

An Iterative Specification Game

Master Thesis

Thesis number 174 IK

Jeffrey Kwee

Information Science

s0313270

jeffrey@zoja.nl

Institute for Computing and Information Sciences (ICIS)

Radboud University Nijmegen

October 2012

Supervised by Dr. S.J.B.A. (Stijn) Hoppenbrouwers

Abstract

The objective of this thesis was to investigate, design and implement a framework of rules to construct a Serious Game that accommodates functional specification. This framework should also incorporate the concepts of iteration and functional decomposition.

By performing literature research, distilling existing theories about Serious Gaming, iteration, functional decomposition and the Chinese Boxes principle, and combining the findings, we constructed such a framework. This framework consists of several rules and guidelines which could be used to construct a Serious Game that guides a non-technical player through the process of functional decomposition. The purpose of the game is to construct a model that contributes to the functional specification of a system or process, while incorporating the concept of iteration.

We also implemented a prototype of the framework to showcase the theory. The Fun2Build game is a Serious Game which offers the player instruction about the concepts behind models, functional specification and functional decomposition. The game provides the player with a guided experience through the game and means to independently construct a model based on the framework's principles.

Acknowledgments

Nijmegen, September 2012

At last, my Master's thesis is finished. Before you lies a product that is the result of lots of hard work during the past year. I spent hour after hour writing theory, then re-writing it, programming the Serious Game, cursing when a setback caused me to re-write PHP code or if an entire segment of the thesis had to be scrapped, and celebrating when a chapter had been completed or some complex PHP function actually did what it had to do.

In the first place, I would like to thank my supervisor, Stijn Hoppenbrouwers, for providing me with lots of useful feedback and support during the writing of my thesis. Stijn directed and coached me towards a successful completion of the thesis, while also considering the time I put into my full-time job as an IT Consultant and the little spare time I had left to work on the thesis each week.

I also want to thank Patrick van Bommel, who opted to be the second reader of the thesis.

A big word of thanks is in place for my parents, David and Fatima, my girlfriend Stéfanie, my family and friends. Their continuous support carried me through the whole process and convinced me I could do it, time after time. Thank you so much guys!

With the completion of this thesis, I also officially say farewell to my student life. Ten wonderful (albeit a few too many ☺) years during which I didn't only work towards receiving my Bachelor's and Master's Degree in Information Sciences, but during which I also discovered the world, people and places, and which shaped me to become the man that I am today.

Jeffrey Kwee

Cohort 2003

Index

Abstract.....	2
Acknowledgments.....	3
Index.....	4
1. Introduction.....	8
1.1 Introduction to the digital society.....	8
1.2 History and emergence of information systems.....	9
1.3 Importance of good quality.....	10
1.4 Engineering an accurate information system.....	12
1.5 The research problem.....	17
1.5.1 The research goal.....	18
1.5.2 Thesis structure.....	18
2. Serious Gaming.....	20
2.1 Introduction to Serious Games.....	20
2.2 Benefits of Serious Gaming.....	21
2.3 History.....	23
2.3 Serious Game Characteristics.....	24
2.4 Genres and application.....	25
2.5 Evolution of the learning process.....	27
2.6 Preview of our Serious Game.....	30
3. Iteration.....	31
3.1 Introduction to Process Iteration.....	31
3.2 Waterfall process vs. Iterative process.....	33
3.3 Iteration benefits.....	34
3.4 Process iteration steps.....	34
3.5 Types of iteration.....	35
3.5.1 Planned iteration.....	35
3.5.2 Triggered iteration.....	36
3.5.3 Ad-hoc iteration.....	36

3.6 Preview of our iterative Serious Game.....	37
4. Functional decomposition and the Chinese Boxes principle	38
4.1 Functional Decomposition.....	38
4.2 Function composition	40
4.3 The Chinese Boxes principle.....	41
4.4 Chinese Boxes principle building blocks.....	42
4.5 Chinese Boxes principle sequence	45
4.5.1 Step 1: View the System function as a Black Box.....	45
4.5.2 Step 2: Turn the Black Box into a Glass Box	46
4.5.3 Step 3: Subfunction identification	46
4.5.4 Step 4: Transaction identification	48
4.5.5 Step 5: (Optional) Formal proof of the model	51
4.5.6 Step 6: Turn the Glass Box into the Black Box.....	52
4.5.7 Step 7: Doing it all over again.....	53
4.6 Final thoughts.....	55
5. The Orange Case	56
5.1 Introduction to The Orange Case	56
5.2 Orange Case mechanics.....	57
5.3 The Orange Case Game	58
6. Creating the Game	59
6.1 Research Goal and starting point	59
6.2 Preparation	60
6.3 Serious Game vs. 'standard tool'	62
6.4 Analysis and Design.....	64
6.4.1 Hoppenbrouwers, van Bommel and Järvinen's comparison.....	64
6.4.2 Technical Environment	66
6.4.3 Gaming environment.....	66
6.4.4 Goals, sub-goals and tasks	67
6.4.5 Game elements.....	67
6.4.6 Game mechanics and actions.....	70
6.4.7 Guidance and Feedback	71

6.4.8 Graphical User interface (GUI).....	72
6.4.9 Procedure.....	72
6.5 Implementation	74
7. Results.....	75
7.1 Getting started	75
7.2. Index.php - Welcome to the Fun2Build Game!.....	76
7.3. Tutorial	78
7.3.1 Tutorial.php - Tutorial - Functions and Transactions	78
7.3.2 Tutorial2.php - Tutorial – Your tools	80
7.3.3 Tutorial3.php - Tutorial – Who doesn't like coffee?	82
7.3.4 Tutorial4.php - Tutorial – Getting started	83
7.3.5 Tutorial5.php - Tutorial – Show Overview.....	84
7.3.6 Tutorial6.php - Tutorial – ADD Main function	85
7.3.7 Tutorial7.php - <Main Function Name>	86
7.3.8 Tutorial8.php - <Main Function Name> – ADD Function	87
7.3.9 Tutorial9.php - <Main Function Name> – Show Overview.....	88
7.3.10 Tutorial10.php - <Main Function Name> – ADD Transaction	89
7.3.11 Tutorial11.php - <Main Function Name> – Show Overview	91
7.3.12 Tutorial12.php - <Main Function Name> – ADD Transaction	92
7.3.13 Tutorial13.php - <Main Function Name> – Show Overview	94
7.3.14 Tutorial14.php - <Main Function Name> – REMOVE Transaction.....	95
7.3.15 Tutorial15.php - <Main Function Name> – Show Overview	96
7.3.16 Tutorial16.php - <Main Function Name> – REMOVE Function.....	97
7.3.17 Tutorial17.php - Tutorial – Getting started.....	98
7.4. Start modelling	99
7.4.1 startmodelling.php - Fun2Build - ADD Main function	99
7.4.2 showoverview.php - <Main Function Name> - Show Overview	100
7.4.3 addfunction.php - <Main Function Name> - ADD Function	103
7.4.4 removefunction.php - <Main Function Name> - REMOVE Function.....	103
7.4.5 addtransaction.php - <Main Function Name> - ADD Transaction	104
7.4.5 removetransaction.php - <Main Function Name> - REMOVE Transaction.....	105

8. Conclusion.....	106
8.1 Summary.....	106
8.2 Final Conclusion	108
8.3 Recommendations for future research.....	108
8.4 Final words.....	109
Sources	110
Appendix A – Source Codes.....	112

1. Introduction

There are many ways to design an accurate, working information system and many support tools that can help designing it. In this thesis, we focus on designing and implementing a Serious Game in which the functional specification for an information system can be modelled. This first chapter of the thesis introduces the context of the subject and why it is an interesting subject to research.

1.1 Introduction to the digital society

Current society is situated in a fast changing world, one which never remains the same for long and constantly evolves. The entire globe is bustling with activity, with people conducting business and connecting to each other everywhere throughout the planet. Since the middle of the 20th century, people have been empowered by use of technology. Daily life wouldn't be the same without information technology (IT). The usage of inter-global communications, computer networking and the Internet has changed our way of acting and thinking.

The technical advancement of society has affected companies and their products around the world. Where the majority of companies used to produce goods to be competitive in olden days, nowadays they shift more and more towards providing services and knowledge as a way to put them ahead of competition. Driven by increasing global demand of information, knowledge and services, we conduct business, provide services and live a modern life at speeds that were unthinkable a century ago.

This increase in service-oriented business can be accredited for a great part to the emergence of computers and the digital society we currently live in. Today, a person is subjected to more information in a day than a person in the middle ages in his entire life. Even though human beings have grown in intelligence and brain capacity since then, they need an aid in processing that huge stream of information.

A big role in aiding people in that fast evolving, digital life is played by *information systems* (IS), systems comprising of computers that are interconnected through computer- and telecommunications networks, and the way people use these IT resources to support (business) processes. Information systems enable users to provide and retrieve services all around the globe. The usage of these systems is inextricably interwoven with our daily course of affairs.

Definition

Information System

An information system is any combination of information technology and people's activities using that technology to support operations, management, and decision-making.

Note: Information systems engineering is often mistaken for computer engineering. However, it does not only encompass computer engineering, but also the strategic, managerial and operational activities surrounding computer engineering to make gathering, processing and using information possible.

Information systems can differ in size, price and computing power, ranging from a single computer that provides information about stock prices through the internet, to a supercomputer consisting of numerous clusters, spanning a whole building. All kinds of information systems exist, each tailored to the situation in which they will be used. The bottom line is that they provide us with means to gather lots of data and information and to analyse, compute and execute actions on that data. A well designed and implemented information system can do the work of hundreds or even thousands of people simultaneously, and provides us with computing power, interconnectivity, increased means of collaboration, easy means to share information and knowledge and automation of tedious activities. We truly live in the digital age, in a digital society.

1.2 History and emergence of information systems

In the past, information systems used to exist only within large corporations, taking over tasks and activities within business processes since the invention of the modern computer and the introduction of computers in the world of business in the 1960s. The effectiveness of these systems was evident and astounding at that time. Business processes were automated, along with other matters like customer relationship management and organisational communications. Work was smoothed out, optimized and digitally enhanced through use of information systems. Users of these information systems were able to work more efficient and more economical. These systems were precisely designed with the support of complex business processes in mind, and because they were tailored to suit exactly those business

processes, the cost to implement an information system within a company was very high. Only a select few people knew how to operate these machines, and it took a lot of work to prepare them for those tasks.

With years passing, computer manufacturing processes were modernized, which resulted in smaller, more powerful and, most important, less expensive computers. In the 1970s and 1980s, companies like Hewlett-Packard, Apple and IBM started to build computers for use at home, aiming to place a personal computer on every desk and in every household. Even though ownership of these machines was still costly, extra computing power was available for the people, offering lots of new possibilities. Instead of big, cumbersome machines like the mainframes used at work, people could access computing power through easy to use appliances at home. Small and medium-sized companies used these computers and turned them into information systems by using software that supported their business processes.

Nowadays, more and more and medium-sized businesses own and use information systems to plan and execute their business strategies. Even common households are able to use small scale information systems to interconnect computers to share data and computing power. The cost of ownership of a small information system, which can be as simple as a single computer connected to the Internet to gather information, is being lowered continually. This leads to very high penetration of information system use in daily life.

In 2008, information technology research and advisory firm Gartner Inc. suggested that there were more than one billion computer systems in use worldwide and that the growth of computer instalments was 12 per cent per year (Gartner08). This means by calculation that there will be two billion computer systems in 2014, of which many are used as information systems. As we can conclude, information systems are all around us, even if we aren't consciously aware of it.

1.3 Importance of good quality

Billions of information transactions are completed each day, analysed, simplified and supported for a great part by information systems. A lot of actions that used to require human physical interaction are replaced by digital counterparts, like the replacement of paper mail by electronic mail. We have become so used to having some actions automated by information systems, that we can't even imagine how some tasks were executed before they were automated.

Information systems process data and take actions behind our backs, automatically executing the instructions and rules that humans have specified for the system. We depend on their preciseness and reliability to carry out tasks and are used to the system functioning properly to such an extent that things in our daily life can go wrong if the information system we use

malfunctions. If we want to withdraw money at an ATM, we expect to get money from our bank account, in exactly the amount we have entered into the system. If a pilot lands a plane and consults his flight instruments for information, he expects the information the system displays is true and correct.

Herein lies the crux of this section: because we rely very much on the information systems around us to function correctly, we trust they always work like they're supposed to. Unfortunately, this is not always the case. The more information transactions are performed, the higher the chance of errors occurring and mistakes made.

Sometimes, an information system has been through all the stages of the systems engineering process and the end product doesn't meet all the stakeholders' requirements, or it isn't usable by the end-users for some reason, or the system doesn't work at all. This is an annoying situation which leads to loss of productivity, financial assets and time, but it can be a lot worse.

We depend on information systems to take care of some very important issues, issues that we severely depend on, sometimes even with our lives. Nuclear plants, missiles, traffic control systems, banking accounts, digital storage of important documents, hospital patient monitoring and life support are all examples of cases in which the correct functioning of the information system can have an effect on our health and well-being. In the past, several malfunctioning systems that were responsible for safety in critical circumstances have led to disasters, resulting in deaths and immense economic consequences. Most of these problems are caused by faulty information systems. Below are some examples to illustrate the catastrophic consequences of an erroneous information system:

Between 1985 and 1987, design and testing errors in the Therac-25 radiation therapy machine controlling system caused six people to receive an overdose of radiation and died (Leveson93 and Sarter97).

In 1992, the London Ambulance Service used a computer aided despatch system to automatically assign ambulances to nearby emergency situations. The badly designed system failed, resulting in ambulances arriving too late or at a wrong destination, which lead directly or indirectly to the loss of 46 lives (Mellor94 and Finkelstein96).

In 1996, the European Space Agency built a prototype of the Ariane 5 rocket, developed during ten years and costing 7 billion dollars to develop, which was destroyed in flight after less than a minute after take-off due to a fault in the software, which was reused from the Ariane 4 rocket (Lions96).

It is very clear that critical systems have to be engineered accurately, with a lot of expertise at hand and implemented with great precision, or the consequences will be very dire. Apart

from that, not only critical systems, but all information systems should be of high quality to fulfil their purpose as helper of mankind in our daily lives.

1.4 Engineering an accurate information system

So far, we can already conclude two things: there are a lot of information systems around the globe, and it is of great importance that those information systems are designed, implemented and maintained as accurate as possible. The problems that the information systems have to solve are becoming more complex and of greater magnitude, so the engineering of those systems has to be done in an orderly and quality-controlled manner to prevent system errors and engineering flaws.

There are many ways in which systems can be engineered, using different methods and mechanics, like eXtreme Programming, Scrum, the Rational Unified Process, Dynamic Systems Development Method, but most of the phases in systems engineering are roughly the same, which include:

Requirements engineering

A correct information system fulfils the correct requirements. Too often, an information system fails after delivery because it is the wrong system, a perfectly working system that solves different problems than it is intended for. To counter this, the process of requirements engineering is carried out, eliciting the actual requirements of the system from the stakeholders instead of features that might be nice to have, but don't solve the problem the system is intended for. The 'what must the system do' question is answered.

Analysis

During analysis, a selection of requirements is made that will be implemented in the system according to available time and budget. These requirements are then distilled into high level functions and operations that the system will provide. End-user information needs are also analysed, ensuring the system will provide the information and functions that are indispensable.

Design

The high level functions and operations are refined into detailed, tangible design, including screen layouts, business rules, user interface design and pseudo code, all of which are thoroughly documented for direct use in the implementation phase. The '*how will the system work*' question is answered in detail.

Implementation

The real code is written in this phase. Sometimes the entire code is written in one go, but in larger projects, the design is usually split into several parts that are written by separate teams.

Testing

The testing phase brings together all system components that are manufactured during implementation, and tests these components in a special testing environment. Errors and bugs in the code are inspected, and interoperability between the system components is thoroughly checked.

The problem with stakeholders

In this thesis, we focus on the design phase. Designing a system right is one of the most important aspects of system engineering. The system design is the basis on which the entire system is built. The system functions, internal and external interactions, user interfaces, usability, interoperability, even the colour of the buttons, all of those and more are decided upon in the design. Just as with requirements flaws, flaws in the design will be incorporated in the later phase of implementation, which leads to great loss of time and effort if they have to be corrected. In such manner, small errors in design can exponentially grow to become huge disasters later on in the process of engineering. If systems design was done by only professionals and expert designers, there would be no problem, because of the experience they usually have. However, there is a major element that must be kept in mind: involvement of stakeholders.

Definition

Stakeholder

A stakeholder is a person or organisation who or which is influenced by or can influence the activities surrounding an information system. In this context, they can be virtually anyone; an investor, a domain expert or an end-user, as long as they directly or indirectly benefit from the information system that is being built.

Note: On this assumption, in this thesis, by 'stakeholders', we specifically mean those stakeholders that are directly involved in the process of systems engineering and that are relevant to communicate with during that process.

Stakeholders have to be interviewed and cooperated with to ensure the information system will meet everyone's wishes and demands. In the end, a perfectly working system that solves the right problem but that nobody wants to use because it is not user-friendly or efficient, is a poorly designed information system. The problem with stakeholders is that they often, or better said, usually lack exact technical knowledge. During interviews or workshops, the information architect, who is in charge of designing an information system, tries to elicit the stakeholder's input in such way that it can be incorporated into the design. Because of their lack of technical knowledge, wrong input is often acquired, as stakeholders usually don't know what a Use Case is, or what usability testing is about.

Stakeholders often do not know how to solve the core problem that the information architect wants to solve or support with an information system and instead focus on what nice features

and digital trinkets they want to be implemented. This can be a problem, as the information system is built to support the end-users, who are also stakeholders.

The functional specification

It is clear that stakeholders of an information system fulfil two functions. Firstly, the system is built to support a process in their business or organization, and they are at the receiving end of the systems engineering process. Secondly, they are a source of information and input that can be used in engineering that system, so they contribute to the systems engineering process as well. This second point is of great importance, as the stakeholders usually have lots of information concerning the processes that the system will support. Complex design such as abstraction layers in software are too technical for stakeholders to participate in, but there are several subjects on which stakeholders can provide useful input, such as the functional specification.

Definition

Functional specification

A detailed description of what the system will look like and how it will behave. This is seen from an external perspective, which shows what external parties, like end-users, other computers and the organization in which the system is implemented will observe when the system is functional.

The functional specification is an aid in conceptual modelling, supporting the analysis and the design phases in systems engineering. It is a document that clearly describes the *technical structure* of a system and its components and its *functions*. It can be used as a reference document and to allow consistent communication between people involved in the process of systems engineering. The functional specification is made by the information architect in cooperation with the stakeholders. All parties work together to form an accurate functional specification which will help avoid inconsistencies in the system, as well as duplication of code, redundancy and to establish a shared understanding of the workings of the system.

The functional specification serves several purposes:

- The project team can discuss and reach an accord with the customer and stakeholders on exactly what will actually be built, based on their list of requirements.
- It gives instructions to system engineers on what to build.
- It serves as a basis for estimating what work has to be done.
- It serves as a point of synchronising workloads and activities for the whole project team.

The details that are documented in the functional specification are understandable for the information architect, the systems engineers and the other stakeholders. The functional specification contains information about the structure and the functions at an abstract level.

