of 700-800 FP. In fact 720 FP was the initial estimate, but 850, FP were developed. Due to changes during development the EFP counted was 896.3. This was an increase of roughly 24.5% in development. If this would be true for effort as well, it would be great, as IT projects normally are bound to go over the original estimates this would be a good score. Unfortunately this was not the case. The original estimate using NESMA FPA was 2870 while the actual hours made were 6800.5. This is an increase of 137% . Even though the original estimate in expert judgment was even worse with roughly 1955 hours.

The project needed some modifications from the method proposed:

- The requirements were detailed from the start already, so using indicative or estimated FP was out of the question.
- It was a portal application with out of the box functionality using labels on the functions to separate out of the box functions from actual development
- The user-stories did not have the right granularity to make them suitable for grouping. Comparing was done on modules instead of user-stories.

In the project we have seen increases as well as decreases in the functionality estimated vs. the functionality developed, we have seen differences in the productivity and even the length of the sprints in time.

Even though the project used in the case study could not be considered to be fully Agile, it does show some remarkable things:

- Velocity in terms of functionality is declining while velocity is normally increasing to the points of hyper-productivity when looking at the normal way SCRUM is advertised.
- the difference between the EFP and FP is relatively small, which is caused by most changes occurring before the functionality is developed and therefore it is not counted as an enhancement as such.
- The difference between the estimated functionality and the developed functionality shows an increase of 15.3%, while the amount of hours spent show an increase of 237 % This means that the estimation power of estimating FP is bigger than the estimation in hours using FP analysis. This will be elaborated upon in the next part.

Reasons for these phenomena are widespread, but the main concern seems to be a lack of Agile maturity in the team-client combination.

While implementing new functionality, a lot of bug-fixing was still done on the functionality delivered in previous sprints. These hours were not allocated to the sprints in which the functionality was developed.It might be interesting to do this, but it should be done outside of this thesis, as time and information available did not allow us to easily get this information.

Part III

VALIDATING NESMA AGILE FPA

Now that the method is tested in a case study, the original research question can be answered. First the estimations are critically looked at. Then, the effort of the measurements is inventarised. Flaws and points of improvements that were found will be stipulated. After drawing our conclusions, the research itself will be critically discussed and future research ideas will be given.

15 INTERPRETING THE RESULTS

In the previous chapter we have shown the measurements done in our case study, which was a project done in three sprints. Some additional work done in the User Acceptance Test. We have seen that it was possible to connect the quantity of functionality delivered with the estimates done in the sprints and to the original estimates. Furthermore scope creep could be anticipated upon and we came to some productivity measures that can be compared to other projects and even other companies.

15.1 ACCURACY OF ESTIMATIONS

Now we can interpret the results of the case study and draw conclusions regarding the proposed method. When we look at the actual project estimate, we see that the original estimate was 1700 hour with 15% risk percentage so this makes roughly 1955. The estimation done in FPA was around 2870 hours. This is quite a difference. The similarity is this respect is the fact that both estimates differ from the actual value to great extent.

When looking at the accuracy of the estimations and re estimations done regarding the (enhanced) functionality we see the following:

Sprint	FP est.	EFP est. 2	EFP est. 3	Total EFP dev
1	227			259
2	243	257		251
3	260	275	282	289
UAT	0	0	0	97.25
Total (scope)	**730**	**759**	**792**	**896.25**

Table 22: Overview of the different sprints and the three chronological effort estimations in hours in EFP

As was clear from the previous chapter, due to changes being delayed to the point of the UAT, the estimates in the end were quite different from the end result. On a positive note, what can also be seen is that for the sprints, the re-estimates were getting better with each estimation. For the total, they were note.

If we would do the same for the hours, a totally different conclusion can be drawn:

Sprint	Hours est. 1	Hours est. 2	Hours est 3.	Total Hours
1	927			763.25
2	1142	1212		1130.25
3	800	875	910	1807.25
UAT	0	0	0	3099.75
Total	**2870**	**2850**	**2803.5**	**6800.5**

Table 23: Overview of the different sprints and the three chronological effort estimations in hours

As was clear from the previous chapter, the amount of hours were mostly made in the last sprint and the UAT. As the functional size seems to be crawling towards the actual result. the hours made in these sprints sure were not. Less functionality was developed than anticipated in sprint 2 and less hours were made than anticipated in the first two sprints. This makes sure that actually the accuracy of the estimation in hours during the sprints becomes less.

When looking at the estimations and its accuracy it is important to make a distinction between the accuracy of the estimation in functional size and the accuracy of the estimation in effort.

The accuracy will be determined by calculating the estimated values with the actual value, using the following formula:
$\frac{Estimated-Value}{Real-Value}$ Independent of what these values are.
This can be compared with the cone of uncertainty as explained in section 2.2 about estimation.

From the previous interpreted results we already know that all the estimates were below the actual values, therefore only the below part of the cone is displayed, this will give us a good overview on how the estimates are relation to the 'rule of thumb' of the cone of uncertainty.