Actions such as “When button A is pressed, process A2 will commence and the dialog screen will close” are documented, but not the real code that is required to implement that action, which would require technical knowledge to understand. This keeps the functional specification clear and understandable for all parties and as such, it is very helpful in building the right system with the right functionalities to solve the right problem.

Operational structure

Even stakeholders who lack technical knowledge have knowledge about how the processes work that the system will support, and therefore are able to provide useful input in constructing the functional specification.

To communicate with stakeholders and colleagues and to transfer knowledge about the information system, the information architect can use models, textual or graphical representations of parts of the domain of interest. These models can be basic or complex, depending on the recipient’s level of technical knowledge. Modelling, the construction of these models, is done alongside the analysis and design phases in systems engineering. During analysis, existing business processes are analysed and documented, which serve as a basis on which the design of the intended solution in the form of an information system is based.

The operational structure of a functional specification is the description of all elements that are crucial for the main processes to be carried out. It gives an oversight of the several processes within the main process of an information system, its sub-processes and, for each (sub-)process, its required inputs and outputs. Furthermore, it shows how goods, assets and information are transformed in the processes and how they flow from one process to another. The operational structure is usually fabricated by making models of the structure with pen and paper, communicating about the models with peers, and then digitizing the models.

The operational structure of a functional specification can be constructed and visualised by all stakeholders who understand the main processes. These models are *declaratively specified*, which describes *what* has to happen, as opposing to imperatively specifying, which describes *how* it has to happen. This declaratively specifying aspect of the models enables stakeholders with little or no technical knowledge to contribute to building the model. Because the main processes that are being described are very abstract, like ‘Serving the customer’ or ‘Looking up information’, virtually any employee can fill them in.

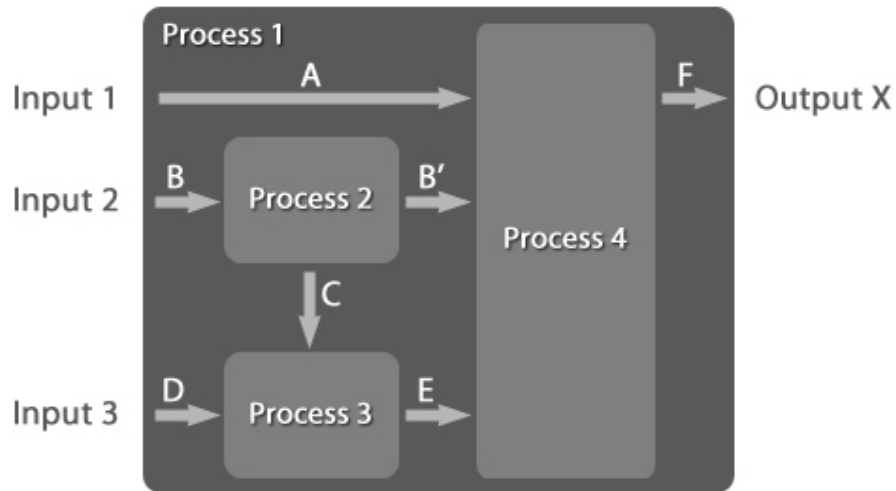


Figure 1.1. Example of an operational structure.

Figure 1.1 illustrates an example of a model of an operational structure. Inputs and outputs can be anything from goods to information and money. In this system, Input 1 is interpreted by Process 1 (which can be a specific function like “Retrieve information from several sources”, or even the system itself, which can be seen as a process as well) as input A for Process 4. Input 2 is interpreted as input B for Process 2, which has two outputs, B’, which serves as an input for Process 4, and C, which serves as an input for Process 3. Input 3 is interpreted as input D and together with C, Process 3 generates an output E, which serves as an input for Process 4. Process 4 at last, turns inputs A, B’ and E into output F, which ultimately is interpreted and exported as output X.

From an external view of this system, it seems that it consists of Process 1, which requires Inputs 1, 2 and 3 and delivers an Output X. From the internal view, a model of the operational structure, the inner workings of the system and its sub functions are made visible and structured.

To complete the functional specification, more details about the system are documented, like extensive declarative descriptions of the processes, screenshots of the intended design of the graphical user interface, lists of functional and non-functional requirements that are or are not realised and Use Cases. More details are documented or some are omitted, according to the project situation.

1.5 The research problem

An operational functional specification application

In this thesis, we are going to design and construct an application that helps to realise a part of the functional specification for an information system, specifically the operational structure. Functional specifications are processed as digital textual documents. The manufacturing of these documents is usually done in meetings between the information architect and the users and stakeholders. During these meetings, verbal communication between several parties and writing down the proceedings of the communication lead to the realization of the functional specification document. Until now, there aren't any freely available tools that help information architects and stakeholders to design the operational structure, which leaves room for research and realization and which will be focused upon in this thesis.

An operational functional specification Serious Game

To make the operational functional specification application work, it has to be fully functional and relatively easy to use for people with little or no technical knowledge, for reasons that are explained earlier on. Serious Gaming is a concept in which information and knowledge are transferred, real life skills are learned and actions are performed in a game-like environment, using game-like elements like a graphical interface, game rules and mechanics and a gaming world. The difference with regular games is that these games are tailored to specifically suit situations in which interactive training can be beneficial. Serious Games are an attractive way to let users undergo experiences with little guidance, while still keeping control of the experience and exerting influence on its outcome by using mechanics that can also be seen in games. The concept behind Serious Gaming and all its benefits will be discussed further on, in Chapter 2: Serious Gaming.

An iterative operational functional specification Serious Game...?

Iteration, the act of repeating a process until the desired outcome is achieved, is an important part of this thesis. Iteration is found in all kinds of processes in all kinds of business sectors. Iterative ways of design and implementation are widely used in agile systems and software development. Most applications that help build models work in a straight-on manner, going in a linear path from start to finish. By accommodating iteration in our application, we enable users to work in an incremental way, starting with a small operational structure of the functional specification, and iteratively building a model, being able to reflect and change things after every addition. The different kinds of iteration and how to handle them are discussed in Chapter 3: Iteration.

The Orange Case

Students of Computer Science and Information Science studies at the Radboud University in Nijmegen, the Netherlands, are taught the foundations of functional decomposition by playing a game called the Orange Case, a Serious Game which is an exercise in functional decomposition. The backstory of the Orange Case is "Your uncle in Spain has a lot of left-

over oranges which he can't sell on the Spanish market. You can have them for free and want to sell orange juice in Nijmegen, the Netherlands.”

The students then have to specify the needs for a system which is needed to achieve their goal. This is done by hand with paper and pen, but there is a need for a digital version of the game. The iterative specification game mentioned that we intend to research and build could be used for that purpose, with minor adjustments according to the story of the Orange Case. We will not do that in this thesis however, but propose it beforehand as a subject for future work and research.

1.5.1 The research goal

The subject is now narrowed down from a general view of information systems to a detailed one. To prevent scope creep in this thesis, we will narrow down our field of interest: *systems engineering => information systems design => functional specification => operational structure*. This operational part of the functional specification will be our main point of interest.

To conclude, in short, the research goal of this thesis is *the design and construction of an operational functional specification game*.

This will be done by achieving two sub-goals:

1. The analysis and design of a theoretical framework based on game-like elements to guide a player through a Serious Game in which an existing system is declaratively specified through functional decomposition. The guidance is provided in such a way so that iteration is facilitated.
2. The implementation of this framework in the form of a Serious Game prototype to demonstrate a basic showcase of the framework's rules and guidelines.

During the research that accompanies this thesis, we will discuss several ways of designing a Serious Game which enables users to build a model of the operational structure of an information system. We will then code and implement that Serious Game with accompanying game mechanics and a graphical user interface.

1.5.2 Thesis structure

Chapter 2: Serious Gaming

This chapter will provide a comprehensive theoretical background on the subject of Serious Gaming, a certain type of software which enables users to acquire information and new skills

through a game-like application. The application we intend to research, design and implement will be a Serious Game.

Chapter 3: Iteration

Chapter 3 is about the concept of iteration, the several different types of iteration and how to cope with it in business processes. It also explains how iteration can be used to build our Serious Game.

Chapter 4: Functional Decomposition and the Chinese Boxes principle

This chapter is the introduction of the concept of functional decomposition, a method to model complex problems by breaking it down into smaller, simpler problems. The chapter then illustrates the Chinese Boxes principle, a principle that combines the concepts of iteration and functional decomposition to help produce a functional specification.

Chapter 5: The Orange Case

In this chapter, the existing Orange Case is introduced, a procedure that teaches students of Information Science and Computer Science at the Radboud University in Nijmegen, the Netherlands, the principles of functional decomposition. The Orange Case is an example of a real life situation in which a Serious Game that we develop in this thesis can be used.

Chapter 6: Creating the Game

The actual design and implementation of our application is done in this chapter. Combining iteration and functional decomposition with concepts of Serious Gaming, we intend to devise a game-play mechanic that guides players through our Serious Game with as little as possible external guidance. A framework for the application will be fabricated, which can be used for both the Orange Case and a more general use of functional specification of systems.

Chapter 7: Results

The results of the design and implementation are documented. A walkthrough of our game is given, demonstrating the game mechanics and how the user is affected by them by showing screenshots of the game and providing textual explanation.

Chapter 8: Conclusion

The results are analysed, we draw our final conclusions concerning the subject, and the results are discussed with suggestions for future work and research.

2. Serious Gaming

Serious Gaming is a relatively new concept in the world of computing. However, the idea behind it, a game in which the main goal isn't entertainment, but functionality or transfer of knowledge, isn't new; indeed, it has been in existence for hundreds of years. Serious Gaming offers lots of benefits, including conveying serious matter in a playful way to the user and to motivate that user.

2.1 Introduction to Serious Games

Serious Gaming can be a powerful aid to help solve problems within the world of learning. Learning new skills and gaining new knowledge takes a lot of time, money and motivation. Serious Gaming is a relatively new concept, in which people are trained, business processes simulated and information and knowledge transferred through use of a game-oriented environment.

This game-oriented environment includes a virtual game world within an application, in which the user of the application (also called 'player') plays the main part. Through executing actions in the game world, which are bound to certain rules, the user undergoes an interactive experience of a learning process. Within the game world, there is interaction between the user and the environment, so the consequences of his or her actions are clear.

There are several different definitions for the term Serious Game, of which the definition of Mike Zyda (Zyda05) is mentioned most often:

Definition

Game

"A physical or mental contest, played according to specific rules, with the goal of amusing or rewarding the participant."

Video Game

"A mental contest, played with a computer according to certain rules for amusement, recreation, or winning a stake."

Serious Game

"A mental contest, played with a computer in accordance with specific rules that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives." - Mike Zyda, (Zyda05).

This three-step definition clearly illustrates the different layers within the concept of Serious Gaming; it is an entertaining, virtual computer contest that strives to learn the players new insights. This definition is only one of many however, and a paper version of a Serious Game is considered a Serious Game too, so it's not totally constrained to use of computers.

Because the current generation professionals work more and more with communications- and information technology, as stated in chapter 1, it is a logical development that learning methods also move towards the virtual world of computing. According to Garris et al., games can be effective tools for enhancing learning and understanding of complex subject matter (Garris02). Serious Gaming is a developing niche in the world of computer applications. It responds to the shift towards virtual learning methods by offering an attractive, involving, game-oriented environment that acts as a complex simulation with a serious message. The player is served a complex problem that he or she has to solve in an active manner.

The goals of Serious Gaming are to stimulate involvement in the subject, to stimulate cooperation in different fields of work and to transfer and retain information and knowledge. This is all done in a pleasant way, which causes the skills and knowledge that are learned in the Serious Game to be learned in less time and will be retained for longer periods of time. The idea is that though nobody wants to take a boring test, everybody wants to join a game show, while it's about the same matter.

2.2 Benefits of Serious Gaming

Serious Gaming provides various benefits, suiting different groups of people. Whether they're used by end-users, financiers, domain experts or the information architect, practice has proven many times that the experience that is presented in a Serious Game has a positive influence on the design of an information system.

Interaction

Serious Games provide an interactive experience to the user, usually in the form of training or a simulation. The user is given some instruction or practice, and then stimulated to take action and to act for themselves, instead of receiving step-by-step instruction. This form of

learning requires active participation and stimulates user involvement in the subject at matter, which leads to longer retention of knowledge and skills.

Motivation

Learning new skills or receiving information can be a tedious job. Even if the subject at matter is crucial to the everyday activities of a user, that user can still be unmotivated to develop knowledge about it. By offering graphically and mentally attractive applications and making the learning or training process fun to perform, the process shifts from a 'must do' activity to a 'play' activity. This can also be done by adding engaging goals and rules, adding challenges and mystery to the game and to make the user feel like he or she is in charge of the situation (Corti06). Serious Games can be used to both intrinsically and extrinsically motivate the user, which stimulates the user to take an active role in learning the subject's matter. Serious Games can be used to develop learners who are self-directed and self-motivated, who are interested in the activity itself and because they think achieving a positive outcome is important (Garris02).

Control

Serious Gaming enables companies to offer employees or clients a safe, controlled and repeatable manner to experience a virtual version of an experience that in practice can be dangerous and expensive (Verbraeck09). Because the games can be a simulation of any situation, situations which otherwise would require expensive instruments, like learning to fly an airplane, can take place in a seat with controls and a screen instead of a full-fledged airplane. Hazardous situations, such as practicing how to control a meltdown in a nuclear power station, can be simulated as well, training users how to act in a real situation, but in a safe, controlled situation.

Time reduction

By intensely involving users in a learning process or letting them learn new skills, Serious Games reduce the time that is required to acquire those new skills. Instead of constantly giving the users instructions on what to do in the form of commands, they are taught how to act in a certain situation so they are able to act on their own if that situation occurs again. They are then encouraged to perform those actions in a controlled environment. This 'learning by doing' approach makes the subject easier to understand and master than simply taking in information from a screen or book, which leads to time reduction.

Another way in which Serious Games lead to time reduction is the simulation of business processes. The virtual version of these processes can be sped up to see what the effect of an action or intervention is over a period of time, without having to wait in 'real' time.

Cost reduction

Costs can be reduced by Serious Games by the above mentioned time reduction. Less time spent training or learning means more time spent on productivity, which leads to lower costs.

Of course, the costs for developing a Serious Game have to be taken in account, but a well-designed game can recover those costs over time.

The simulation aspect of Serious Games also enable the users to practice with virtual versions of expensive hardware, facilities and instruments. This training establishes a strong mind-to-matter link so that after some training, the users will be able to interact with those instruments in real life as if they had trained with the real instruments all the time.

A third cost reduction aspect of Serious Games is the fact that digital matters can be distributed very easily through communications technology worldwide. A trainer in Sydney can train a domain modeller in India and an end-user in France at the same time, without the need to travel to their physical locations. This enables companies to outsource their training and learning facilities to countries where labour is cheaper, while being able to remotely keep control of the learning process.

2.3 History

Serious Games have been in use for quite some time in the world of computers and software. Records of Serious Games go back tens of years. In 1980, even before the personal computer had made its introduction at home, 'Battlezone' was published, a computer game for the Atari game computer, in which one was driving a 3d version of a battle tank. A year after, the game was published as the first Serious Game called 'The Bradley Trainer', which was used by the US Army to train its troops to operate the Bradley military vehicle (Stone05).

Through the years, commercial games were published with an accompanying software development kit (SDK), enabling users and third-parties to modify the games. A version of *Doom II* was used to train US Marines at the Marine Corps Modelling and Simulation Management Office in Quantico Base, Virginia. After the events of September 11th, 2001, *Tom Clancy's Rainbow Six*, was modified using maps and scenarios by the US Army to train troops to fight terrorists in urban terrain (Stone06).

Serious Games were mainly used by governments and armies to train their personnel, but commercial organizations such as Intel and IBM nowadays have discovered the power of a gripping game that can be fun, but also serve as a platform to promote services and products.

In 2002, the Serious Gaming Initiative (SGI) was presented by the Woodrow Wilson International Center for Scholars. This American initiative stimulates the development of games which address political and management issues. Since the establishment of the SGI, it has published lots of articles about the development of Serious Games and held workshops

to promote the awareness of Serious Games, which lead to the development of Serious Games for non-profit organizations, hospitals and high schools.

Though it might seem as a relatively new concept, the history of Serious Gaming goes back much further in time. Nobody knows how far exactly. Long ago, practices of Serious Gaming were already to be found, like the popular game Backgammon, the Japanese strategy game Go and the modern Settlers of Catan. These are all examples of games which taught people the principles of gambling, strategy and mercantilism.

2.3 Serious Game Characteristics

Serious Gaming is a general concept which can be implemented according to the situation. Virtually any simulation can be built with virtually any possibilities. Therefore, there aren't any specified rules to which a Serious Game designer has to accord while designing or implementing. Some simulations are so highly detailed that they aren't distinguishable from the hardware or process they simulate. In other occasions, only an abstract version of a process with minimal gaming elements is needed. Even so, there are some typical characteristics that are represented in most Serious Games.

The player

The player is the main executor of the game and is a virtual representation of the Serious Game user. He or she plays an active actor role in the game, interacts with the gaming environment and has the possibility to execute predefined actions. Sometimes the player is portrayed as a digital character on screen in the game called an avatar, to enhance immersion. Some Serious Games involve multiple players, who can compete or collaborate on a subject.

The game world

A unique gaming world in which a problem is presented to the player. The gaming world has definitions of gaming elements, like movable or editable objects, a set of predefined actions that the player can undertake and a set of game rules to which the player must keep.

The user interface

The user interface is a gaming element which enables the Serious Game user to interact with the player and the gaming world and to see what is going on in the game. It often provides basic statistics that the player can use to monitor his or her current status in the game and controlling elements to execute actions. The user interface can be customized to suit the situation, whether a full 3D representation of the game world is needed, or just a plain screen with some buttons and text. It often has performance indicators which indicate how well the player is playing the game. These indicators vary per type of game and can range from

financial performance to moral decision making, depending on what goal the Serious Game is trying to achieve.

Guidance and feedback

Apart from these gaming elements, guidance is also provided, usually in textual or graphical form. This guidance explains the whereabouts of the gaming world to the user of the Serious Game and informs him about the rules that govern that gaming world and what actions the user can take in the game through his virtual representation, the player.

Feedback is another important gaming element. It informs the user in textual, audible or graphical form of the consequences of his actions in the game, on which the user can base his next actions. Feedback can be given directly within the game through the user interface, or afterwards as an evaluation. This evaluation points out whether the user has learned something about the game, which the user then can use to apply to his loop of judgement, behaviour and feedback. After some iterations of this loop, the player is able to devise new strategies, which help him perform better over time. Feedback from the game can also be a tool for game developers to adjust the learning goals of the Serious Game or to fine-tune processes within the game.

2.4 Genres and application

Serious Games are used in different sectors, in all kinds of customized versions to suit the specific need that the Serious Game has to fulfil. We can distinguish several different genres within the concept Serious Games, which all serve their own purposes, for example:

- *Advergaming*, games with a primary focus on commercial offering of goods.
- *Edutainment*, games with a focus on learning skills or training the player.
- *Infotainment*, games with a focus on transferring information and knowledge to the player.
- *Therapeutic* games, games which are used as an alternative therapy for treatment of diseases or illnesses.
- *Propaganda*, games with a focus on transfer a certain thinking pattern or behaviour to the player.

This list is non-exhaustive, as Serious Games can be put to use in virtually any public or business sector, as long as the purpose is to teach skills and transfer information and knowledge to the user. Serious Games can contain multiple genres and serve multiple purposes. Most sectors in society use Serious Games to train and inform their employees. Following are some examples in practice.

While creating our Serious Game, we will focus on the Edutainment and Infotainment genres.

Educational sector

The applications of Serious Games within the educational sector are endless. Serious Games can be used as a supportive tool to teach subjects like languages, math and even physical education. Sports coaches use adapted versions of sports manager games like FIFA Manager (soccer) and NFL Head Coach (American Football) to enhance organisational skills, invent and visualise new tactics and to present them to their players using a simulation. Virtually any subject can be taught in an interactive way through a Serious Game.

Commercial sector

Big companies like Intel, Coca Cola and MacDonald's use Serious Games as a playful way to inform customers of their products. Intel put visitors of their IT Manager 3 website in the role of an IT Manager within a fictive company using Intel technology, and gave them responsibility of decision making and managing cash flows. Proper use of commercial Serious Games teaches people to work with virtual representations of existing goods as if they were using it in real life and can be beneficial for a company's public image.

Public sector

Companies within the public sector use Serious Games to cultivate knowledge and public awareness of their subject.

The Netherlands is a country that is worldwide renowned for their construction of structures to manage the flow and containment of large bodies of water. A group of Dutch companies led by Tygron has developed Watergame, a Serious Game that trains players in water management and offers them information about developing water management projects, thereby increasing their awareness of the subject and spreading knowledge (Tygron).

Medical sector

Companies in the medical sector use Serious Games to train medical professionals in specialised medical techniques and to teach them how to use medical products like vaccines and medicine.

TruSim, a division of the British Blitz Game Studios, has developed Interactive Triage Trainer, a three-dimensional virtual representation of a crisis situation in which medical professionals are trained how to prioritize treatment of victims of a biological or environmental disaster (Keegan08 and Trusim).

Military sector

Armies and military intelligence from around the world train their soldiers using virtual simulations of actual battlefield situations. This prepares them to execute procedures in stressful situations in areas they have never been to, in a controlled, fail-safe environment.

In 2005, America's Army, a Serious Game that was based on Epic's Unreal engine, was originally developed to allow young Americans to explore military career opportunities and to make them aware of the purpose of military actions. The game was a 3D online shooter,

developed by the Moves Institute for the US Army and made it possible for thousands of players to experience a day in the life of an army soldier. The game was used for recruitment and training and was considered one of the most successful online games at the moment (Stone06).

2.5 Evolution of the learning process

The success of Serious Gaming and the use of it is correlated to the change in learning and thinking processes of the current generation, who were exposed to technology during growing up. This so-called 'Gaming Generation' was described in 2001 by Marc Prensky in his influential book *Digital Game-Based Learning* (Prensky01), and grew up in a world in which media played a big role in daily life. Prensky states that of the group of people that were born after 1961, most didn't use an analogue telephone, haven't known a time in which music wasn't portable, and have never lived without being exposed to hundreds of thousands of visual impulses per day. This Gaming Generation currently makes up a big part of the professional world population. Members of this generation simply can't imagine a world without digital media, computers, computer games and the Internet. This new generation has undergone a total different collection of experiences than the previous generations, more specifically a *digital* collection of experiences. The changes in thinking patterns following the introductions of modern digital technologies and media have led to a variety of new demands and preferences, especially in the field of learning.

By solving mysteries and puzzles, building and maintaining cities, running corporations, building civilizations and conducting strategically fought wars, all through playing games, employees of corporations worldwide have become keen in taking in lots of information, processing multiple inputs at a time and perform complex mental tasks. Prensky states that there are ten important changes in the cognitive processes of the Games Generation, which all contribute to the fact that the Games Generation can process a larger stream of information and is more adapted to digital learning methods. These changes are described in short below.

Twitch speed vs. conventional speed

The Games Generation grew up with information that is fed at high speeds. Television channels which broadcast video clips and television shows with lots of images per second and computer games in which information is to be taken in quickly, processed and action is to be undertaken, have trained the current generation in processing more information at a time, and at a higher speed.

Parallel processing vs. linear processing

The use of multiple information agents at a time, like listening to music while browsing webpages, or the use of mobile phones while watching television, have led to a better

performance in multitasking of the Games Generation. The so-called 'tickers' at the bottom of the screen in business television programmes, make use of this; parallel to the usual news broadcast, extra information about global events or stock value is shown on screen.

Random access vs. step-by-step

The Games Generation was the first generation that experienced the usage of hypertext and hyperlinks, providing interactive links between multiple digital sources of information. By using these methods, the Games Generation has become used to acquiring information from multiple sources and in a less sequential manner. This has made the members of the generation aware of the fact that there are multiple ways to acquire information and enabled them to find and combine multiple, less structured sources information, requiring them to think in structures and patterns.

Graphics first vs. text first

The generations preceding the Games Generation were shown graphical illustrations as an addition to text and to help explain matter. In the current situation, it is the exact opposite; text is provided to make the subject that is first presented in graphical form, more tangible. The current generation has been exposed from childhood by television, video and computer games that show very expressive images in high quality, with no or little accompanying text. This has led to a growing preference for graphical material as a source for learning.

Connected vs. standalone

The Games Generation was raised with possibilities for global connection by email, bulletin boards, newsgroups, chat, multiplayer games and instant messaging. This resulted in the Games Generation finding new ways to acquire information and to solve problems, by connecting with others. For instance, instead of using direct communication like a phone call to propose a question, they post a question on an online forum, where it is read by thousands of people and at which a huge source of combined knowledge is available. This connectedness of the Games Generation has also taught them to work outside of their physical location and encouraged them to work in virtual teams, consisting of team members worldwide, who cooperate through digital communication channels.

Active vs. passive

Games Generation members are active learners. They have grown up developing a great curiosity for new things, and want to experiment much more than their previous generations, who usually read the manual for a new appliance thoroughly before trying it out. The Games Generation click a button without fear, press something, and execute something, simply to see the results of their actions. They play with software, try any possibility until they understand how it works. This leads to less support of passive situations like lectures and traditional board meetings, and they prefer active experiences like chat, posting messages on the internet and using digital learning tools. This keeps them busy in an active way, but they also feel more in control of what's going on.

Play vs. work

By using modern digital media, the Games Generation is used to see everything as a game. To them, gaming is work. The fact that those games are played in the real world and have bigger, more serious impact than the virtual games, doesn't matter to them. The achievement of goals, winning and beating competition are part of their ethics and processes. Prensky states that managers and trainers have to find a way to incorporate the playful attitude of the younger generation in the 'real' world of business.

Payoff vs. patience

One of the most important lessons that growing up with computer games has taught the Games Generation is that if you spend sufficient time on a game and master all skills, you shall be rewarded with a next level to play, a victory or a place on the list of high scores. Your actions in the game decide what you'll get, and what you'll get is worth the effort you've put into it. This development has caused an intolerance in the Games Generation for activities that reward insufficient according to their expectations. Why finish college when you can start a company in high school and become like Bill Gates, a Harvard university dropout, who turned out to become the richest man in the world? Members of the Games Generation expect excellent long-term rewards for their work, but good short-term rewards in the meanwhile to keep them interested.

Fantasy vs. reality

The Games Generation has been exposed to the fantasy and science fiction genres for a long time. Indeed, fantasy has always excited young people, but in the generation that grew up with computers, a virtual fantasy world is easier and more realistic to construct than ever. A communal fantasy world can be seen as a way to bind with others, like in massively multiplayer online role playing games like World of Warcraft, Lineage and Final Fantasy. These games have million players worldwide and a large online community, in which people learn to play and work together. Some technology companies like Google and Microsoft take this fact in account, which can be seen in working environments in which employees are placed in a more or less fantasy place like an alpine mountain cabin as brainstorm location and places where employees can play computer games when they take a break from work. Fantasy-based Serious Games are an accepted and inspiring method to train and guide employees.

Technology as friend vs. technology as foe

Generations preceding the Games Generation are mostly not used to modern technology. Growing up, it was in their thinking process to shun new things. This is caused mostly by the introduction of the new technology at a greater age than with the Games Generation, which has used technology since childhood. The Games Generation sees technology as a friend, one that is always available if there's a need for pleasure or to play and relax. This generation sees the accessibility of modern technology as a birth right and it is in this generation in which a generation switch occurs: parents ask their children for advice about using their own

expensive appliances. Trainers and managers can stimulate employees from older generations to use modern technology for purposes which suit their own generation, like webpages and computer models, or to let them work in a team which uses modern technology. This leads to more technological acceptance and awareness in older generations.

'Attitude'

In addition to all previously described changes in thinking processes relating to work and using modern media, according to Prensky, the Games Generation has a lot of 'attitude', a direct, often sarcastic view of the world. They point out faults in others directly and are less afraid to criticise peers than older generations. To communicate effectively with this generation, one has to be direct and hold no taboos.

All these cognitive changes, which are consequences of years of 'new media socialization', as Prensky describes it, have had a lot of effect on the way of learning and acquiring skills in the Games Generation. Though it's not the only available method, computer games and video games are common information structures which the Games Generation is acquainted with and are able to support the changed needs and demands in learning. This is one of the reasons that digital learning methods, and especially Serious Gaming, is being put to practice and is a thriving business.

2.6 Preview of our Serious Game

We conclude this chapter with a preview of our own Serious Game, of which the design and implementation will be elaborated upon in a later chapter, chapter 5.

The concept of Serious Games is very general; each implementation of the concept results in a different game, according to the end user's wishes and the purpose of the game. In light of Mike Zyda's definition of a Serious Game at the beginning of this chapter, we can determine that the Serious Game we will develop will be a mental contest, which makes use of specific rules to reach our communication objectives of providing education on the subject of functional specification and to provide training on that same subject.

Concerning the subject of this thesis, we will incorporate the mentioned aspects of interaction, motivation and control while designing the game mechanics. The game will be targeted at the educational sector and will focus on transfer of knowledge and information and 'on the fly' training. While the design and implementation of the game takes place in a later chapter, we can already ascertain that the game is going to use certain Serious Games characteristics to reach its educational goal, like an attractive looking graphical user interface, textual and graphical feedback on user input and predefined modelling or gaming elements, each with their own behaviour.

3. Iteration

Iteration is a concept which is defined by the Merriam-Webster dictionary as: “The action or a process of iterating or repeating: as [...] a procedure in which repetition of a sequence of operations yields results successively closer to a desired result.” *In other words, iteration consists of continuously repeating (parts of) a process until the desired end result is achieved. Iteration is part of, but not limited to several agile software development methods. In this chapter, iteration in processes and specifically, iteration within the context of information systems modelling and engineering is discussed.*

3.1 Introduction to Process Iteration

Iteration in practice can be explained by example of the technique that a sculptor uses to produce a marble sculpture. He starts out with a cube shaped chunk of marble and hacks and shaves away at the surface of the chunk on all sides, turning the angular surfaces into curves. After the first round of sculpting, he then inspects the sculpture to see the changes that he has made and to see what actions he has to perform next. With each following round of review and refinement, more and more details of the sculpture emerge as the sculptor hacks away pieces of marble, one piece at a time. After rounds and rounds of sculpting, he is finished; the sculpture has transformed from a rough-hewn chunk of marble into a beautiful, detailed masterpiece during each repetition, each *iteration* of the sculptor’s modification of the chunk of marble.

Iteration is a common concept which is applied in many processes in many industries, as it has many uses. Within the field of information system engineering, it forms a powerful concept for analysing, modifying and reviewing an existing artefact (a document or model) or object in a cyclical manner. This enables people to work on the artefact in small, iterative cycles and build an overall more complete artefact than using straight-forward production processes. This iterative way of developing software and information systems requires the modellers and developers to continuously review the artefact in a changed setting. This

change occurs in the modified artefact itself as well as the environment in which the previous version of the artefact was conceived, as the introduction of a modified artefact affects its direct environment.

While iteratively building an artefact, the modeller or developer views the situation from a different perspective with each iteration, adding or omitting functionality or functional structure until the desired goal, a complete and correctly functioning artefact is achieved. Iteration usually results in higher quality end products, because the requirements, design and implementation of the product are open to change and evolve over time.

In system development terms, iteration is often used in Agile information system development methods like Scrum, eXtreme Programming and the Rational Unified Process. These methods start out with a basic, but functioning version of the system. This system is carefully designed and implemented, but often does not yet include all desired functionality. The system is then run through multiple sequences of the whole or partial system development cycle, called *iterations*, to increasingly add complexity to the system (Jacobson92). These iterations result in a stable, working system that becomes increasingly more complete with each version or release (Pressman05).

An example of an iteration is depicted in figure 3.1, which illustrates an example in which the initial planning is formed and the product that is developed is continuously run through the phases of Planning, Requirements, Analysis and Design, Implementation, Test and Evaluation. When the product's requirements are met sufficiently, the iteration cycles stop and the product carries over into the Deployment phase. As is stated, each iteration results in an executable release.

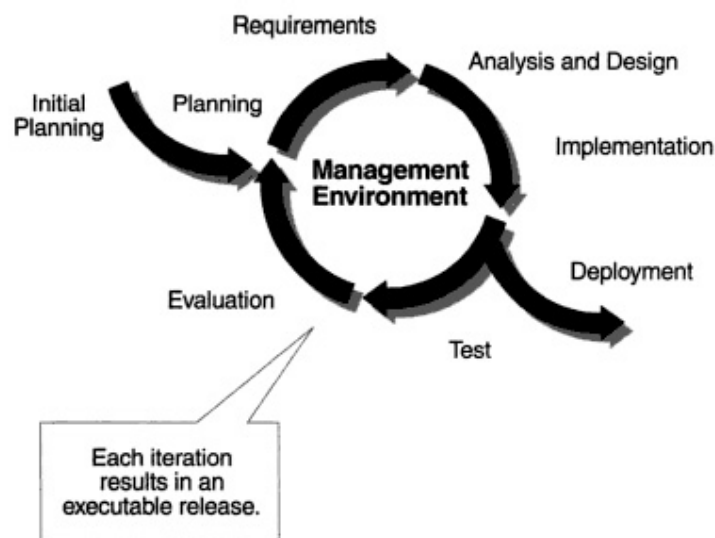


Figure 3.1. Iteration example as used in the Rational Unified Process in (Kruchten04).

3.2 Waterfall process vs. Iterative process

The conventional Waterfall approach is an old but still widely used system development method which completes each of the *requirements gathering, analysis, design, implementation, and testing* phases in succession with limited possibility of changing anything in a previous phase. In contrast to the serial succession of the phases in the Waterfall approach, iteration uses a little of all phases in a smaller development cycle. This provides much more feedback from the development process, which is illustrated in figure 3.2.

The Waterfall approach can be applied in systems development if the problem that the information system is meant to solve, is well-documented and well-planned in advance. The big disadvantage of the Waterfall approach is that it doesn't cope well with changes in the developing environment. If a project carries over to a later phase in the development process and changes in the environment demand that something in a previous phase has to be changed, the whole process has to be started from the beginning again; there is little flexibility. An example of this is the fact that stakeholders usually aren't clear about what they exactly want of the information system at the beginning of systems development. This can lead to faults in the system design and subsequent implementation, which are hard to correct in a late phase because of the linear approach; all phases previous to the one that has to be corrected, have to be run through again to check for inconsistencies.

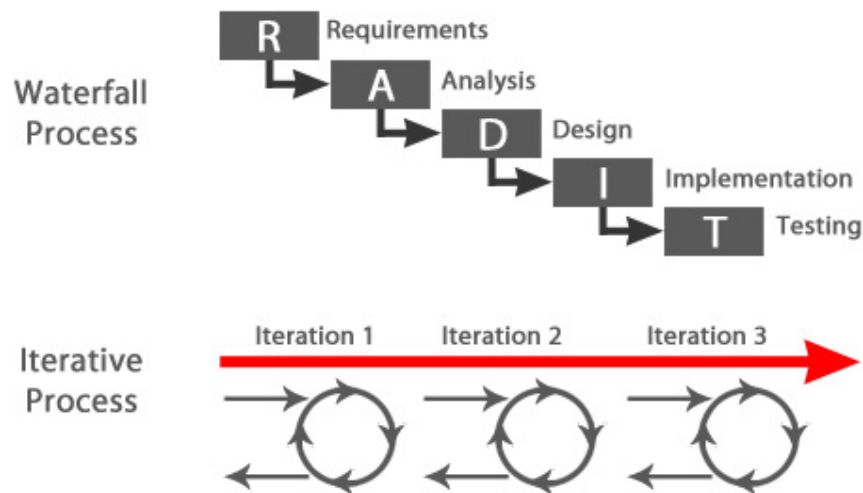


Figure 3.2. Waterfall process vs. Iteration

It is unrealistic that all documents and models are complete and error-free upon the first creation. Especially the system requirements are often not clear before the customers can put

their hands on a system prototype. Therefore, Waterfall-based projects tend to require very much rework at the very end (Kruchten00).

Another great disadvantage of the Waterfall approach is that no working version of the system is available to the stakeholders until the final phase of the development cycle is completed. Iteration, on the other hand, builds small, functional system components that can directly be tested, presented to the stakeholders and integrated into a whole, working information system.

3.3 Iteration benefits

Iteration does not necessarily mean less work and shorter schedules than the Waterfall approach, but it brings more predictability to the outcome and the schedule of the process. An additional benefit of working in iterations is that the development cycles are relatively easy to plan, control and execute because of their smaller size, which result in a more complete end result.

In (Kruchten04), Kruchten mentions even more benefits of iterative system development, among which are most important:

- Serious misunderstandings are made evident early in the lifecycle when it's possible to react to them.
- Iteration enables and encourages user feedback so as to elicit the system's real requirements.
- The development team is forced to focus on those issues that are most critical to the project and are shielded from those issues that distract them from the project's real risks.
- The project team can leverage lessons learned and therefore can continuously improve the process.
- Stakeholders in the project can be given concrete evidence of the project's status throughout its lifecycle.

3.4 Process iteration steps

The standard steps in an iterative process are quite simple, but very useful:

The initial iteration

The process starts with the inception phase of the process, the *initial iteration*. In this iteration, the initial artefact is created, which will grow incrementally with each iteration. The first iteration of a new process is usually the hardest. Many aspects in the process are

elaborated upon in theory, yet undiscovered in practice, such as getting used to using system software, learning to cope with problems, understanding a new domain et cetera.

The first iteration is about establishing a basis for future iterations, creating a basic architecture upon which the next iterations will be based. If the first iteration is too complex or too large, meaning there are too many operations in the iteration cycle or steps that require too many effort, you risk delaying the completion of the first iteration and reducing the total number of iterations planned in a fixed time schedule. It is of great importance that the functionality of the first iteration is reduced to a minimum when dealing with a new or unknown development environment and/or methods. The focus should be on integrating and tuning the environment and becoming proficient with the tools, before planning to build in lots of functionality in subsequent iterations.