One difficult thing is that the cone of uncertainty was originally developed for waterfall projects. This is solved by mapping the stages of the cone of uncertainty to the different sprints.Assuming that during development, the developers are mostly doing development and test on the functionality they selected as user-stories with the highest priority, the functionalities in the future sprints might be defined only as (high-level) requirements. Given that the developed part of the solution will become bigger over the course of the project and the to-be-defined or to-be-designed part gets less, a correspondence with the stages pointed out by the cone of uncertainty can be made. To do this, the following reasoning is applied:

A. Sprint 0 is the Pre-Sales count done. This should be related to 'planning and requirements' as the requirements were the input to this first count. As we know the requirements were quite detailed from the start, it might have been more similar to the product design stage. However, uncertainty about some decisions and the actual architecture compensated for this. Nevertheless the cone of uncertainty might be considered a bit pessimistic in this respect.

B. Sprint 1 is related to the product design stage, as was specified above. The fact that development and test was already done for less than one third of the application, is compensated by the fact that the remainder of the project needed further specification.

C. Sprint 2 is related to detailed design for the similar reason as for sprint 1.

D. Sprint 3 is related to the development and test stage, in that stage development and testing was done on all parts that were not developed yet.

E. Sprint 4 is the end of the project, therefore it is always 1 and mapped to the project end. All estimated end here.

Given the choices made in this mapping, one would expect the values to fall inside the cone as the stages in waterfall should have more insecurity at those given moments than the sprints. The result can be seen in figure 25.

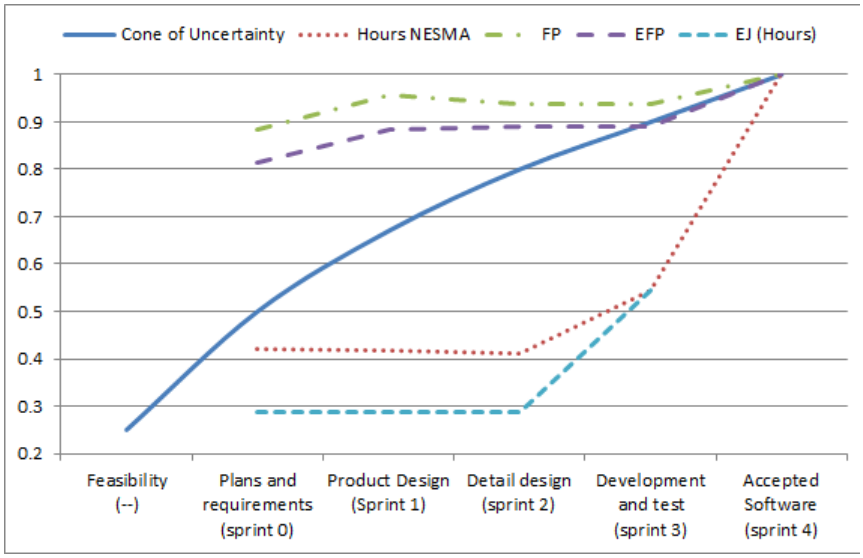

Figure 25: Visualization of the cone of uncertainty with the re-estimates of the end-goals over the course of the project. With hours in NESMA, the original EJ estimate and the FP and EFP values

Looking at the estimate for the final size in Function Points, we see that they fall within the cone. This is by all means a good result. One side-note is that the estimations do not improve over time and even get less. This is due the most of changes being done at the end during the user acceptance test. Furthermore, in the second sprint less FP were developed than estimated compensating for the creep in scope from the first sprint.

The hours estimate using FPA and Expert Judgment, is below the normal cone. This means that the estimating power in this case was less than expected. This is due to the fact that there were more bugs than usual in the code as well as a supposed underestimation of the productivity in general. The estimation does not improve, until sprint 3. This improvement is mostly due to the amount of hours made exceeding the amount of hours estimated until that point.

When looking at the estimate for expert judgment we see that certainly at the beginning, this is the worst estimate. This is could be due to the complicated nature of the project done and the same underestimation as was the case for the hours estimated with NESMA. Unfortunately due to restrictions on the available data it was not possible to re-estimate using planning poker. . Therefore the initial Expert Judgment estimation was kept.

15.1.1 *Using the Function Points as a tool for contract negotiations.*

When the assumption is made that all the quantified extra work done is actually used in the contract with the client, we can retrieve the following information:

- The amount of hours estimated of 2870 were based on the estimate of 730 FP
- The actual developed functionality was 896 EFP. These are 166 FP extra.
- The supposed rate of 5 hours / FP would mean that the client got 830 hours of extra rework, regardless of the fact of the client wanted that extra functionality, which as is assumed he did. This should always be covered by proper change management, as was the case in this particular project. Even when FPA can help determine the functional rework, it should never make proper change management obsolete.

This means (hypothetically) that 3750 hours are accounted for by the client, as he got more functionality than was agreed on and the clients accepted the rework. That leaves 3090 hours for the supplier. So out of the 3920 extra hours made on this project, 830 will be paid for by the client and 3090 will be for the account of the supplier. The client and supplier might choose a different velocity to calculate with, which could be closer to the reality. This makes contract negotiation fairly simple and would provide a fair base for the client-supplier relationship. Whether the supplier would be satisfied in the above example is doubtful. What is not taken into account in this example is the changes made before development. It might be the case that strategy or the software architecture can change dramatically without it having the impact accounted for by functional size. In the project suddenly the change was made from having email functionality, to using a reusable component for emailing with templates for example. Even though no functionality has been implemented, other designs, researches or mock-ups made by the team were in vain, even when they never reached the point of being implemented totally. These are hours made by the team which are not accounted for by functional size, but for sure could be billed to the client.