An iteration

An *iteration* is the repetition of a process or sequence of operations. During each iteration, certain steps in the iterative process are repeated. Usually, the artefact that is focused upon will be examined, modified, and reviewed to see if the modification has contributed something useful to reaching the main goal. After reviewing, it can be decided upon to permanently implement the changes or to omit them if the results aren't sufficiently satisfying. The iteration cycles can be of different lengths, as the information architect can choose to add or omit an operation within an iteration cycle.

The ending step

An *ending step* stops the sequence of iterations. The conditions that have to be reached to initiate an ending step can be determined by many factors, for instance when the number of iterations reaches a specified number or a system value reaches a specified quantity. This ending step is optional and not necessarily present in all iterative development cycles, as some systems are continuously developed and their evolution never stops.

3.5 Types of iteration

Hoppenbrouwers et al. discussed in (Hoppen09) the idea that there are three types of iterations, namely planned, triggered and ad-hoc iterations, each occurring on different occasions.

3.5.1 Planned iteration

Planned iterations are scheduled into a process in advance. They start off with a planned initial step at a certain phase in a process or at a certain time, go through a certain amount of iterations and usually have a planned ending step with a clearly defined end result. Planned iterations are relatively easy to execute, because all changes to system variables during the

iterations have been taken into account in advance during the planning of this set of iteration cycles. In this way, the outcome of a process can be controlled with relative ease.

Planned iterations can also occur when a process is completed and a final round of reviewing is required in addition to the in-cycle reviews. Furthermore, a planned iteration can be used to refine a rough artefact into a tighter, more formally specified artefact with each iteration.

In a Serious Game, a planned iteration can be seen as reaching a new level after completing a set of objectives. Upon reaching this level, the tasks that are presented to the player are more or less the same, but with increased complexity or difficulty, making the game more challenging.

3.5.2 Triggered iteration

Triggered iterations occur when a specific event takes place that acts as a trigger for the iteration. They share some characteristics with planned iterations in that they usually occur when an artefact has to be modified because another, related artefact has changed, or when a specific date and time, or a specific system value is reached. The trigger that sparks the iteration can be either internal or external and doesn't have to be planned to occur at a certain moment in advance, which distinguishes it from a planned iteration.

From a Serious Game perspective, a triggered iteration can be seen as the reviewing of a list of tasks when a certain task is completed and adding a new task to that list. This moment can occur at several occasions during the process, but the handling of the triggered iteration remains the same.

3.5.3 Ad-hoc iteration

Ad hoc iterations are iterations that can occur 'out of the blue'. An information system should be able to handle ad-hoc iterations, which take place after a user has made a change that wasn't planned in the basic course of events. These ad hoc events are usually user instantiated, for instance if a user has to input some data of a certain type. If he or she changes his or her ideas about the subject at hand and tries to enter a totally different input than was anticipated, the system has to decide whether these ad-hoc changes are allowed according to its rules. If this is done in an iterative manner, the situation can be specified as an ad-hoc iteration. The initiative with ad-hoc iterations lies with the user in contrary to planned and triggered iterations, which are initiated by the system.

An example of ad-hoc iteration in a Serious Game is when a player is performing tasks belonging to one level, then realises that some step at a certain level or stage that has been completed before, has to be modified due to new insights. The player then suddenly switches from one level in the game to another, performing and finishing the modification. In doing so, the player makes use of iteration because the step has been completed before. After the

modification has been finished, the game should provide feedback and guidance about what steps to perform next.

The division of the term iteration into these three different types of iterations is not always clearly distinctive; iterations can be of multiple types, depending on their behaviour and characteristics.

3.6 Preview of our iterative Serious Game

Iteration is a versatile concept that can be used to add lots of added value to the design and implementation of a system. In light of this thesis' subject, we will use the concept of iteration to create a Serious Game that lets its players perform actions and tasks in an iterative way, providing them with several 'chances' to refine their model more and more with each iteration.

One of the foremost aspects of a Serious Game is the level of immersion and interaction that the game provides. Iteration will enable us to incorporate non-linear game-like mechanics into our Serious Game framework, like providing clear feedback on the player's actions and then enabling the user to use that feedback to repeat their tasks like before, but with new insights.

4. Functional decomposition and the Chinese Boxes principle

Functional decomposition *is a powerful technique that is used in modelling of information systems, but also in modelling in general. Functional decomposition enables the breakdown of a complex process or system into smaller parts, which are easier to comprehend, design and control. This chapter explains the concept of functional decomposition, a method to reverse it called function composition and the Chinese Boxes principle, a modelling principle that combines both functional specification and iteration.*

4.1 Functional Decomposition

The functional specification itself is a quite straightforward document. In contrary, the system that the functional specification is written for, is often complex, with lots of system components, and connections and cross-references between those components. Functional decomposition can help a modeller or developer to model all of these system parts in a clear and understandable manner.

The technique focuses on the functional relationships within an information system and divides them up into several, basic elements (or subfunctions) in such a way that the original function can be reconstructed from those elements by function composition. In systems specification, functional decomposition is done by performing three steps, which can be seen in figure 4.1:

1. Initially, the high-level processes of a system are identified, which can be considered functions with known inputs and outputs. The system architect has to identify all system critical functions to ensure all key processes are considered.

2. These high-level functions are broken down or split up into lower-level processes. Each high-level function, which represents a general process, can be broken down into smaller functions which represent more specific processes, which can be seen as changing the perspective from a more abstract level to a more concrete level. The resulting functions are also broken down when possible, until a single identifiable task is reached that cannot be broken down any further, or after a certain number of splits.
3. These single identifiable functions are then modelled as basic elements, building blocks which each have their own inputs, function and outputs. After this step, the complex system is modelled into small, controllable and manageable parts.

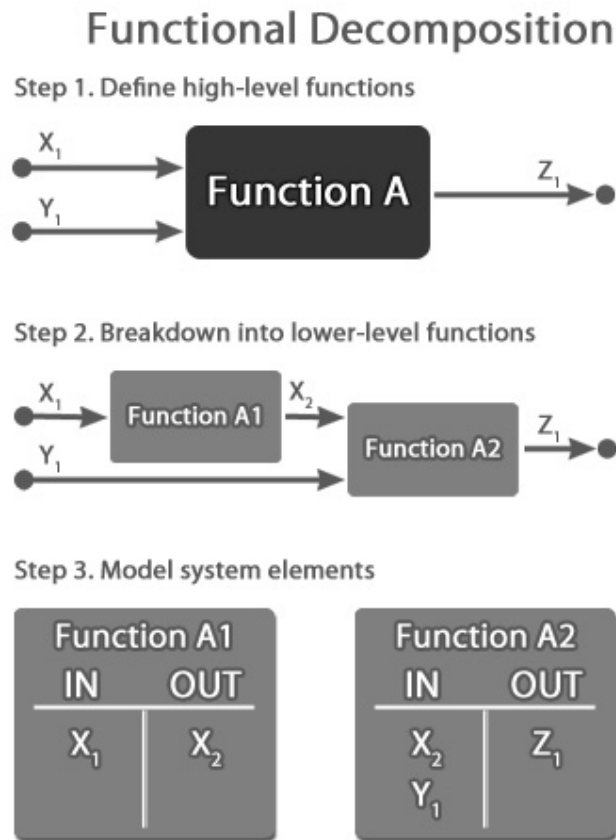


Figure 4.1: Example of Functional Decomposition

The main benefit of using functional decomposition is that a complex problem can be gradually broken down into easily understandable, implementable and maintainable elements in multiple steps, which eases the interpretation and analysis of the system. The parts themselves don't lose any functionality in their split up form, because they can be combined to form complex functions.

Furthermore, stakeholders with little to no technical information can help in the process of modelling using functional decomposition. The splitting of high-level functions and filling in

the properties of the basic elements can be done with help of the stakeholders by asking them comprehensible questions like “What does this step in the process require to be fulfilled?” and “Can this step be split into multiple steps?”.

Therefore, functional decomposition can be used by a variety of persons. People who lack technical knowledge can split and specify processes known to them in easy to understand, high level basic elements, whereas people with advanced domain knowledge can divide those basic elements further into specialized elements. In this way, stakeholders can interact with and consult each other during the modelling process.

4.2 Function composition

After the basic elements are individually modelled and constructed, a check can be executed to see if the elements, when grouped together to form a composed entity, logically produce the same outcome as the original higher-level function.

Function composition, which can function as such a check, is the concept of using the outcome of one function as the input for another function to create a new, composed function. The outcome of this composed function can then be used as input for another function, and so on, until the targeted level of composition or functionality is reached. When a high-level function is split up into several lower-level functions, the outcome of the composed function based on lower-level functions combined through function composition should give the same output of the high-level function they represent. If that fails, the high-level function is decomposed in an erroneous way, or the function composition is done in wrong order. The denotation of a function composition is done using the composition-symbol (\circ).

Example

In the example modelled in figure 4.1, a high-level function, function A, is split up into two lower level functions A1 and A2, each with their own inputs and outputs.

If we consider function A as a function $FA: (X, Y) \rightarrow Z$, which transforms its inputs X and Y into output Z, the constituent sub-functions A1 and A2 perform the actions $FA1: X_1 \rightarrow X_2$, and $FA2: (X_2, Y_1) \rightarrow Z_1$. In step 2 of the model, it is visible that the output X_2 of FA1 acts as an input X_2 for FA2. FA2 in turn transforms that input, and another input Y_1 , into its own output Z.

So, we have:

$$FA: (X, Y) \rightarrow Z$$

$$FA1: X_1 \rightarrow X_2 \quad FA2: (X_2, Y_1) \rightarrow Z_1$$

Function composition check

Is high-level function $FA(X, Y)$ equal to the function composition of low-level functions $FA1$ and $FA2$, in other words, $FA(X, Y) == FA1 \circ FA2$?

Using the inputs and outputs from the example in figure 3.3:

$$FA1: X_1 \rightarrow X_2 \quad FA2: (X_2, Y_1) \rightarrow Z_1$$

$$FA1 \circ FA2 = FA2(FA1(X_1), Y_1) \rightarrow Z_1$$

$$FA1 \circ FA2 = FA2(X_2, Y_1) \rightarrow Z_1$$

$$FA(X_1, Y_1) \rightarrow Z_1$$

$$FA(X_1, Y_1) == FA1 \circ FA2$$

We can see that the results of the composed function of $FA1$ and $FA2$ equals the results of function A , and therefore can say that the logical processing of the function composition is correct.

4.3 The Chinese Boxes principle

The Chinese Boxes principle is a method for hierarchical decomposition and combines the concepts behind functional decomposition and iteration. The principle helps specifying complex problems according to a few simple rules and steps.

An information system is constructed from lots of different parts, which exist at different levels of detail, which we will call levels of abstraction. Not only the information system's virtual services are constructed, but the operational or physical structure has to be constructed as well. This physical structure, consisting of connected computers and infrastructure, in turn exists of separate components like silicon chips and copper wire.

Specifying the system at a too high level yields little to no information about the system components. Specifying a system at a too low level gives us millions of loose parts to account for. The Chinese Boxes principle chooses to initially specify parts at a higher, more abstract

level, then iteratively dissects those parts into smaller parts, specifying the system at a more refined, lower level.

The eponymous Chinese Boxes represent a traditional style in Chinese design, which features a set of boxes of graduated size, each fitting inside the next larger box.

4.4 Chinese Boxes principle building blocks

The principle itself is a general concept and can be interpreted in many ways. In our interpretation of the Chinese Boxes principle, we use certain building blocks or components to model a system. We will now first elaborate upon those building blocks before we explain how the principle can be used for our research goal:

System context

The system context encompasses all components that are part of the system that is being specified. Components within the system context will be called **internal components**, whereas components out of system context, which exist beyond the *system boundary*, which delimits our system context, will be called **external components**.

Abstraction levels

Abstraction is the act of simplifying details. In our case, the system and its system context can be seen from different points of view. These range from a very abstract level, showing as little detail possible, to a very concrete level, providing lots of detail. These abstraction levels can be used to communicate at which level of 'deep-diving' into details one can see the model.

An example of a system view with a high level of abstraction is one in which the entire system is seen as one 'machine' which simply transforms its inputs into its outputs, whereas a system view with low level of abstraction describes the system in much detail, stating subsystems, internal process flows et cetera.

In our interpretation of the Chinese Boxes principle, we see system components at a certain **level of abstraction**, a certain level of detail. To be able to distinguish abstraction levels in an easy to communicate manner, we denote these abstraction levels with a number, starting at level 0 for *the system level*, the highest level of abstraction, and adding one level for each level deeper than the system level.

The **abstraction level denotation** is thus inverse to the level of abstraction, meaning that a higher abstraction level denotation represents more levels of 'deep-diving' into details is performed. In this fashion, in a parent-child relationship between system components, the parent is of abstraction level N and the child of abstraction level $(N+1)$.

A subsystem of the system itself would thus be seen at abstraction level 1. A subsystem of such a subsystem would be seen at abstraction level 2, and so on. It is also possible to see

parts the system from a combined view, in which system components of different abstraction levels are visible.

Functions

Functions are important components that are used in the Chinese Boxes principle which represent a system process or one of its subprocesses. Functions can be seen as miniature systems themselves, with only one simple operational function: to transform its **function inputs** into its **function outputs**. This means that a complete, fully operating function needs at least one function input and one function output. Furthermore, functions have a declarative function name, which briefly describes what the function as a whole does (like 'Transport Fruit to the store').

As explained earlier, a function can be functionally decomposed. The elementary actions that are the result of a functional decomposition from a function are called its **subfunctions**. Keeping the several levels of abstraction in mind, the subfunctions exist at a lower level of abstraction than the original function because they provide more detail, and therefore have a higher abstraction level denotation.

See figure 4.2 for an example in which the system function *Transport Fruit to the store*, a high level function at the system level, or abstraction level 0, is functionally decomposed into the subfunctions *Load the Fruit into the Truck*, *Drive the Truck to the store* and the *Unload Fruit from the Truck*, which exist at one lower level of abstraction, or abstraction level 1.

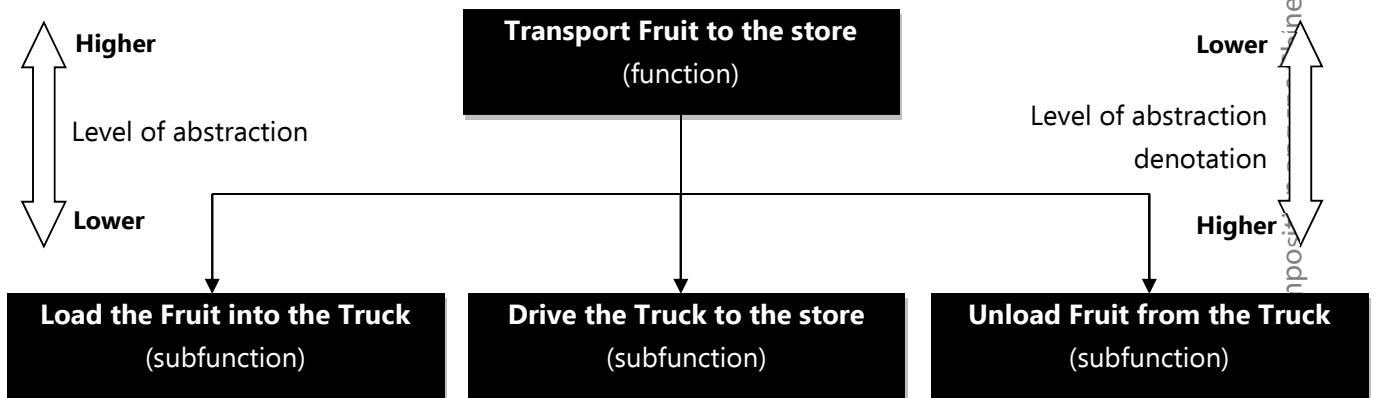


Figure 4.2: Example of subfunctions and level of abstraction.
View of abstraction levels 0 and 1.

If one of these subfunctions were to be functionally decomposed themselves into **sub-subfunctions** (for instance, by functionally decomposing the subfunction *Load the Fruit into the Truck* into sub-subfunctions *Open the Truck door* and *Loading the Fruit*), those sub-subfunctions would exist on two lower levels of abstraction, of 'deep-diving' into details, than the system function *Transport Fruit to the store*, meaning those sub-subfunctions would exist at abstraction level 2.

Inputs, outputs and transactions

A **transaction** represents something that is passed on between exactly two system components. Such a transaction can be virtually anything, ranging from an action that is performed by one component on the other, a physical object or virtual piece of information that is passed on from the supplying party to the receiving party of the transaction.

We call the supplying party the **supplier of the transaction** and the receiving party the **customer of the transaction**, or in short, the supplier and the customer. Each transaction is seen as unique in our case, so multiple transactions between the same supplier and the same customer are seen as individual transactions.

The aforementioned function inputs and function outputs are seen as transactions between two parties that are involved in executing the function. A function output can also act as a function input for another function, or the other way around. This is why we use the term *transaction* in addition to the terms function input and function output.

A system interacts with its direct environment, so the inputs at system level have to originate from an external source, that is, a source out of system context. The same goes for the receiving end of the system outputs.

Finally, the parties that represent suppliers or customers can be either **internal parties**, as part of the system, like a function or subfunction, or **external parties**, like a third party supplier that isn't part of the system itself. Following that logic, transactions can originate from either internal or external suppliers and target internal or external customers.

Now that the most important components have been described and introduced, we will explain how the Chinese Boxes principle can be used to perform functional decomposition in an iterative way.

4.5 Chinese Boxes principle sequence

The Chinese Boxes principle in its most simple form works as follows:

4.5.1 Step 1: View the System function as a Black Box

The main subject that will be functionally decomposed, whether it is a single artefact, an entire system or a process, is seen as the **system function**. The system function transforms its inputs, the *system inputs*, into its outputs, the *system outputs*. Some of these system inputs and outputs are known from the start, others will be identified during further stages of the Chinese Boxes principle. The system function exists at the system level of abstraction, or abstraction level 0.

In this step of the Chinese Boxes principle, nothing much happens, except that the system function is considered a **black box**, a single artefact that transforms its inputs into its outputs, but doesn't disclose any information about how the transformation takes place. In iteration terms, this step could be seen as the initial iteration.

Aside from the system function which is within system context, other concepts that can be seen in figure 4.3 are the external parties *Supplier 1* and *Customer 1* which are out of system context, the transaction or system input *System input 1*, the transaction or system output *System output 1* and the *system boundary*.

We can assume that the model of the system in this state is not complete yet. If the system were indeed such a simple system that it would be complete in this first step already, with the bare minimum of modelled system components, then there would be no use for modelling it with the tool that we will construct.

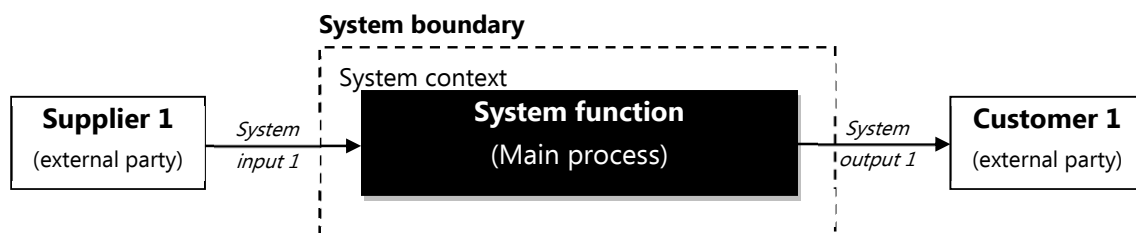


Figure 4.3: Overview: the system function as a Black Box.
View of abstraction level 0.

4.5.2 Step 2: Turn the Black Box into a Glass Box

In this step, the system function itself is viewed from a different perspective; the black box that is the system function, is 'opened up' and now considered a **glass box**, a transparent system which shows its inner structure at one deeper level of abstraction, as shown in figure 4.4.

In this step, system components from the highest level of abstraction are visible at abstraction level 0, in this case *System input 1*, the *System function* and *System output 1*. According to our framework, abstraction level 1 and its components should also be shown in this step because we are now one level of 'deep-diving' deeper. There aren't any system components modelled at that level yet, but this will take place in the next step of the Chinese Boxes principle.



Figure 4.4: Opened up: the system function as a Glass Box.
View of abstraction levels 0 and 1.

4.5.3 Step 3: Subfunction identification

The next step is to flesh out the model by filling in the inner workings of the system function, step by step. Through functional decomposition, several **subfunctions** will be added to the model in this step. These subfunctions combined will offer the same functionality of the system function through function composition, that is, transforming the existing system inputs into the existing system outputs.

The criterion for exactly what system component or collection of system components could effectively make up a subfunction depends on the person who is doing the modelling. For example, a system architect could group together different system components from his perspective than a financial director, but both models would provide useful information about the process that will be modelled.

To be able to keep a general oversight, a system usually is split up into a maximum of 7 ± 2 main parts or elements per abstraction level. Counting exceptions, the rule of thumb is that the maximum number of internal artefacts at each abstraction level doesn't exceed 9

elements. In a common system, there is a theoretical maximum of 9^n artefacts in total for n abstraction levels.

Breaking down the system function into subfunctions will be done by questioning which subfunctions or subsystems combined offer the same functionality as that system function.

Questions that can help identify subfunctions are:

"Which system critical operations have to be performed by the system to modify its system inputs into its system outputs and are therefore in any case necessary?"

"Is a logical division of system components that make up the system function possible?"

"Which system component is the component that will take in the system inputs from the supplier?"

"Which system component is the component that will transfer the system outputs to the customer?"

It is clear to see that to identify and model the subfunctions, communication between stakeholders and the modeller is of great importance to reach a mutual understanding of the correct subfunctions that will be used.

The rule of thumb is that every high level function, be it the system function or another function, when functionally decomposed into lower level functions, will result in at least two of those lower level functions. If, for instance, the system function could not be functionally decomposed into more than one subfunction, there would be no use for functional decomposition at all.

When a new subfunction is identified, it is added to the model. See figure 4.5 for an example in which Subfunction A and Subfunction B are added to the existing model. As stated before, these subfunctions are one level of 'deep-diving' into details deeper, and therefore exist at abstraction level 1.

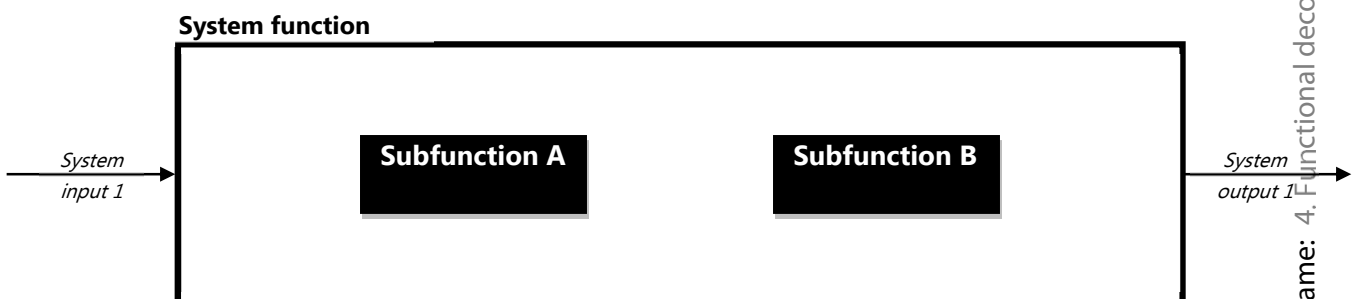


Figure 4.5: Adding subfunctions A and B
View of abstraction levels 0 and 1.

Function table

A simple way to denote a function and its transactions is by using a *function table*, a visual overview of all information belonging to a certain function or subfunction. An example of a function table is shown in figure 4.6.

The name of the function or subfunction is clearly visible, as are two columns below the name. When the function table is completely filled in, table cells in the column marked 'IN' will contain its incoming transactions and the cells in the column marked 'OUT' will contain its outgoing transactions.

Subfunction A	
IN	OUT

Figure 4.6: Example of a function table

It is quite unrealistic to assume that a model of an entire system will be produced in one go. The stakeholders and experts that participate in modelling the system will have to discuss and align ideas about which system component exists on which level of abstraction, which inputs and outputs will have to be added at what position in the entire process, et cetera. From this perspective, it is possible that some of the subfunctions at this level of abstraction can easily be specified already, whereas other subfunctions will emerge at a later moment in the modelling process.

Because the Chinese Boxes principle works iteratively, there will be many moments in the process during which additional subfunctions can be identified and modelled, one of the aforementioned benefits of using an iterative modelling approach.

4.5.4 Step 4: Transaction identification

After identifying a new subfunction, its function inputs and function outputs will have to be identified and specified. As stated before, a function requires at least one function input and one function output, because even functions like 'remove X' will have a function output like 'X removed'.

As in identifying subfunctions, a lot of communication between the stakeholders and the modeller has to take place to reach a mutual agreement on the model. Questions have to be asked concerning which subfunction requires which inputs or outputs, how many, who are the suppliers and the customers of that transaction, are two existing transactions the same, et cetera.

An example of the addition of new transactions is depicted in figure 4.7. Newly specified Subfunction A transforms its input, Transaction 1, into its output, Transaction 2. Subfunction B then transforms its input Transaction 2 into its output Transaction 3. Function tables for Subfunction A and Subfunction B are added, supplemented with their incoming and outgoing transactions.

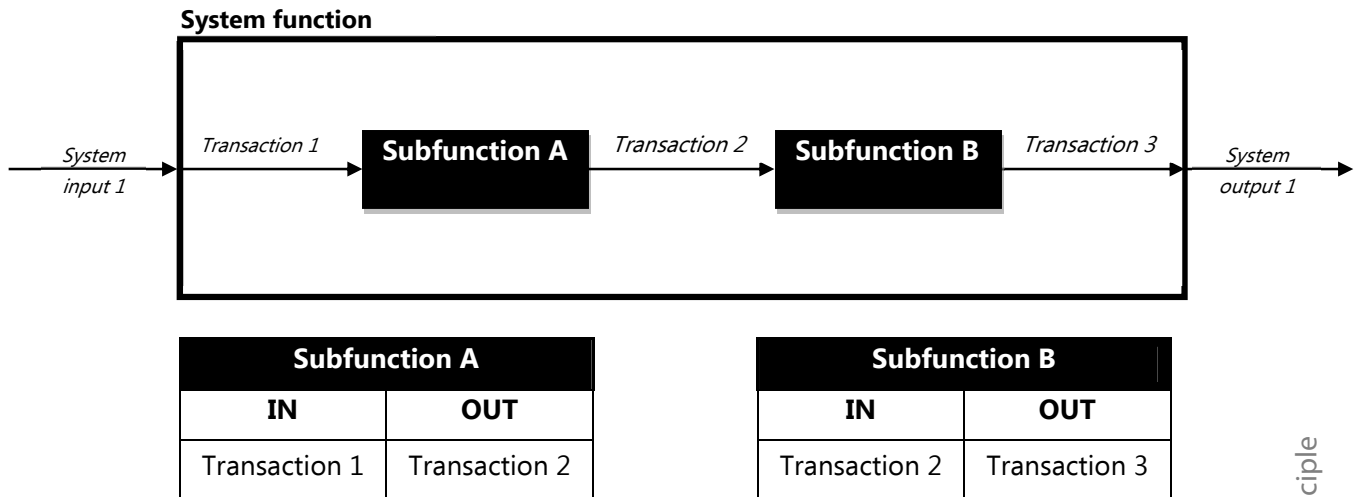


Figure 4.7: Adding Transactions 1, 2 and 3 to the model
View of abstraction levels 0 and 1.

Hierarchical passing of transactions

An important thing to notice is that inputs and outputs can be passed on hierarchically from higher levels of abstraction to lower levels of abstraction and vice versa. In light of the Chinese Boxes principle, the system function, the system as a whole, is seen as a Black Box. External inputs that are required for a function or subfunction which exists at a lower level than system level, still enter the system context at that system level. This is because from an external perspective at system level, the subfunction that requires the inputs isn't visible at all. The system level will take the external inputs in and passes it on to its subfunction. The same principle is valid for outputs from subfunctions which have an external customer. Therefore, a hierarchical passing of those inputs and outputs takes place.

Example:

System input 1 in figure 4.7 originates from an external supplier and is seen as an *external* transaction that is executed between the external supplier and the system function, as shown in figure 4.8 *top*. However, the moment the system function passes the system input down to Subfunction A, it becomes an *internal* transaction between the system function and the subfunction, as depicted in figure 4.8 *bottom*.

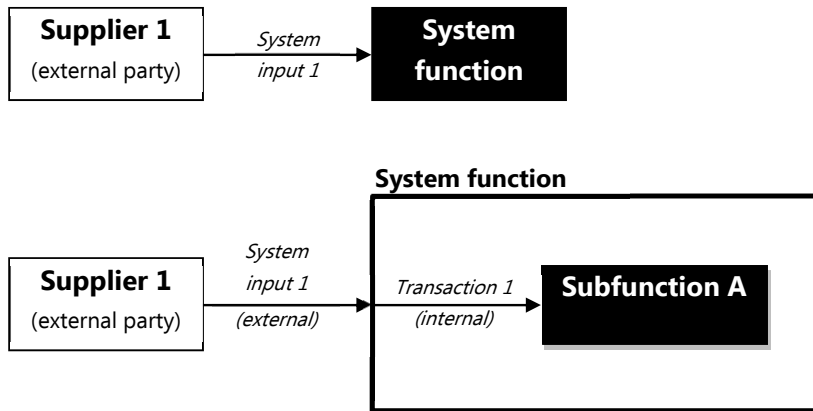


Figure 4.8: Hierarchical passing of transactions

Top: passing of System input 1 to System function

Bottom: external passing of System input 1 to System function and internal passing from System function to Subfunction A

Concerning abstraction levels in the situation of figure 4.7, external transactions System input 1 and System output 1 exist at system level, or abstraction level 0, and internal transactions Transaction 1, 2 and 3 exist at the level of Subfunctions A and B, or at abstraction level 1.

Identifying even more transactions

We proceed to identify new transactions and adding them to our model example. Let's say that after careful analysis, it seems that Subfunction A needs more than only Transaction 1 as an input. In this case it requires a new transaction, which we will call Transaction 4. It also seems that Subfunction B has another output, which we will call Transaction 5. These are added in figure 4.9.

As explained before, hierarchical passing of transactions takes place, because Transaction 4 enters and Transaction 5 exits the current abstraction level through the system level. For both internal transactions there has to be an external transaction that connects Transaction 4 and 5 to the external supplier and customer of the transactions. These are also added and called System input 2 and System output 2.

And a loose end

It is also possible that a function has a function output that isn't used as a function input somewhere else in the system. This kind of transactions is called a loose end.

For example, let's say that Subfunction A is discovered to have a second function output, called Transaction 6. If it isn't used as an input anywhere else at the same abstraction level, it becomes an output of the system level, because hierarchical passing takes place and it is seen to the external environment as an output of the system function. Consequently, Transaction 6 becomes System output 3. This is also depicted in figure 4.9.

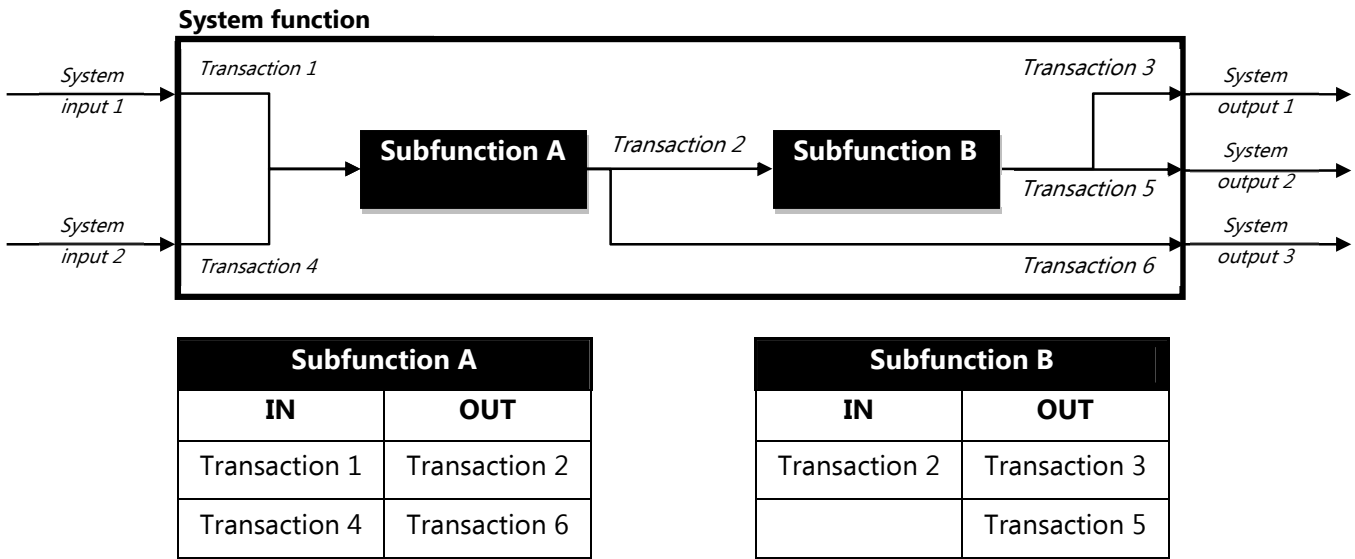


Figure 4.9: Adding even more transactions and a loose end
View at abstraction levels 0 and 1.

4.5.5 Step 5: (Optional) Formal proof of the model

As our model takes shape through functional decomposition, the need for a formal proof of the model may arise. This can be done by applying the aforementioned function composition, and formal logic, inductive and deductive reasoning to the model and its components, resulting in a formal proof that validates the functionality of each of the individual functional system components.

However, this formal proof is not in scope of this thesis, because a formal proof is only of added value for certain specific persons or roles within a process of system development. As it is a very technical and theoretical matter, it doesn't serve the general target audience, the non-technical stakeholders that want to build the right system that solves the right problem.

Formal proof failure

We will not dig further into the subject of applying formal logic to the model, but we do want to point out an important fact: if it turns out that the model cannot be formally proven or fails the formal proof checks, it might identify a gap between the current model and the end product, a correct and fully detailed model of the system function. Possible causes for a failed formal proof check can be that more subfunctions have to be added or some have to be removed, the functionality of some subfunctions has to be changed or transactions have to be added or removed.

Either way, the model has to be modified to reflect the correct functionality. To achieve this, there can be no loose ends in the model, meaning that each transaction has an identified supplier and customer and each subfunction has at least one input and one output. This modification can be done immediately after the identification of a gap, or at a later moment in the process, when the iteration cycle goes through this step again.

It is also important to note that while a model constructed using the Chinese Boxes principle always has to pass a formal proof check to be fully complete, it is also fully possible that it is the *intention* of the modeller to construct a version of the model that isn't functionally fully complete yet, and that it is planned to be fixed in a later iteration.

4.5.6 Step 6: Turn the Glass Box into the Black Box

In the first step of the Chinese Boxes principle, we viewed the system function as a Black Box, an artefact that didn't show its inner workings and processes. By 'opening up' the Black Box into a Glass Box and performing the subsequent steps of the Chinese Boxes principle, we identified and modelled the system's subfunctions, the transactions and their dependencies. We also identified a new external supplier to and a new external customer of the system function. Step by step, we have enriched the initial Black Box model.

This step is about 'closing' the Glass Box, which will result in the Black Box view of the system function again. To do this, we revert the model view back to only show the system components that exist at system level, or abstraction level 0. We retain all information and data that we gained at the Glass Box level, so we can always use it as reference documentation for the system function's inner workings.

Figure 4.10 shows the new Black Box version of the system function. This new Black Box view of the system function differs a great deal from the initial version of the Black Box in step 1 of the Chinese Boxes principle. The new Black Box has a 'hidden' information layer in the form of the information and data at the Glass Box level, describing the functionally decomposed system components that make up the system function at one level of 'deep-diving' deeper, and the logical processing of all system inputs into system outputs by the system components at that level. If some information about that level is needed, all that needs to be done is to turn the Black Box into the Glass Box again and all details of the subfunctions at abstraction level 1 are shown again.

At system level itself, we also added a newly defined transaction that led to the definition of a new system input and external supplier, two newly defined transactions that led to the definition of new system outputs and a new external customers, and a function table which describes the logical processing of the system inputs into the system outputs.

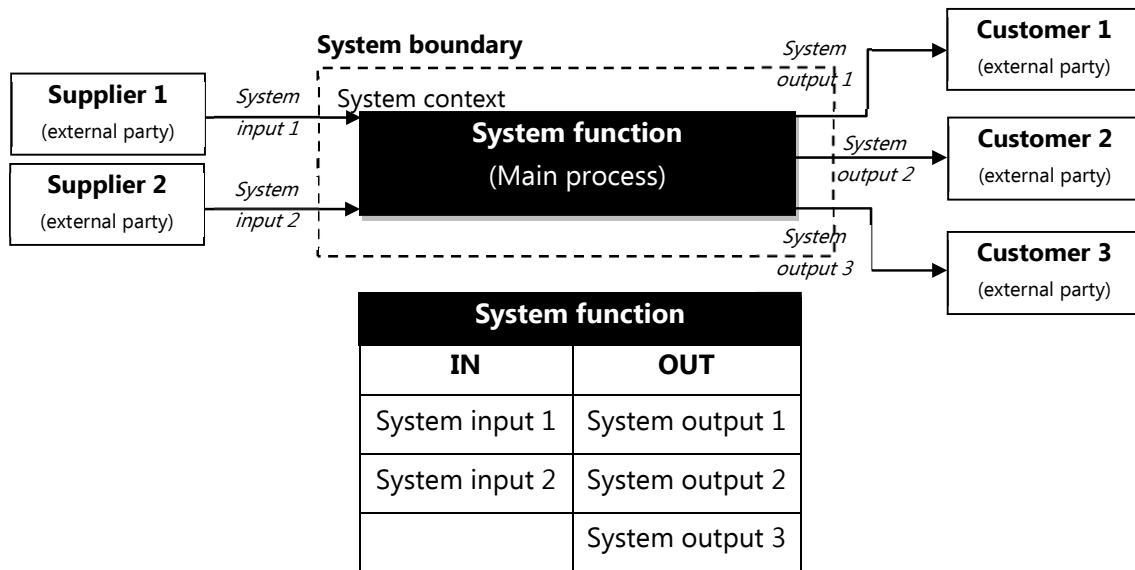


Figure 4.10: New overview of the system function as a Black Box and its function table
View at abstraction level 0.