Whatever the client and supplier decide, at least there are some quantified measurement available to help them see how to distribute the project costs, if necessary.

In this section the effort to work with NESMA in the way proposed will be discussed and compared with expert judgment.

We see that expert judgment for a project this size should be about 1-3 experts, providing expensive hours 4 hours of preparations and a workday of deliberations.

Function point counting depends on the quality of the documentation but could be estimated at around 500 FP in a day by experts [1]

.

In the case this was a lot less due to the counter being novice.The fact that in the proposed method of using FPA, the following needed to be done:

- Make sure the functional processes counted were either OOB or DEV.
- Count every functionality twice (one for high level, one for after - sprint) means that the counting effort will be at least double.
- Count the right enhancement for the changes

This made sure that the counting done in surpasses all the other counts for the other projects in terms of effort needed to do the estimations.

In fact, in principle it could be about 3 times the effort of a normal function point count. At least everything will be counted twice, sometimes even more rigorous.

In this case we estimate the total effort to be 720 for the high level count is about 12 hours.

For the sprints with enhancement this would be 896 which is about 14 - 15 hours.

Taking into account that all 5 counts done entail starting and finishing those counts, some extra hours might be added to the whole. This amounts to about 30 hours. That would be the investment of counting hours proposed in the method. If done by a person with enough experience in counting function points. Whether this is worth is depending on the opinion of the organization that would like to use NAFPA. For expert judgment the counting time might be less, the hours will be more expensive due to the experts often being on a high position in the organization. Not too mention, the scarcest resource in terms of experts is the time they have available. The methods proposed in SCRUM encourage people to work together and get to a common understanding, not only about the estimates, also about the requirements. This is something FPA normally does not provide. The counts are typically done by one person.

1 Due to lack on proper sources regarding the effort it takes to count FP, the estimates here are taken from the LinkedIn group on Function Point Analysis where the experts said either between 150-900 FP per day, or around 500 per day. The could be accessed by http://www.linkedin.com/groupItem?view=&gid=90250&type=member&item=255821099 when being a LinkedIn member.

Furthermore, from a personal account, the analyst in the case study did not think the counting was a very pleasant task to do. This could be contributed by the fact that the analyst had no real connection with the project, or it is simply personal preference. Fact is, that it is a tedious task to perform which offers no certainty on the correctness of what the analyst is doing. This might improve over time after getting practice. Comparing it to other measurements, when using a measuring rod or ruler to determine length, one can be sure that this is actually the length of the object. When using FPA, however the analyst has no clear imagine of whether he is measuring correctly. Even when writing down all assumption, fitting the reality to the method . This uncertainty, coupled with a slow speed and the little impact this measurement had on the company or the project, made for a not so motivating environment to do the count. In chapter 17, some guidelines are given regarding the analyst that could improve this.

16 ANSWER TO THE MAIN RESEARCH QUESTION

The research question will be answered in two parts. The first part will focus on the fit of NAFPA and SCRUM regarding the process and the meaning of the resulting values after using the method. The second part will present the flaws the NESMA method has that should be solved to improve the efficiency and the resulting values of the method.

16.1 SUITABILITY OF NAFPA IN SCRUM.

Applying the method we have seen that to some extent it is possible to:

- Quantify the scope and the functionality requested and delivered.
- Track project process and take scope creep into account during the project.
- connect high level estimates to the in-sprint estimates and the actual work done
- See the amount of rework done during development due to functional changes. (functional productivity)

However, we have also seen that there are some shortcomings in the method. The technical details will be explained later. In general:

- The rework seen was very small as compared to the actual rework as perceived by the team
- The correlation to the amount of hours per function points with the estimated one showed a negative trend. From the outlier 3 hours per FP at the beginning, to the outlier 60 hours / FP at the end.
- The actual work done and re-estimate had little influence on the re-estimates.

What this means is the following:

16.1.1 *Function points show merely the functionality developed*

The function points show merely the functionality developed. What this means is that all the unfruitful hours spend by the team researching things and trying out different things are not included.

All the changes made in the architecture and the consequences thereof are not taken into account either. As soon as a piece of software is developed and then it is has to be remade, only then it add

to the developed size. This is important to take into account when looking at the measurements done. The rework in the case was little compared to the functional size. The project however was perceived as being extremely chaotic and full of changes.

The rework done which is counted by the method only applies if the following is true:

- Functionality has been delivered at the end of a sprint N.
- A change was made in this functionality in a sprint > N.
- The change was not the result of a bug-fix.

All the other changes along the way are not taken into account. This in important to keep in mind.

16.1.2 *Bug-fixing previous sprints leads to messy Agile practices*

What was shown in this project is something which happens quite often: after the first and second sprint, in the following sprints the developers are still bug-fixing functionalities that have been done earlier.

This works for the customer as well as for the supplier, as the customer will get better quality software and the supplier will have more billable hours. This is not compatible with the method, in which functionalities are clearly split among different sprints. The result is that when not working according to strict incremental software development with a strict *definition of done.* When bug-fixing becomes more apparent, along the project less and less new functionality will be delivered. This goes against the very first part of the Agile principle introduced at the beginning of this thesis: *Working software* over comprehensive documentation.