After we turn the system Glass Box back into the system Black Box, we have completed the functional decomposition part of the standard sequence of the Chinese Boxes principle. Next up is the concept of iteration.

4.5.7 Step 7: Doing it all over again

As mentioned before, it is very much possible that the model that was created in steps 1 to 6 of the Chinese Boxes principle of iterative functional decomposition is not entirely complete yet, as the modelling of a system usually requires several rounds of modelling and review and feedback sessions in order to improve the model. In this step, the modeller will iterate through steps 1 to 6 again to navigate the model towards more completeness.

Desired number of abstraction levels

An important aspect to think about when using the Chinese Boxes principle is that of the *desired number of abstraction levels*. In the previous explanation of the principle, we only performed 'deep-diving' of one level deep, that of going from abstraction level 0, the system level, to abstraction level 1, the subfunction level.

Keep in mind that these subfunctions are still Black Boxes themselves too. A function table for a subfunction can help to understand and make explicit *which* function inputs get transformed by the subfunction into which function outputs, to help perform the high level system function, but it doesn't say anything about *how* the subfunction itself works internally.

These subfunctions themselves can be turned into Glass Boxes and functionally decomposed as well, resulting in system components of abstraction level 2, or sub-subfunctions. These sub-subfunctions can be modelled into the model too, and also require correct functional

composition checks before their 'parents' at the subfunction level can be correctly functionally composed into the system function. An example of this is given by figure 4.11.

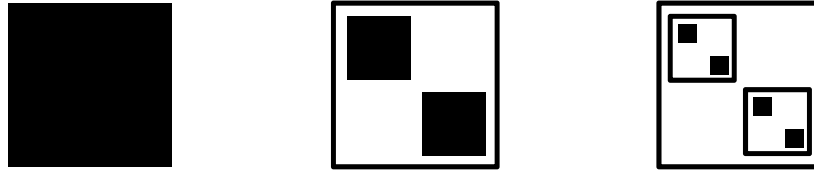


Figure 4.11: *Left*: System function Black Box, view at abstraction level 0, *Middle*: System function Glass Box, subfunctions Black Boxes, view at abstraction levels 0 and 1, *Right*: System function and subfunctions Glass Boxes, sub-subfunctions Black Boxes, View at abstraction levels 0, 1 and 2.

Iteration variations

Each system is different, and different situations require different methods to completely model a system through functional decomposition. The iteration steps that the modeller will follow after performing the first six steps will differ per system as well, so we cannot provide a step-by-step instruction of what iteration steps or sequences have to be performed in this step. We would like to provide some guidelines for the iterations, however.

During the first run of steps 1 to 6, it is possible that iteration takes place already. For instance, imagine a system function that is modelled as a Black Box, which is then turned into a Glass Box, and subfunctions are identified and added to the model. In the following step, identifying transactions, it is quite likely that while thinking about inputs and outputs, a new subfunction comes to light. It is entirely possible within the Chinese Boxes principle to then iterate back to the subfunction identifying step, add a new subfunction, add transactions and then perform an optional formal check before closing the Glass Box again.

The Chinese Boxes principle enables *full iteration cycles*, in which steps 1 to 6 (with step 5, formal proof of the model, as an optional step) have to be performed a first time to set a 'baseline' model (an initial iteration), after which it is refined step by step by performing iteration cycles of steps 1 to 6 over and over again to come to an incrementally complete model by each iteration. After performing steps 1 to 6 of the Chinese Boxes principle, the modellers are free to repeat the cycle over and over again in an iterative fashion, adding or removing subfunctions or transactions each cycle.

Because the principle is very flexible, it also enables *iteration sub cycles*, which don't span the entire process of performing steps 1 to 6, but only the steps that have to be performed. An example of this is a model which is the result of performing steps 1 to 6, which then appears to be lacking only one extra transaction from the last subfunction to a supplier. The next iteration could consist of only performing four steps: turning the system function into a Glass

Box, adding the transaction, performing an optional formal function composition check and then turning the system function into a Black Box again.

4.6 Final thoughts

We have now modelled an entire system, initially specified in rough, abstract terms, the single entire system Black Box. The Black Box is refined and becomes more detailed by opening it and turning it into a Glass Box, revealing smaller boxes inside, which we call subfunctions. By gradually adding details to the model and going through a number of iterative cycles, we deliver a detailed model which can be used to document the operational model of a functional specification of a system.

The framework we will setup in the next chapters will enable a modeller to specify the operational structure at a high level of abstraction and then iteratively break it down to smaller pieces until the desired level of detail is reached. This operational structure can be incorporated in the functional specification for an existing system, or one that has not yet been built. Working with this framework will include iterative specification cycles and enables stakeholders with no or little technical knowledge to contribute to the model.

5. The Orange Case

Students at the Faculty of Computer Science and Information Science at the Radboud University in Nijmegen, the Netherlands, learn the fundamentals of Computer Science and its key concepts through all kinds of learning methods. Some of these methods are more practically oriented, like programming, some are more theoretical like formal logic. One of the key concepts that is taught is the concept of functional decomposition, which is taught by taking part in 'de Sinaasappelcasus', or in English, 'the Orange Case'.

5.1 Introduction to The Orange Case

The Orange Case is a case built around a story, which incorporates workflow modelling and the Chinese Box principle, which relies on functional decomposition. The background story is as follows:

"Your uncle in Spain has an orange orchard. Each year, lots of oranges are harvested and taken to the Spanish fruit markets, where they are sold. Not every orange is suitable to be sold, however, as during harvest and logistics, some oranges get bruised or even worse damaged. Your uncle, who thinks of you as one of his favourite relatives, makes you an offer: You can have those bruised oranges for free, so you can sell the orange juice. You accept his offer and start making plans to sell the orange juice in your hometown, Nijmegen, in the Netherlands. And so the story begins..."

This story is told to the new students, who then have to come up with an idea to transport those oranges to the Netherlands from Spain, a distance of about 1600 kilometres. However, every thinkable solution can be used, whether it concerns a realistic, feasible and commonly used method of extracting the juice from the oranges in Spain, freezing the juice, transporting it to the Netherlands and once arrived, add water to make juice from concentrated juice, to absurd, out-of-the-box ideas like laying a pipeline from Spain through France to the Netherlands and pumping the juice through that.

The objective of this exercise is to come up with a highly original idea that has to be worked out in detail. Besides developing the idea, the students have to describe several other things:

Operational specification

This describes in natural language what the entire system has to do. The result will be a finished product that arrives at the right destination.

'Recipe'

The recipe describes which steps in the production process have to be passed to deliver one batch of the final product at the final destination.

Functional network

A diagram which shows which services are needed to execute the recipe and the congruence between those services.

Functional specification

The functionality of the separate services are described here in detail, stating inputs and outputs and the workings of the services and optional sub services.

These are all documented with paper and pen, and by working together in groups of up to 4 persons.

5.2 Orange Case mechanics

The workings of the Orange Case are quite simple. The students have to think up an idea or plan, clarify it in simple terms in the operational specification, divide the plan into detailed steps in the recipe, state the basic functional needs for each of the steps to work and map the functionality of the separate steps together in the functional specification. The step we are interested in is the generation of the functional specification. The other steps will be elaborated upon, but are not within the main scope of this thesis.

Functional decomposition

The concept of functional decomposition was introduced earlier in Chapter 4, in which the usage of it to break down complex functionality into easy to understand subfunctions was explained. Functional decomposition comes into play with the functional specification step of the Orange Game, in which it can be used to functionally decompose the processes of the functional specification. Because a detailed description of the separate services is required, functional decomposition not only delivers that description of the services themselves, but also their inputs and outputs and maps those between the separate services.

The whole idea behind the Orange Game is to teach new students the concept of functional decomposition. As functional decomposition using paper and pen can be a tedious job, we seek to find a suitable method to facilitate the process, making it engaging, inviting and challenging, which brings us to the subject of Serious Games.

5.3 The Orange Case Game

It is already mentioned that the current version of the Orange Case is performed with paper and pen. This way of working enables the students to correct mistakes with little effort, allowing them to develop a conceptual idea and work it out before handing in the final version. In the current version of the Orange Case, students have to learn the matter of functional decomposition by the 'learning by doing' approach.

There are many students however, and limited lecturers and teaching assistants. This can lead to students receiving less coaching and feedback on their models than needed, and misunderstanding of the teaching material. The goal of the Orange Case is to teach the students the principle of functional decomposition, but as this is totally new matter to them, they need a lot of feedback or guiding, to tell them they're heading into the right direction with their models, or if they need to modify anything.

The Orange Case is a good example in which a Serious Game can help the students understand the theory of functional decomposition, with added benefits that Serious Games provide, such as an attractive graphical user interface, direct feedback on the students' input, a digital workspace with easy exporting options (screen prints, email, even social media), and visual guidance on tasks that still have to be performed to deliver a correct functional decomposition model.

In the next chapter, we will take our thoughts on information systems modelling, functional specification, Serious Games, iteration, functional decomposition and the Chinese Boxes principle and construct a framework which will result in a Serious Game in which iterative functional decomposition is incorporated. This game will be made in such a manner that it will be a helping hand for anyone who wants to perform functional decomposition, so it will be as general as possible, but with some minor tweaks, like graphical and textual changes, it can be made into a Serious Game that Orange Case users can use.

6. Creating the Game

So far, we have delved into several innovative concepts, all of which will be combined in this chapter. We will analyse our situation and goal, specify the game elements that we need to reach our goal, make a design for each of those game elements and then implement them, constructing our **Iterative Specification Game**.

6.1 Research Goal and starting point

The research goal of this thesis is *the design and construction of an operational functional specification game*, as stated in the introductory chapter. To reach this goal, we will achieve these two sub-goals:

1. The design and construction of a theoretical framework based on game-like elements to guide a player through a Serious Game in which an existing system is declaratively specified through functional decomposition. The guidance is provided in such a way so that iteration is facilitated.
2. Apply the rules of the theoretical framework to the current state of the Orange Case.

Method

We will break down these high-level sub-goals into tangible steps for better comprehension:

- We will decide upon certain design and implementation aspects of the game beforehand, like the functional scope of the game, the intended audience and implementation methods we will use.
- We define the theoretical framework, a collection of rules and guidelines which enable a user to specify an (information) system through functional decomposition.
- For each of these rules and guidelines:

- We analyse and substantiate how each of these rules and guidelines enable the player to reach the main goal of the game, that is, modelling the system through iterative functional decomposition;
- We shortly describe which game-like mechanics are required to enforce this rule or guideline and which concepts of functional composition, iteration or other methods can be used to implement it;
- We then construct an implementation of the theoretical framework, resulting in our Serious Game.

In the previous chapters, we have covered the topics of models and their importance, the involvement of stakeholders and their technical limitations in a systems engineering process, the functional specification and its operational model, Serious Games and how they benefit learners and trainers, iteration and how it's used in systems engineering, functional decomposition and functional composition, and the Chinese Boxes principle and how it combines both iteration and functional decomposition.

6.2 Preparation

Before we get to the actual analysis, design and implementation of our game, we make some design decisions and assumptions in advance. This will help us to stay focused on the task at hand and to build the right application to solve our research problem.

Game name

We choose the name *Fun2Build* for our game. It is a simple name, which indicates that the game will be 'fun to play' and that there's a building aspect involved, the building of a model for the functional specification.

Intended audience

The intended audience of our game is everyone who is involved in constructing a functional specification of an information system, be it a modeller, a stakeholder or an end user. Anyone who delivers direct input for the system's functional specification can learn the process of functional specification through use of the game. The main player of the game will be referred to as *the player* of the Fun2Build game.

Analysis and design scope

During analysis and design of the game, only the aspects of Serious Games, iteration and functional decomposition and their related matters are in scope. Our mission is to provide the end user with a game to combine these aspects in order to reach the main goals, and we will keep that in mind the whole time to prevent wandering off.

Implementation focus on functionality, not technical performance

During the implementation of our game, we will focus on the core task, that is the realisation of an operational functional specification game and its characteristics. As this is the first time we perform this task, we will not focus on technical details like implementing the most efficient functions possible or using the least data bandwidth. The application is what counts, CPU loads or timing are not in focus.

Used techniques

As stated before, our game will involve the techniques of functional specification, Serious Games, iteration and functional decomposition.

Used technologies

As a manner of implementing, we have chosen to use the server-side scripting language PHP (abbreviation of *PHP: Hypertext Processor*), specifically PHP version 5.3.3. PHP is an open source, freely available language which runs on various operating systems, is approachable through a web browser and is a proven scripting language. Large websites like Facebook, Wikipedia, Yahoo, Digg and Wordpress are implemented for the greatest part or even entirely in PHP, providing output for dynamic web pages and the ability to implement 'programming' functions.

For our data structure, we have chosen to use XML (eXtensible Markup Language). XML is a data structure that is easy to read and understand for both information systems and humans, very flexible and extensible as you can add new XML tags at any moment in time, facilitates comparison and aggregation of data through its internal tree structure and is open source. All of these properties are beneficial for implementing our game.

For our Fun2Build Serious Game, the combination of PHP and XML will provide us with means to implement our own, carefully designed functions, a flexible data structure and a way to display the data and visible output in a visually attractive manner.

Functional decomposition number of abstraction levels

Functional decomposition will play an important role in realising our game. As explained before in chapter 4, functional decomposition can be performed at several abstraction levels, with each level providing a different level of detail. We choose to delimit the number of abstraction levels for functional decomposition at 3 levels. This provides us high level information of the main process that we will model at abstraction level 0 or the system level, detailed information about the subprocesses that make up the main process at abstraction level 1 and even more specialized, detailed information about the sub-subprocesses at abstraction level 2.

Using the three internal levels of abstraction and the principle to use 7 ± 2 functions or system components at each level to keep a good oversight, this gives us a theoretical maximum of

$9^3 = 729$ functions, subfunctions and sub-subfunctions that make up the functionally decomposed model of the dissected system.

6.3 Serious Game vs. 'standard tool'

The main concept that we will use as a basis is that of a Serious Game. In the end, the game is what the end user will use. But why do we use a game in this particular situation and not a standard tool, in the form of 'static' software? What makes a Serious Game so explicitly appropriate to help our target audience, the stakeholders and other non-technical people that are involved, in knowledge transfer and training?

Technical subject

The technical framework we will create and the subsequent game that will be based on that focuses on creating a functional specification. Creating the functional specification is a vital task in the process of information systems engineering. The functional specification itself describes the technical structure of the system that will be built, its system components and its functions.

Although it is of great importance for a good result, the specification itself in its traditional form is nothing more than a document with a lot of text, a document that clearly describes the *technical structure* of an *information system* and its *system components* and its *functions*, *inputs*, *outputs* and *system behaviour*. While all of these terms may sound familiar to a technical specialist like an information architect or a programmer, most of the stakeholders in a system development project lack the technical knowledge to directly point out the information sought after by the modeller.

They are a great source of information however, so the task that our framework and game will have to accomplish is actually two-fold: on one hand, we will try to teach stakeholders the concepts behind functional decomposition, so they can provide useful input for the model that will help build the functional specification, on the other hand, it is to provide the modeller with an instrument to actually build that model. A Serious Game is a perfect medium to accomplish both tasks, providing several benefits that suit our situation.

Serious Games for motivation

Teaching the concept of functional decomposition to people with little to no technical background can prove a tedious job. Even though functional decomposition is a fairly simple technical subject which can be applied to all kinds of situations, even outside of the domain of systems engineering, it still is a technical concept which needs careful instruction.

To keep non-technical stakeholders motivated throughout the process of learning-by-doing, Serious Games encompasses lots of game-like mechanics to present players with an entertaining experience, offering them challenges, visual feedback, and short and long term

gratification , which helps to keep them motivated. A problem that is presented as dry matter is less attractive than the same problem, presented as a television show with exciting visuals.

Serious Games for engagement

Games can be used to teach a particular topic to its players, providing active transfer of knowledge. Standard tuition of a subject is usually a static, one direction activity. A game is dynamic and interactive, and lures the player into actively participating in solving the problem, or in our case, providing information about system functions, transactions, et cetera.

If executed well, a Serious Game which engages the player actively in the process of learning by doing, the game can also leave a cognitive impression on the learners apart from the communicated message. This cognitive impression can possibly lead to better mastery of skills and strategies concerning the subject taught (Gros07, Salomon91). Part of a Serious Game is the 'fun factor', and although it isn't the most important factor, if a game has a sufficient fun factor to be enjoyable to the player, it can lead to an increased level of engagement in the game, and subsequently, an increased understanding of the game's rules, procedures, et cetera.

In our game, an increased understanding of the main subject, the concept of functional decomposition, gives the player control over the situation. Instead of just providing the answers to the questions that the modeller asks them, they can use their knowledge they gained by playing the game to engage in the modelling process themselves. The visual aspect of a Serious Game enables it to be played on a screen that is visible to multiple players, and in such, it also encourages simultaneous engagement in the modelling process by a group of players.

Serious Games for better understanding

Functional decomposition is a subject that is best learned by practising it, as mentioned before. Our game will be focused on one main problem, that is that a particular system hasn't been modelled through functional decomposition yet. The player then will have to actually use the concept of functional decomposition himself to solve the problem, and in doing so, learns the theory by performing it in practice, providing them with hands-on experience and better understanding of the theory. Serious Games also offer the player feedback in many forms, to help guide the player towards the correct learning goals of the game.

According to Whitton (Whitton07), if a Serious Game is problem-based and it is clear to the player which problem to solve, it allows the player to experiment with game components and it actively engages the player, it has the potential to be a very effective environment for learning, not because it's a game, but because it exhibits the characteristic of constructivist learning environments. All in all, we pick a Serious Game above a static piece of software because it offers much added value to the user than only the transfer of knowledge itself.

6.4 Analysis and Design

For Fun2Build, our Serious Game, we will need to implement certain functionality, instruments and game mechanisms to ensure that the player will be able to play the game we intend them to. As stated before, the concept of a Serious Game is very general; each Serious Game has its own characteristics, unique feel and look.

6.4.1 Hoppenbrouwers, van Bommel and Järvinen's comparison

In (Hoppen08), Hoppenbrouwers, van Bommel and Järvinen review some of their own work in the field of Method Engineering, a collections of concept that make up a framework for an interaction system which brings forth models. These concepts are then compared to a gaming framework for analysis and design by Järvinen (Järvinen07).

The comparison which is part of the end result of (Hoppen08), supports Hoppenbrouwers et al.'s vision of analysis and design of systems that embody and support the creation of models as if it concerns the design of games. As this is very in line with our own vision for our Serious Game, which will support the creation of models to contribute to a functional specification of a system, the comparison of (Hoppen08) seems a viable reference and possible starting point for the analysis and design of our game.

Hoppenbrouwers et al. mention these concepts for 'games for modelling' in their resulting comparison. Following is a short summary of their concepts:

Components

These are defined as the intermediary deliverables and end deliverables of the game, and all related elements. This general description includes elements like objects, relations, processes, textual descriptions, et cetera.