16.1.3 *Great for functional scoping, not so much for hours.*

We have seen in this project that the scope remained quite the same while the hours did not. We have also seen that when comparing the functionality with the hours worked on it the whole project seems to make little sense. This was due to a number of factors, all explained above. We have seen that to some extent the 'size' of the project was quite correctly estimated. The 'effort' however was not. It seems that there are more factors contributing to the effort part. To have functional size as the only parameter to the parametric cost estimation seems therefore utopian.

16.2 SHORTCOMING OF NESMA FPA FOR THE USE IN AGILE ENVIRONMENTS

While measuring the functional size of the project during the difference sprints, shortcomings where found in the NESMA method. Those shortcomings either make it difficult to do the estimation in an

efficient way, or make sure there are some strange measurements in the functional sizing done.

16.2.1 *Separation of Data and transactions*

The separation of data and transactions makes the estimation process more lengthy and difficult than it could be. It is a constant struggle of making sure the data and the transactions match each other. There is a mutual dependency: Transaction function should in principle only use DET and FTRs that are counted as well in the data functions. Data functions should not have data counted that is not part of a transaction function.

The data functions and their separation in external interface files and internal logical files is clear in definition as well as in how to apply it. Unfortunately in present times it is hard to make a distinction as software development has a more modular and integrative approach than it used to. Interaction with different web-services for example is hard to place in either EIFs or as input to and output from the system as was made clear in section 8.1. According to the standard there are clear rules regarding this, but it might happen that during development this changes. E.g. the architect decides to use some more components from another source than to actually develop it. This might lead to a different interpretation and to a different count. This is difficult and time-consuming, but when interpreted well by all the rules of the method, could be possible. After all, in fact there is less functionality developed when choosing an existing component. Furthermore it is a matter of viewpoint whether this should matter for the functionality delivered to the client.

Besides this, there is another rather odd behavior of this part of the count. The method has strict rules about what should be considered part of the same ILF or not. In these rules an ILF could be found so large that it will span different sprints. When looking at the case we see there that Health goal and interventions which are related to each other. In fact, when deleting a health goal, also the connected interventions are deleted. This means that this is one ILF according to NESMA. When developing however, we see that the Health goals are developed in sprint 1 and the interventions in Sprint 2. This was handled by counting the health goals as an IF developed in sprint 1 and in sprint 2, the enhancement of adding the interventions was added to this. However, when we would have also done this part in sprint 1 and not sprint 2, this would have counted as merely one count. This left us with the following values:

As compared to:

The amount of work in both situation should be about this same, as about the same work is done by the developers. This shows that this mechanism is in the basis not correct. The main problem is that data can be built up gradually over the course of the project and it should not take a lot more time to do so. Certainly not the amount of at least 30 hours shown above. With transactions this is different. It will not

Sprint	FP	EFP
1	10	10
2	5	11.3
Total	15	21.3

Table 24: The count of the Health goals with intervention spanning two sprints

Sprint	FP	EFP
1	15	15
2	0	0
Total	15	15

Table 25: The count of the health goals with intervention spanning one sprint

really happen for a transaction function to be done halfway through one sprint and halfway through another. Even when it would happen the same problem will occur. This will be to lesser extent because the maximum size of transaction function is considerably less than the size for a data function. Therefore the transactions are not a big problem, but the counting of the data is a problem, which should be solved somehow to guarantee an outcome of the FPA that is closer to reality.

16.2.2 *FPA Tables*

FPA tables are somewhat special in the method. These are containers for different data that is unrelated to the core data of the application, but it is necessary to be stored. These could be restrictions on values or the general knowledge in the application. An example of this is the 'F' or 'M' value for gender, aliases for any other identifier and other similar parts. This is manageable when estimating or doing a final count, but difficult when measuring progress over the project. In the case study the FPA tables were counted as being developed in the first sprint. This could be incorrect in nature. Parts of the FPA tables might have been developed later. The granularity is simply too coarse to estimate this in detail.

The FPA tables are in that respect a disadvantage. It makes the method more complex, making the analyst think about the data as a possibility to put it in FPA tables as well. It could create false rework if the development of the data in the FPA tables spans more than one sprint. This happens in the same way as was explained in the previous section.

16.2.3 *the way EFPA looks at enhancements and counts.*

There is something rather unusual in the way functionality is sized when looking at the increase in size versus the enhancement size. It could be that the functional size is not increasing but the enhancement size is. This is fine, when changing functionality work has been done, but the functional size might not be the same or even smaller. This is the aspect that makes the enhancement count so valuable. However the following can occur as well:

We add 1 DET to a functionality, which therefore gets a higher complexity, but the enhancement size is smaller than the increase in functional size.

This could happen when for example we have an ILF with an average complexity, where 1 or 2 DETs are added. This is a small enhancement, but might lead to the next level of complexity which results in a higher functional size for the product. This increase is not visible in the developed size for the same amount. In itself this is not a huge problem as we can imagine we would also have no increase in product size, yet we have an increase in developed size somewhere else in the project. However in principle this would lead us to an end-result where the difference between the added functionality and the enhancement size is less expressive than it should be.