Rule set

A distinction between two types of rule sets is made:

Goals – A modelling games should have a logical ultimate goal that is to produce a specific model, with underlying sub-goals to complete other requirements of the model. These sub-goals can also be intermediary steps and deliverables in a process, or even time-related process goals.

Procedures – Advised or prescribed ways for the participants to structure and fulfil their tasks as well as rules that apply to the game system.

Environment

'*The stage for gameplay*', as the authors call it. The environment exists of views, visualizations and other surroundings in which the game mechanics are acted out.

Game Mechanics

Game mechanics are the interaction types that can be applied to modelling, translated to a sort of functionality that the system or game offers. Examples are arranging, building, choosing and storytelling. In addition, more complex, compound game mechanics may be specified that amount to *tasks*, which the player can choose to perform as part of the gameplay.

Theme

A theme is the domain context in which helps the player to focus the mode of communication.

Information

The authors mention information about modelling events, agents/participants, objects and information about system rules, scores et cetera, which are present all around.

Interface

The objects or deliverables need to be acted upon, which requires operationalization of game mechanics in the form of basic interactions. The interface enables the player to initiate and execute these interactions.

Players

There are several aspects of the players of the game that can be taken into account, like player preferences, behaviour, competencies et cetera.

Context

The game can be set in a specific context, which relates to, among others, the purpose of the game as a whole, backgrounds and capacities of players.

Rule-based method modelling

The authors present rule-based method modelling principles like the representation of methods by rules, which enable techniques like automated model checking and context checking. They also mention situational procedural regulation, the checking of a process and providing guidance to the player by means of a dynamic 'modelling agenda'.

Score systems

Feedback for players, presented by way of a scoring system tells them how well they are doing. The authors want to link score systems to operational quality metrics, which enables encouragement and guidance of the players to achieve the highest possible quality without prescribing them what to do in every step.

Playability and emotive factors

The link between the game system and the emotions and experiences it creates in the players are crucial aspects in game design. The authors mention that the main goal in a design game is not entertainment, but it benefits from aspects like excitement and challenge to evoke positive emotions, while also helping to avoid negative emotions like frustration and boredom.

For our own Serious Game, we will adapt some of these definitions in our own analysis and design, but not all of them. We will start with the concepts which we already have information about, to derive the rules and guidelines that form our framework.

6.4.2 Technical Environment

Framework guideline

The Serious Game will be an accessible web application (based on PHP and XML) that will be accessed through a web browser.

One of the first things we already decided upon is the environment for our Serious Game. To make the game technically accessible to a broad audience, we will use technology that works cross-platform. In this case, we have already chosen to implement the game in the scripting language PHP and to use XML for our data structure. A combination of these two technologies will result in a web application which runs on most of, if not all modern internet browsing programs like Microsoft Internet Explorer, Google Chrome and Mozilla Firefox. The application will thus be designed for viewing within a web browser window.

6.4.3 Gaming environment

Framework guideline

The gaming environment will be displayed in the web browser and will be visually attractive.

The gaming environment itself is the entire application as it is shown on screen. The gaming environment contains the game's interface (graphical user interface or GUI) and is the 'main stage' in which the player's actions will take place and its results are shown.

The environment will be visually attractive to increase the motivation of the player to actively participate in the task at hand. To implement this, we will make use of a colourful design for the game, using lots of visual elements like nice logo's and icons for the GUI and providing clear textual information. The gaming environment will also make use of a logical layout of GUI items, grouping them together. The contents of the GUI and how the player will be able to interact with it will be explained later on.

6.4.4 Goals, sub-goals and tasks

Framework guidelines

The main goal of the game is to build a model that contributes to a functional specification of a certain system that is modelled.

The main goal can be broken down into sub-goals, which can be achieved by completing the tasks that are linked to the sub-goals.

The main goal and the sub-goals of the game will be communicated to the player in a clear way.

The main idea behind the game is to reach a certain goal, that is, to build a model that contributes to a functional specification of a certain system. Therefore the main goal has to be made clear to the player before and while playing the game. To achieve this main goal, several sub-goals can be defined.

The sub-goals can be quite general, like 'Complete the function table', and can be translated into a set of tasks that the player can perform to complete those goals or sub-goals. In the case of a goal 'Complete the function table', the task at hand can be 'Add a transaction coming from the function'. Information about goals, sub-goals and tasks can be used in the concept of guidance and feedback, which will be discussed later on.

To keep the player motivated and to provide sufficient feedback on the player's performance, it has to be clear to the player which goals and sub-goals he or she has to achieve. Possibly, previously accomplished goals can be shown as an example in which he/she has completed this goal successfully before.

When all of the game's sub-goals have been completed, the main goal is reached and the player has completed the game. Through the concept of iteration however, the player can decide to perform actions after completion, like adding new functions to the model, which result in new sub-goals to appear. This reverts the game from a status in which the main goal is achieved to one in which the player has to solve new sub-goals first, after which the game is completed again. This gives the player control of how detailed the model can become and consequently, how many modelling iterations take place.

6.4.5 Game elements

Framework guideline

The game will present the player with several game elements, which represent elements within the concepts of functional specification and functional decomposition.

To achieve the goals of the game, and to play it well, the player will have to make use of the actions that he/she can perform within the game. Each of those actions perform some

transformation on one of the game elements. We define several game elements and their properties within our game:

Model

Type

XML Data structure

The model is a data structure which holds all our modelled data. Each of the gaming elements that are modelled in the game will be added to the model. The contents of the model can be presented through the graphical user interface at request of the player, so the player can see what has been modelled so far.

Function

Type

Modelling element

Attributes

Function name

Function input (Type Transaction)

Function output (Type Transaction)

Parent function (Type function)

Definition

The function is the most basic building block of our game and represents a part of our system. The player will be able to functionally decompose a function into other functions, each representing a piece of functionality of the original function. Within our game, a function can exist at three abstraction levels:

1. As a *system function*, which represents the entire system as a whole.
2. As a *subfunction* which is a functionally decomposed component of the system function.
3. As a *sub-subfunction* which is a functionally decomposed component of a subfunction.

Rules

1. A function is identified by an obligatory, unique function name (e.g. Produce Financial reports), a declarative description of what the function does. The function name exists of a maximum of 7 words.
2. A function can represent an entire system or process, or parts of that system.
3. A function can have one or more functions that it can be functionally decomposed into.
4. A function which is a functionally decomposed component of a higher level function is called a *subfunction* of that higher level function. For each following level of functional decomposition, the prefix 'sub-' will be prepended (e.g. sub-subfunction for two levels of

functional decomposition). The higher level function is called the *parent function* of the decomposed function. The system function has no parent function, as it is at the top of the hierarchical structure.

5. A function has at least one function input of type Transaction.

6. A function has at least one function output of type Transaction.

7. A function will be depicted as a function table, which displays the function name, the function inputs and the function outputs.

Transaction

Type

Modelling element

Attributes

Transaction name

Transaction supplier

Transaction customer

Definition

A transaction is a flow of goods, actions, or information that is passed on from a supplier to a customer. A transaction can act as an input or an output to a system process.

A transaction represents something that is passed on between exactly two system components. Such a transaction can be virtually anything, ranging from an action that is performed by one component on the other, a physical object or virtual piece of information that is passed on from the supplying party to the receiving party of the transaction.

Rules

1. A function is identified by an obligatory, unique transaction name (e.g. firewood, product package), a declarative description of what the transaction represents. The function name exists of a maximum of 7 words.

2. The transaction always flows from the transaction supplier to the transaction customer.

3. Each transaction has at exactly one supplier and one customer. If there are multiple transactions of the same type between the same supplier and customer, these are modelled as separate entities (e.g. *Bank account withdrawal 1* and *Bank account withdrawal 2* for two transactions between the same bank and the same bank customer).

6.4.6 Game mechanics and actions

Framework guideline

The player will be able to interact with the game environment through actions which allows him/her to manipulate the game elements.

All actions that are taken while playing the game are performed within the gaming environment. The actions enable the player to add, remove or otherwise modify the game elements. Certain core actions have to be available to the player, while others can be added to provide the player with an enhanced gameplay experience. In this thesis, we will focus on the core actions that are required in a functional modelling game:

Add Function

Because building a model requires the addition of our main game elements, the functions, there has to be an action available to add a new function to the model. When adding a new function to the model, the player is required to provide a function name, as stated in the rules for the game element function. The abstraction level also has to be taken into account, so the player also has to determine whether the new function is a system function, a subfunction or a sub-subfunction. In addition, when creating a new subfunction or sub-subfunction, the player has to indicate which function is the newly created function's parent function. This is done in order to set a baseline hierarchy. Note that when adding a new function, we only look at parent-child hierarchical connections, while hierarchy concerning siblings is of no concern to us.

Add Transaction

Transactions connect the functions, they are the inputs and outputs of the functions and get passed on from one function to another. When adding a new transaction, two things have to be specified, its supplier and its customer. A customer or supplier can be an existing function within the system, or an external source, which is not part of the system that is modelled. In the case of a transaction having an external source as a supplier or customer, that transaction also becomes a system transaction, a transaction of the system function. The reason for this is the hierarchical passing of transactions, as described in chapter 4.

Remove Function

When a function is deemed unnecessary in the model, it can be removed. When a function is removed from the model that has function inputs or outputs, those transactions become loose ends, transactions that miss either a supplier or customer, as described in chapter 4.

Loose ends themselves are not by default unnecessary or erroneous, as it is possible that the modeller has created one on purpose. An example of this is when the removed function remains structurally unchanged, but is placed out of system context, for instance in case of outsourcing of the function to a third party company. The existence of the loose end

transaction is then still valid, but its destination is now a system external component instead of a system internal component.

A special case is when a function is removed and it has a function input that originates from an external source. If the function is removed, the external input has no validity anymore and will have to be removed as a whole.

Remove Transaction

As with unnecessary functions, unnecessary transactions will have to be removed as well. While doing this, if there's a function that acted as either a supplier or customer to this transaction, it has to be re-evaluated to see whether that function or those functions still have at least one function input and at least one function output, as stated in the function rules described before.

Show overview

The player is building a model with several functions and transactions. The game should contain an action that shows the current state of the model, so the player can see what has been added to the model and to have an overview of what actions still have to be executed to complete the model.

6.4.7 Guidance and Feedback

Framework guideline

The player will receive visual and textual feedback while playing the game, to guide him/her towards completing the right sub-goals and the main goal.

The main goal, sub-goals, tasks, functions, subfunctions, sub-subfunctions transactions, and most important, the model itself are all important components of the game, which have to be constructed, monitored, added, removed et cetera meticulously to build an as complete as possible model. All this information can be quite a lot for the player of the game to process and retain in memory. This is the reason that the game should give the player plenty of feedback, preferably in a visual way, as the game is visually oriented.

The overview of the model mentioned in the section above is the main feedback method for the player. It shows the player what has been added to the model, the structure of all the functions and how they are connected. The player can also identify information gaps like a missing function input or output by looking at the model.

A second method of feedback is the so called *To do list*, a checklist type of list that shows the player the active sub-goals and corresponding tasks that have to be fulfilled in order to build a proper model and to complete the game. The sub-goals and the tasks in our game will be based on several conditions, which depend on the execution of certain actions on the game elements. The To do list will be shown in this format:

To do List

Sub-goal heading

Task that has to be fulfilled to complete the sub-goal 1

Task that has to be fulfilled to complete the sub-goal 2

etc.

For example, after a function has been added while playing the game, at least one function input and at least one function output must be added as well. This will be shown as:

To do List

Complete Function Table

Add a function input to function A

Add a function output to function A

The To do list provides direct feedback to the player after each action, and also provides the player with an incentive to perform iteration, in a sense that the player has to review the model, re-think his or her past actions and to execute the correct follow-up action.

6.4.8 Graphical User interface (GUI)

Framework guideline

The GUI elements which enables the player to perform actions and provide feedback for the player to process, will be positioned on screen in a logical manner.

The GUI allows the player to execute the aforementioned actions and show the visual feedback. Apart from having a colourful and inviting design, the GUI should be as functional as possible, that is, providing no extra information that is not needed to perform the task of modelling.

GUI elements that need to be displayed are the model, which is shown in the middle of the screen, as it is the most important element, a menu component that shows all available actions and a component that shows the To do list. We choose to display the GUI elements in a structured, three column view, placing the action menu to the left of the model and the To do list to the right of the model.

6.4.9 Procedure

The basic course of events will be as follows:

1. The player opens the Fun2Build game in his/her internet browser.
2. An introductory text is shown which states the purpose of the game. At the end of the text, the player is able to choose to follow a tutorial which teaches the player the actions available within the game and how to use them to build an example model.

3. (optional) The player follows the tutorial, learns the purpose of each action, and how to combine functions and transactions.
4. The player chooses to play the game itself by pressing a 'start modelling' button.
5. The player is asked for the name of the main function or process that will be modelled. This is labelled the system function and displayed as an empty function table. The to do list is updated to show the player that a function input and function output.
6. The game actually starts, the player adds functions and transactions to the model and receives feedback in the form of a popup or items on the To do list. The player has to perform tasks to complete the To do list sub-goals. Eventually, all sub-goals will be reached and the player has completed the game.
7. The player presses a button 'End game' and is shown a screen, congratulating him/her for successfully completing a task which is actually quite technical and that he/she should be very proud.

The goal of the Fun2Build game is to build a model that contributes to the functional specification of a system or process. At the end of the basic course of events, the player will have succeeded in doing that.

However, the described basic course of events is just a baseline. The actual game play will rely on what choices the player makes, for example: Will he/she model all subfunctions and sub-subfunctions first and then complete the transactions? Or will he/she add one function, complete it and then move on to the next function? How many levels of abstraction will the player choose to model? How many iterations will the player execute while playing the game?

Framework guideline

The game provides a tutorial in which the player learns how to use the game as a modelling tool, before modelling themselves.

Chances are that the player has little to no technical knowledge and chances are even more slim that the player has performed functional decomposition before. Therefore, we choose to present the player with two choices when he/she starts the game. The first choice is a tutorial, in which the functionalities are explained that the player has access to within the game. The second is the process of modelling itself.

The tutorial will be an interactive one, according to the 'learning by doing' aspect of Serious Gaming. The tutorial will explain all the functionality available within the game and introduce the player to aspects of functional decomposition. The explanation of technical aspects within functional decomposition, like functions and transactions, will be presented in a manner that is not too technical.

By offering the player to choose whether he/she wants instruction in advance, or to choose to start modelling right away, we can assure that the player has at least a basic understanding of the game's mechanics, rules and functionality, before actually starting to model.

6.5 Implementation

We will now implement this framework, its design and its rules and produce a Serious Game. This game will guide a player through the process of functional decomposition to produce a model which contributes to the operational structure of a functional specification of a system or a process.

The game will be implemented in PHP and XML, development will take place in the Eclipse Foundation Eclipse IDE and testing it will be done on Wampserver64, an open source server package which runs PHP 5.3.3. For the complete code of the several webpages of the Fun2Build game, refer to Appendix A. The next chapter, chapter 7, will provide you with a walkthrough of the Fun2Build game.

7. Results

W*e have constructed a first version of a Serious Game, a prototype if you will, based on the framework of rules and guidelines that we proposed in the previous chapter. This Serious Game provides the player with a way to learn how to construct a model that contributes to the functional specification of a system or process, while incorporating the concept of iteration. This chapter will provide you with a walkthrough of the game, showing its visual and functional features.*

7.1 Getting started

As we stated in previous chapters, the game that we made will be played by stakeholders who help a modeller to complete parts of a functional specification of a system or process. The basic course of events of the game was already concisely described in section 6.4.9 and will be now be further elaborated upon.

The game starts when the player opens an internet browser screen and opens the Fun2Build game homepage, **index.php**.

7.2. Index.php - Welcome to the Fun2Build Game!

Screenshot

fun2build

Menu

- Show Overview
- ADD function
- REMOVE function
- ADD transaction
- REMOVE transaction

Welcome to the Fun2Build Game!

Welcome to the *Fun2Build Game!*
In this game, we will learn you how to construct and refine a model for the functional specification of a system or a process. Don't be scared if that sounds technical, it's actually quite easy!

What is a model?
A model is a graphical representation of information, in this case about the structure of a specific system or process. The term used to construct a model is called **modelling**.

Example of a simple model:

```
graph TD; Animal[Animal] --- Cat[Cat]; Animal --- Dog[Dog];
```

What is a functional specification?
A functional specification is a document which helps the system development team to develop the right functionality of the system. It focuses on the **functional process of the system** and how it works.

The model that we will make during this game will tell the team what **YOU** want it to be able to do!

Don't be afraid if you have never done this before, we will help you through this!
Click on the **Follow the Tutorial** button if this is your first time building a functional specification, or Click on the **Start modelling** button to start right away!

Todo List
Empty

Figure 7.1: index.php

Short description

The homepage shows the basic setup of the game's GUI, the **Menu bar** with all available player actions (*Show Overview*, *ADD function*, *REMOVE function*, *ADD transaction* and *REMOVE transaction*) to the left, the page's main content in the middle and the **Todo List**, which provides feedback on the sub-goals and tasks that the player needs to fulfil to the right of the page.

This page welcomes the player to the game, gives the player a general introduction of the terms *model* and *functional specification* and tells the player that he/she will be guided to 'construct and refine a model for the functional specification of a system or a process'. The player is then presented with the choice to **Follow the tutorial** (See section 7.3) or to **Start modelling** (See section 7.4).

Page Transcript

Welcome to the Fun2Build Game!

In this game, we will learn you how to construct and refine a model for the functional specification of a system or a process. Don't be scared if that sounds technical, it's actually quite easy!

What is a model?

A model is a graphical representation of information, in this case about the structure of a specific system or process. The term used to construct a model is called **modelling**.

Example of a simple model:

[image]

What is a functional specification?

A functional specification is a document which helps the system development team to develop the right functionality of the system. It focuses on the functional process of the system and how it works.

The model that we will make during this game will tell the team what **YOU** want it to be able to do!

Don't be afraid if you have never done this before, we will help you through this!

Click on the **Follow the Tutorial** button if this is your first time building a functional specification, or

Click on the **Start modelling** button to start right away!

["Follow the Tutorial" button (Links to **tutorial1.php**)]

["Start modelling!" button (Links to **startmodelling.php**)]

7.3. Tutorial

The player can choose to follow a 17 step tutorial before starting to create the model. During the tutorial, the player is introduced to all actions that he/she can perform to build the model in the Fun2Build game. The tutorial is action-based, meaning that the player performs the action that is explained right away. The instructions that are given are based on an example process of Making a cup of instant coffee, or simply said, **Making coffee**.

7.3.1 Tutorial.php - Tutorial - Functions and Transactions

Screenshot

Tutorial - Functions and Transactions

Let's start our tutorial!
Within our game, we focus on the *functionality of the system* that we will model. Our models exist of two main components, **functions** and **transactions**.

Functions
The system or process that we will model into our model is seen as a **function** within our game. This function will transform its **function inputs** (or just *inputs*) into its function outputs (you guessed it, just *outputs*). The system function has a **function name**, which describes what functionality the system performs.

Transactions
Inputs and outputs combined are called the function's **transactions**.