See the below example:

- Having an ILF, with 2 Record Entity Types and 50 Data Entity Types (DETs) this is considered a ILF with average complexity (10 FP)
- One DET is added to this ILF.
- We have the following FP and EFP count:
  - New ILF: 2 RETs 51 DETs is an ILF with high complexity (15 FP) increase in functional size is 5 FP
  - Enhancement function point count:
  - $DET_{changed} = \frac{\triangle DET}{\prec DET}$ In this case is $\frac{1}{50}$
  - The impact factor for ILF and EIF can be calculated using the table as $I_{changed}\left(\frac{1}{50}\right) = 0.25$
  - $EFP_{changed} = 0.25 * 15 = 3.75$

  This shows us the the increase in product size is 5 FP while the increase in developed size is 3.75. This is true for up until 16 DET added. This is not a big difference, but it might occur a couple of times and therefore show undervalued information with regard to the rework done. The fact that the product size could turn out to be more than the developed size should be considered unwanted behavior of the method. This is inherent when using NESMA for enhancements.

17

SUGGESTIONS TO SOLVE SHORTCOMINGS.

In this section some suggestions will be made to solve the shortcomings that were found in the method or its application. It will be shown that using some strict guidelines for using the method could help.

17.1 GUIDELINES FOR USAGE

When making use of the proposed method, we have seen that, although it is possible to do these measurements ,in a normal business setting it might be ill-advised to do so. As it takes a lot of time to do the measurements, this might not be worth the investment of using it over expert judgment. This is not mutually exclusive. It is wise to perform several estimation for a project, even though in practice there is often not enough time or resources to do this.

If a company decides to use this method, the following guidelines should be taken into account to make sure the count takes less time and shows the most accurate information. It was explained that unfortunately these guidelines could not be taken into account while doing the actual counting in the case study.

17.1.1 *Role of the analyst*

The measures should have good knowledge about what is going on in the sprints, or what the project is about. That being said, it should not be done by any analyst who will personally be held responsible for the outcome of the count. In this respect this could be a requirements engineer or a tester. These roles are less dependent of the productivity measures or the real outcome of the project, than an solution / software architect, project manager or developer. Furthermore the analyst should be experienced, confident and interested or motivated in doing FPA.

17.1.2 *Time of measurement*

The measurement should be done at the moment the project takes place, after each sprint. For the last sprint, an estimation should be done on any changes still to be done, to cover situations like the ones in the case example where in the user acceptance test, quite some extra work is done. This would give an estimate of what is coming and make sure the project control can be done really well. Unfortunately the context did not allow us to do so.

17.1.3 *Validation of measurement*

The measurements should be validated well by a second opinion or through a final count of the delivered product. When differences occur in the interpretation of the results and the actual outcome of the project this should be made clear. Due to unavailability of a old version of the software (the software is an ongoing project even at the time of writing), a validation could not be done. This could be done if the time of the measurement is according to what was specified above. This would enhance the quality of the measurements done and hidden changes or things overlooked during the project can come to surface which will have to be handled.

17.2 A DIFFERENT WAY OF LOOKING AT ENHANCEMENTS

We could solve the issues with enhancements by changing the way we look at them. As introduced before:

1. Deletion counts as 0.4*FP deleted
2. There is a formula to calculate changes, based on the impact factor, which looks at the added, changed and deleted DET or RTE or FTR.
3. Added functionality is 1 FP = 1 EFP

This is in fact a mix-up between two different concepts: **Size** and **effort** as discussed before**.** When we count 0.4*FP as a deletion we get from deleting one functionality, this is based on effort (roughly that amount of hours are counted as a deletion also creates changes that have to be made in layout, other functions etc.) The same is true for the changes. But in general FPA is and should provide us a measurement to give us **size.** Even though effort should easily be derived from that, it should not measure it by default or be influenced like that. The same as that 100 meter of highway will take longer to build than 100 meter of path. But that does not mean that we will measure the path to be build as being only 10 meter. Looking at COSMIC we see that the guidelines COSMIC [18] tells us the same. The context itself should make the difference between what is the effort for these three different operations. In this way you keep a distinction between effort and size in the way that it should be done.

17.3 COSMIC FPP AS POSSIBLE ALTERNATIVE:

The same shortcomings as noticed above are part of discussions on the topic of FPA. COSMIC FPP was a method developed that measures functional size in a different way. The COSMIC method made sure:

- It does not include the use of FPA tables, or any other workaround for these aspects.

- Does not separate data with transactions as such, but instead focuses on transactions (data movements) which count heavier if they move more different data objects. |
- Does not have limitations with regard to sizing components.
- It handles changes in a stricter way considering size, as is discussed in 17.2.

Removing FPA tables, the difference between data functions and transactional functions and the limitations with regard to the size of a functions would change the NESMA method to such extent that it might be better to just start over. Fortunately this was already done for COSMIC. Therefore it is definitely worth considering the COSMIC for Agile projects approach COSMIC [18]. To assess COSMIC FPP as a worthy alternative cannot be established within the boundaries of this research. It might be worth investigating in the future, as will be pointed out in section 18.3, future research.

17.4 (semi)-automated measurements

The measurements done in this case were done using excel files with formulas that were constructed before (NESMA EFPA template) as well as during the counting (NESMA EFPA project count). As was stated before, the time taken to do these measurements was quite high, so the method might not be recommended as a viable business case. However, if these measurements can be done easily by the use of an automated program, this will be a more viable option.

There are software packaged developed for purpose that could be adapted to the method proposed. Total Metrics provides the 'Scope' and related products [1] being the most well known. Besides this there is 'FP modeler' tool [2] or Charismatek's function point workbench [3].

These are programs that aid in the counting of function points and should make it easier. Investigating their options is outside of the scope of this research and will be further elaborated upon in the chapter future research.

Besides this recently the OMG (Object Modeling Group), recently released a document [4]which gives pointers to automatic counting of function points for software projects. This is full automated counting. Giving the nature of NESMA FPA, this could pose more challenges than are handled in this report but it might give indications on how to reduce time investments in measurements. A solution in which the main part of the count is done automatically, but dilemmas are posed to the analyst, could perhaps be the optimal solution in this respect.

1 Source: http://www.totalmetrics.com where all available solutions can be found, retrieved 5/5/2013

2 Source: http://www.functionpointmodeler.com/ for more information, retrieved 5/5/2013

3 Source:http://www.charismatek.com/_public4/html/fpw_overview.htm for more information, retrieved 5/5/2013

4 To be found at:http://it-cisq.org/wp-content/uploads/2012/09/13-02-01-Automated-Function-Points.pdf

Even gamifaction structures might be used to create a solution that not only is a lot faster than to do this automatically, it also gives the analyst a lot more pleasure to do the counts than the current approach.

18 CONCLUSION

In the concluding chapter, a small overview is give of what was done in this research, what can be concluded and what cannot be concluded from the research done.

Secondly, a discussion will be started on the worth of the research done, its validity and what could have been done differently. Finally some suggestions will be given on further research that could be done in this area to improve the size measurement and its usage in Agile development.

18.1 CONCLUSION

In this research it was shown that NESMA FPA could potentially be used in agile environments, specifically with SCRUM. It depends on the purpose and the time available by the organizations using this method, if it is worth doing so. Using functional size as a baseline in Agile projects will give some great advantages in terms of quantifying functionality, estimations and scoping, but as a functional size measurement method NESMA might be too cumbersome to provide us with an efficient and convenient way of performing the analysis. Granted, when using NESMA in the way proposed in this research, handling high level requirements as well as changes, it has flaws in the method itself that were pointed out and written down here. Function points are in no way a holy grail when it comes to estimations. There is a lot more at play when translating user requirement to the effort and finally the cost of the project. Estimates are a way to reduce and document uncertainty. Function points will probably never provide the world with a way of doing extremely accurate estimates. However, it will make sure that each deviation from the estimate will have a clear explanation and a quantifiable measurement for it. It might be a nice starting point to relate the other aspects of productivity.

Using the enhancement function points in a different way it was shown that it is able to handle changes in the project during development. It is also shown in the case project that able change management is just as important. NAFPA only handles changes at the moment they are already implemented and the functionalities need to be changed. Not if the changes are made before or during development.

The opportunities here are to develop a company specific measurements and estimation system using a functional size measurement method that works with Agile. This research would give enough starting points for this, as most of the angles of this problem are covered. Even when it does not give an absolute answer to the question whether NESMA FPA should or should not be used in agile environ-

ments, it does show that if one would like to use it, how to do this. For reasons not to use it, it comes up with suggestions for improvement.

Therefore we argue that if the loose ends could be solved, it might provide software development companies with a helpful tool to measure their project in a uniform way. Looking into COSMIC might be a good alternative as well. A lot of potential problems with NESMA seem to be identified and solved by the COSMIC method. The lower granularity of the COSMIC method however, makes sure that no conclusive advise can be given in this respect. Using that method might take even longer to apply. As the method was tested on only one case, the results obtain are not in any way conclusive.

This research does show the value of having a real baseline for software project that is different from all the measurement that surround the SCRUM method. The development of a good baseline in this respect, still has to be developed and used well in a way that makes it easy to measure and use.

18.2 DISCUSSION

The research done shows us the importance of having a good measure for the concept of 'size' in software development. However it also shows what is still lacking in the method and what is still a challenge when you would have a measure like this. For estimation purposes NESMA seems to be too cumbersome and outdated but as a measure functional size does show some worth. There were shortcomings in this research that one could take into account in interpreting the research results. These will be discussed here.

First, the method constructed is not constructed by experts in the field, so some hurdles might have been overseen or some concepts could be used that are no longer relevant or are misunderstood to some point. Nor were the counts done, so it might be that an expert would have gotten a different result.

Second, As was discussed before, the nature of the project was not totally Agile and some parts of the method could not be tested in that respect. This is unfortunate, but given time restraints and lack of any other project this was inevitable. To handle this no hard claims could be made regarding some parts of the usage of this method, simply some indications were given on the presumed working and efficiency when it was not clear whether this would actually be the case if one ought to apply in the actual world. What could still be done is to find a real agile project were everything is done exactly according to the SCRUM method. It is doubtful that this could be the case. As with most methods available, a method is created given some assumptions on reality. Whenever the method is applied in a certain context, some alignment needs to be done to make sure the method fits. If not, some parts of the method might not be done to full extent or done in the wrong way. This is not different for SCRUM. The fact that the case project was not following every SCRUM principle to full extent is

therefore a reality that will surface again even when it is tested on another project.

Third, the requirements were not high level, but quite detailed. The assumption that high level estimated could be handled well by the method were therefore not tested. It is assumed that this is the case and it is illustrated by an example in section 6.1, 6.2 and 6.3. Actually testing it on a real project would have been a better way to validate these assumptions. Therefore no hard conclusions can be drawn regarding the handling of high level or uncertain requirements and especially the interplay between this and the handling of changes through the EFP method.

Fourth, the counts were not performed during the project so it could be that some functionalities were attributed to being developed in the first sprint but they were actually only part of the delivered functionality later in the process. Furthermore it is simply not enough to test anything on only one project. Therefore the results of this research should not be taken as conclusive findings. It would be enough to answer "is it possible to use NAFPA in Agile environments" is answered, because we managed to do it. Whether it is good idea is left open to the companies that would like to use it.

18.3 FUTURE RESEARCH

In this section, some pointers will be given towards future research that could improve this method. This could be done either by satisfying the same criteria we are trying to reach, optimization of the estimation and measurement process or tracking of the software development management process. This list is far from complete. In a critical area like estimation and scoping for software projects, there is always room for improvement. The list of possible areas to investigate is therefore almost endless. However, some aspects in this research were left open due to scope or time constraints. These will be discussed here. The loose ends were first presented in section 16.2 where the flaws of the method are presented. Some indications of solutions were given in section 17. These could be solved in any future research, especially with regard to the flaws in NESMA EFPA itself.

Investigating untested loose ends

Some other ideas were presented in this research which were not tested on their usefulness or applicability and therefore did not make it as flaws. The most important one is the sizing of different level of high level requirements in SCRUM projects. Especially the working of this combination of three different levels of sizing. Even more so the combination of this with the usage of enhancement function points as presented here. This will have to be tested on a project that has this kind of different level of requirements at the same. Preferably after some of the flaws presented earlier are already solved.

COSMIC as more suitable alternative

Using COSMIC Full Function Points as functional size measurement in Agile projects instead of NESMA could yield better results and prove to be more useful. This reasons for this were pointed out in section 17.3. This means that this research could be done on using the COSMIC functional size to achieve the same result. Rule [26]already pointed out this method. In his approach he takes more rigorous steps and denounces story points altogether. This is not necessary. The most important question is whether the COSMIC counting would be considered a lesser burden and less effort on the team than the NESMA method, next so solving the weaknesses pointed out before. From a first look the method seems easier in its usage, even though it has a finer granularity.

Improving estimations

As was pointed out before, when estimating hours, there are more variables at play than simply the functional size. For one the change percentage was discussed, which is not taken into the estimation, but came out during the project. The productivity, team coherence, quality of requirements are all important to the effort in time requested of the team. Therefore it is ill-advised to merely use functional size as the input to get the number of hours that should be billed as output. Research could be done in identifying and quantifying more factors that will help in estimate the effort in time. This could be combined in a simple factor that will translate the size to effort, which could help in the estimation process.

Using the method as Agile Maturity measuring tool

What we have seen in the research is that the number of hours per function point vastly increased over the sprints. This shows that a lot of work has been done to functionality delivered on previous sprints. While at the same time being a counter intuitive indicator for the productivity, it does show something useful. It shows that the team was not working in SCRUM as they should. Working in SCRUM means to deliver *working* software in iterations. In this case the software was delivered, but was not working correctly. This means that when the functional productivity goes down to such extent during the sprints, while the velocity should increase, it means there is reason to believe that the team is not working Agile or based on quality enough and that therefore the 'Agile Maturity' is low. Research in Agile Maturity is growing as a similar way of grading companies like CMMI, this could help put a quantified measure that shows that an Agile team delivers a vast amount of functionality **each** sprint.

Improving efficiency of measuring

As was discussed in 17.4 automated measurements or half-automated measurements using gamification might be the solution to decrease the time spend on counting the functional size. Further research on that area could improve the biggest burden the method has in present time. First and foremost and inventarisation could be made on the software currently available. If a team member of the project could spend half an hour resolving issues while counting, the main reason not to use this method does not exist anymore. Therefore further research in this area should be done to see how a functional size could be an important part of software development without it being a burden to any member of the team.

ACADEMIC BIBLIOGRAPHY

[1] Sharareh Afsharian, Marco Giacomobono, and Paola Inverardi. A framework for software project estimation based on cosmic, dsm and rework characterization. In *Proceedings of the 1st international workshop on Business impact of process improvements*, BiPi '08, pages 15–24, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-032-6. doi: 10.1145/1370837.1370842. URL http://doi.acm.org/10.1145/1370837.1370842.

[2] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1(1):57–94, December 1995. ISSN 1022-7091. doi: 10.1007/BF02249046. URL http://www.springerlink.com/content/y2386315010g7113/.

[3] Barry W. Boehm. Software Engineering Economics. *IEEE Transactions on Software Engineering*, SE-10(1):4–21, January 1984. ISSN 0098-5589. doi: 10.1109/TSE.1984.5010193. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5010193.

[4] R.S. Brwer, J.A. Dane, C.A. Moore, and P.M. Johnson. Empirically guided software effort guesstimation. *IEEE Software*, 17(6):51–56, 2000. ISSN 07407459. doi: 10.1109/52.895168. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=895168.

[5] L Buglione, J J Cuadrado-Gallego, and J A G De Mesa. Project Sizing and Estimating: A Case Study Using PSU, IFPUG and COSMIC. *Proceedings of the International Conferences on Software Process and Product Measurement*, 5338:1–16, 2008. ISSN 03029743. doi: 10.1007/978-3-540-89403-2. URL http://www.springerlink.com/index/f751712112181182.pdf.

[6] Lawrence Chung and Julio Cesar Prado Leite. *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, July 2009. ISBN 978-3-642-02462-7. doi: 10.1007/978-3-642-02463-4. URL http://dl.acm.org/citation.cfm?id=1577331.1577356.

[7] Mike Cohn. Techniques for Estimating. In *Agile estimating and Planning*, pages 49–60. Prentice Hall, 1 edition, 2005. ISBN 978-0131479418.

[8] Jean-marc Desharnais, Alain Abran, and Bugra Kocaturk. Using the COSMIC Method to Evaluate the Quality of the Documentation of Agile User Stories. In *2011 Joint Conf of 21st Int'l Workshop on Software Measurement and the 6th Int'l Conference on Soft-*

ware Process and Product Measurement, IWSM/Mensura 2011, Nara, Japan, November 3-4, 2011, pages 269–272, 2011. URL http://doi.ieeecomputersociety.org/10.1109/IWSM-MENSURA.2011.45.

[9] Cigdem Gencel and Onur Demirors. Functional size measurement revisited. *ACM Trans. Softw. Eng. Methodol.*, 17(3):15:1—-15:36, June 2008. ISSN 1049-331X. doi: 10.1145/1363102.1363106. URL http://doi.acm.org/10.1145/1363102.1363106.

[10] Cigdem Gencel and Charles Symons. From performance measurement to project estimating using COSMIC functional sizing. In *Software Measurement European Forum*, number 1, 2009. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:From+performance+measurement+to+project+estimating+using+COSMIC+functional+sizing#0.

[11] Mohamad Kassab, Olga Ormandjieva, Maya Daneva, and Alain Abran. Non-Functional Requirements Size Measurement Method (NFSM) with COSMIC-FFP. *Software Process and Product Measurement*, 4895:168–182, 2008. URL http://dx.doi.org/10.1007/978-3-540-85553-8_14.

[12] Roger S Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 6th edition, 2005. ISBN 0072496681.

[13] Jeff Sutherland, Anton Viktorov, Jack Blount, and Nikolai Puntikov. Distributed Scrum: Agile Project Management with Outsourced Development Teams. *2007 40th Annual Hawaii International Conference on System Sciences HICSS07*, 0:274a–274a, 2007. ISSN 15301605. doi: 10.1109/HICSS.2007.180. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4076936.

PROFESSIONAL BIBLIOGRAPHY

[14] A J Albrecht. Measuring Application Development Productivity. In I B M Press, editor, *Proceedings of the Joint SHAREGUIDEIBM Application Development Symposium*, volume 83, pages 83–92. IBM, IBM Cooperation, 1979. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Measuring+Application+Development+Productivity#0.

[15] CHARISMATEK. Function Point Analysis and Software Package Implementation Projects. URL http://www.charismatek.com.au/_public4/pdf/FPAPackages.pdf.

[16] Atul Chaturvedi, Ram Prasad Vadde, Rajeev Ranjan, and Mani Munikrishnan. Estimating the Size of Software Package Implementations using Package Points. 2011.

[17] COSMIC. *COSMIC Method Version 3.0.1, Measurement Manual.*

[18] COSMIC. *Guideline for the use of COSMIC FSM to manage Agile projects*. Number September. Public Domain, 3.0.1 edition, 2011. URL www.cosmicon.com.

[19] Martin Fowler and Jim Highsmith. The Agile Manifesto. *Software Development*, 9(August):28–35, 2001. ISSN 10708588. URL http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf.

[20] H S Van Heeringen. Changing from FPA to COSMIC A transition framework. *Software Measurement European Forum 2007*, 2007.

[21] Roberto Meli. Functional Metrics: Problems and Possible Solutions., 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.8456.

[22] E A Nelson. Management Handbook for the Estimation of Computer Programming Costs. *System Development Corp*, 1966.

[23] NESMA. The application of function point analysis in the early phases of the application life cycle, .

[24] NESMA. FUNCTION POINT ANALYSIS FOR SOFTWARE ENHANCEMENT Professional guide of the Netherlands Software Metrics Users Association, .

[25] Jolijn Onvlee and Rini van Solingen. Scrum en fuctiepunten: vrienden of vijanden? *Automatiserings Gids*, March 2012.

[26] Grant P G Rule. Sizing User Stories with the COSMIC FSM Method. 2010.

[27] Jeff Sutherland. The Scrum Guide. *Framework*, 2(July):17, 2011. URL http://www.scrum.org/scrumguides/.

[28] Jeff Sutherland, Guido Schoonheim, Eelco Rustenburg, and Maurits Rijk. Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams. *Agile 2008 Conference*, pages 339–344, 2008. doi: 10.1109/Agile.2008.92. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4599502.