A **transaction** always has these three properties, whether it's an input or an output:

1. **Supplier** - Where does the transaction come from?
2. **Transaction** - A short descriptive name which indicates what kind of transaction it is. A transaction can be anything that is passed on within the system, ranging from information (a document) to an action (press a button).
3. **Customer** - Where does the transaction go to?

An example of how functions and transactions are related to each other is given below. Input A and Output B are Function 1's transactions. Function 1 can be any system we want to model.

```
graph LR; A[Input A] --> F1[Function 1]; F1 --> B[Output B];
```

In the above example, the function input Input A has these properties:

1. **Supplier** - The supplier is still unknown, but let's say it is called 'Party X'.
2. **Transaction** - Input A.
3. **Customer** - Function 1.

```
graph LR; PX[Party X] --> A[Input A]; A --> F1[Function 1];
```

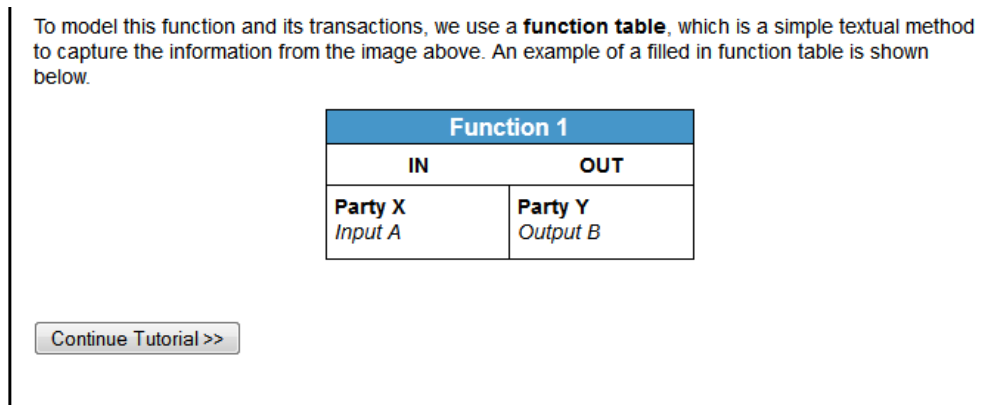


Figure 7.2: tutorial.php

Short description

The first page of the Tutorial explains how the system or process that will be modelled in the model is seen as a function with function inputs and function outputs. The concept of Transactions is also explained and examples of how functions and transactions relate to each other (the Supplier->Transaction->Customer relationship) is shown.

Page Transcript

Let's start our tutorial!

Within our game, we focus on the functionality of the system that we will model. Our models exist of two main components, **functions** and **transactions**.

Functions

The system or process that we will model into our model is seen as a **function** within our game. This function will transform its **function inputs** (or just *inputs*) into its function outputs (you guessed it, just *outputs*). The system function has a function name, which describes what functionality the system performs.

Transactions

Inputs and outputs combined are called the function's **transactions**.

A **transaction** always has these three properties, whether it's an input or an output:

1. **Supplier** - Where does the transaction come from?
2. **Transaction** - A short descriptive name which indicates what kind of transaction it is. A transaction can be anything that is passed on within the system, ranging from information (a document) to an action (press a button).
3. **Customer** - Where does the transaction go to?

An example of how functions and transactions are related to each other is given below. Input A and Output B are Function 1's transactions. Function 1 can be any system we want to model.

[image]

In the above example, the function input Input A has these properties:

1. Supplier - The supplier is still unknown, but let's say it is called 'Party X'.
2. Transaction - Input A.
3. Customer - Function 1.

[image]

To model this function and its transactions, we use a **function table**, which is a simple textual method to capture the information from the image above. An example of a filled in function table is shown below.

[function table]

["Continue Tutorial >>" button (Links to **tutorial2.php**)]

7.3.2 Tutorial2.php - Tutorial – Your tools

Screenshot

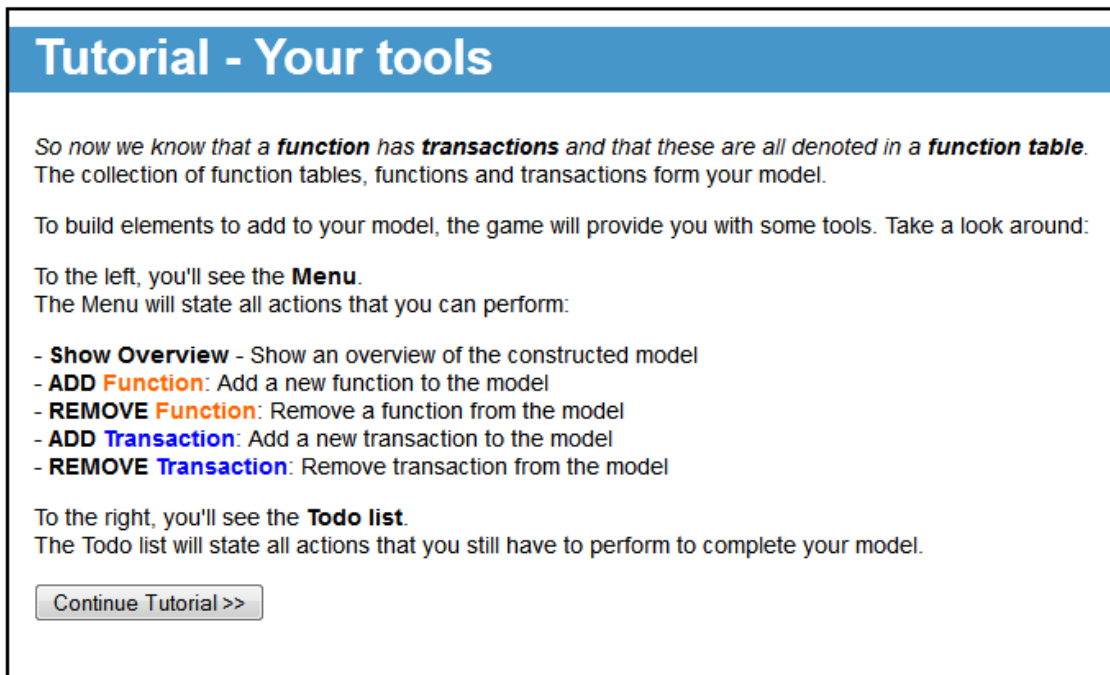


Figure 7.3: tutorial2.php

Short description

The second page of the Tutorial explains the Menu item shown to the left of the page and each of the menu actions that the player can use to build the model. It also introduces the Todo list shown at the right of the page.

Page Transcript

So now we know that a **function** has **transactions** and that these are all denoted in a **function table**. The collection of function tables, functions and transactions form your model.

To build elements to add to your model, the game will provide you with some tools. Take a look around:

To the left, you'll see the **Menu**.

The Menu will state all actions that you can perform:

- **Show Overview** - Show an overview of the constructed model
- **ADD Function**: Add a new function to the model
- **REMOVE Function**: Remove a function from the model
- **ADD Transaction**: Add a new transaction to the model
- **REMOVE Transaction**: Remove transaction from the model

To the right, you'll see the **Todo list**.

The Todo list will state all actions that you still have to perform to complete your model.

["Continue Tutorial >>" button (Links to **tutorial3.php**)]

7.3.3 Tutorial3.php - Tutorial – Who doesn't like coffee?

Screenshot

Tutorial - Who doesn't like coffee?

It is now time to take your first steps in the world of modelling.

We are going to model the process **Making coffee**. Don't be confused that we use the term 'process' instead of the term 'system'. We model the process to show what the system that performs this process, has to do. In other words, we now model *the functionality of a system that will perform the process Making coffee*.


Coffee exists in different kinds of types. For simplicity's sake, we will take a very simple type of coffee, instant coffee. All you need for that is an electric kettle to boil water with, some ground coffee and a cup. Hmm. Can you smell the fresh aroma already?

First, we will take a look at the steps that make up the process of Making (instant) coffee.

Making coffee

- 1. Boil water**
 - Fill kettle
 - Turn kettle on
- 2. Put ground coffee in cup**
 - Scoop coffee and put it in the cup
- 3. Pour water over coffee**
 - Pour boiling water into the cup

And done!



We will now show you how to model this relatively easy process. After we're done, you have a completed model and a nice cup of coffee!

[Continue Tutorial >>](#)

Figure 7.4: tutorial3.php

Short description

The third page of the Tutorial explains the example process of **Making coffee** that the player will use to construct his/her first model with the Fun2Build game. The focus is on the functionality of a system or process that is being modelled.

Page Transcript

It is now time to take your first steps in the world of modelling.

We are going to model the process **Making coffee**. Don't be confused that we use the term 'process' instead of the term 'system'. We model the process to show what the system that

performs this process, has to do. In other words, we now model *the functionality of a system that will perform the process Making coffee*.

Coffee exists in different kinds of types. For simplicity's sake, we will take a very simple type of coffee, instant coffee. All you need for that is an electric kettle to boil water with, some ground coffee and a cup. Hmm. Can you smell the fresh aroma already?

First, we will take a look at the steps that make up the process of Making (instant) coffee.

[table showing the steps for the process Making coffee:

1. Boil water

- Fill kettle
- Turn kettle on

2. Put ground coffee in cup

- Scoop coffee and put it in the cup

3. Pour water over coffee

- Pour boiling water into the cup

And done!]

We will now show you how to model this relatively easy process. After we're done, you have a completed model and a nice cup of coffee!

["Continue Tutorial >>" button (Links to **tutorial4.php**)]

7.3.4 Tutorial4.php - Tutorial – Getting started

Screenshot

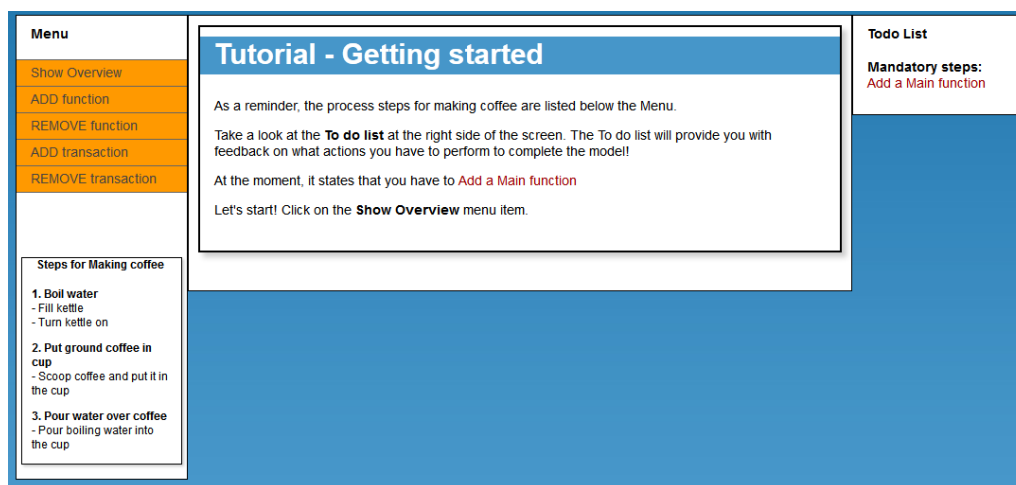


Figure 7.5: tutorial4.php

Short description

The fourth page of the Tutorial adds a reminder of the steps for the process **Making coffee** to the bottom of the menu. The Todo List gives the player its first feedback and task to fulfil, namely to add a main function to the model.

A click on the *Show Overview* menu item links to **tutorial5.php**.

Page Transcript

As a reminder, the process steps for making coffee are listed below the Menu.

Take a look at the **To do list** at the right side of the screen. The To do list will provide you with feedback on what actions you have to perform to complete the model!

At the moment, it states that you have to **Add a Main function**

Let's start! Click on the **Show Overview** menu item.

7.3.5 Tutorial5.php - Tutorial – Show Overview

Screenshot

Tutorial - Show Overview

This is the **Overview** screen. The model that you will build will be displayed here!
Below is an example of a model that you will build:

Making a cup of coffee	
IN	OUT
Boil water: From: Tap Water From: Kitchen Kettle From: Fill kettle Water in kettle From: Electricity socket Electricity Put ground coffee in cup: From: Pot of ground coffee Ground coffee Pour water over coffee: From: Boil water Boiled water From: Scoop coffee and put it in the cup Ground coffee in cup Loose end: Loose end Input Test	Boil water: From: Turn kettle on Water in kettle Put ground coffee in cup: From: Pour boiling water into the cup Ground coffee in cup Loose end: Loose end Output Test Pour water over coffee: From: Client A nice cup of coffee

Subfunctions

Part of Making a cup of coffee: Boil water	
IN	OUT
Fill kettle: From: Tap Water From: Kitchen Kettle Turn kettle on: From: Fill kettle Water in kettle From: Electricity socket Electricity	To: Pour boiling water into the cup: Boiled water Fill kettle: To: Turn kettle on Water in kettle

Sub-subfunctions

Part of Boil water: Fill kettle	
IN	OUT
From: Tap Water From: Kitchen Kettle	To: Turn kettle on Water in kettle

Part of Boil water: Turn kettle on	
IN	OUT
From: Fill kettle Water in kettle From: Electricity	No outputs No food used

Let's start by adding our Main function! Click on the **ADD Function** menu item.

Figure 7.6: tutorial5.php

Short description

The fifth page of the Tutorial shows the player an example of a completed model. A click on the *ADD function* menu item links to **tutorial6.php**.

Page Transcript

This is the **Overview** screen. The model that you will build will be displayed here!

Below is an example of a model that you will build:

[image]

Let's start by adding our Main function! Click on the **ADD Function** menu item.

7.3.6 Tutorial6.php - Tutorial – ADD Main function

Screenshot

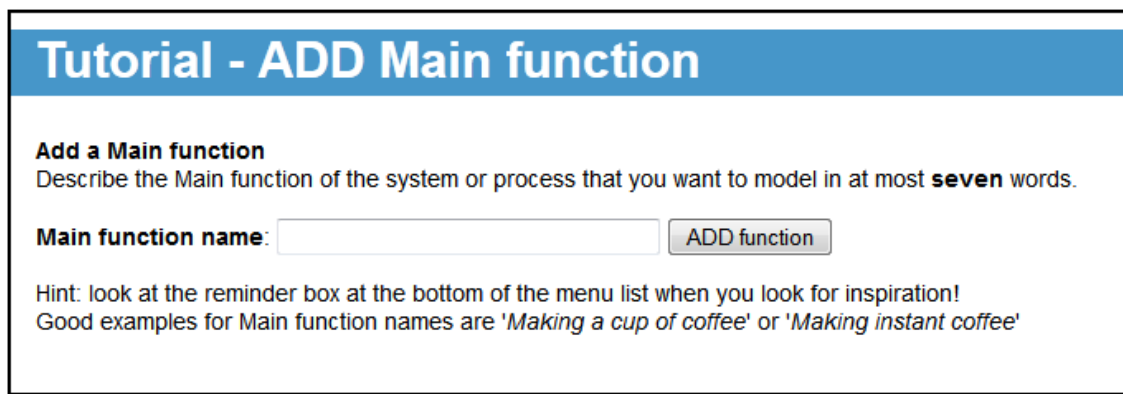


Figure 7.7: tutorial6.php

Short description

The sixth page of the Tutorial instructs the player to add a Main function to the model by entering a name for it. The Main function name which is entered on this page will from now on be referred to as **<Main Function Name>** for explanatory purposes.

Page Transcript

Add a Main function

Describe the Main function of the system or process that you want to model in at most **seven** words.

Main function name: [text box for input] ["ADD function" button (Links to **tutorial7.php**)]

Hint: look at the reminder box at the bottom of the menu list when you look for inspiration!
Good examples for Main function names are *'Making a cup of coffee'* or *'Making instant coffee'*.

7.3.7 Tutorial7.php - <Main Function Name>

Screenshot

The screenshot shows a web interface for a tutorial. On the left is a 'Menu' with orange buttons for 'Show Overview', 'ADD function', 'REMOVE function', 'ADD transaction', and 'REMOVE transaction'. Below the menu is a 'Steps for Making coffee' section with three numbered steps: 1. Boil water (Fill kettle, Turn kettle on), 2. Put ground coffee in cup (Scoop coffee and put it in the cup), and 3. Pour water over coffee (Pour boiling water into the cup). The main content area has a blue header 'Making coffee' and a sub-header 'Main Function Making coffee'. Below this is a table with 'IN' and 'OUT' columns, both containing 'No inputs defined yet.' and 'No outputs defined yet.' respectively. Below the table is a paragraph of text explaining that the main function will be split into three subfunctions: 1. Boil water, 2. Put ground coffee in cup, and 3. Pour water over coffee. The right sidebar contains a 'Todo List' with the text 'Mandatory steps: Add a subfunction'.

Figure 7.8: tutorial7.php

Short description

The seventh page of the Tutorial shows the recently added Main function and a sample function table for that main function. The Todo List is updated so the '*Mandatory steps*' sub-goal and its task '*Add a Main function*' are removed from the list. Instead, a new task, '*Add a subfunction*', is shown to the player.

A click on the *ADD function* menu item links to **tutorial8.php**.

Page Transcript

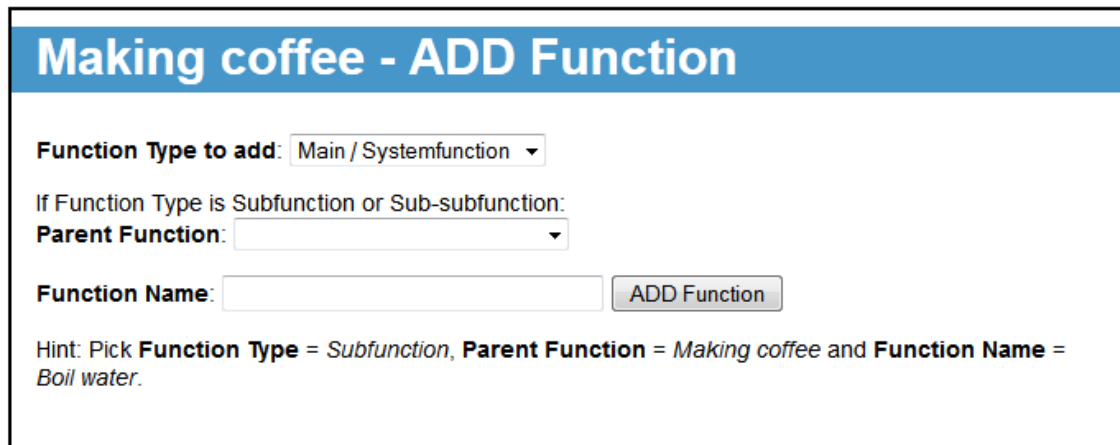
Alright! You have now added your main function (or system function) to the model. A system or a process almost never consists of just one function.

The Main function will therefore be split up into three subfunctions, 1. **Boil water**, 2. **Put ground coffee in cup** and 3. **Pour water over coffee**.

Click on the **ADD Function** menu item to add the first subfunction, **Boil water**, to the model.

7.3.8 Tutorial8.php - <Main Function Name> – ADD Function

Screenshot



The screenshot shows a web form titled "Making coffee - ADD Function". The form contains the following elements:

- A dropdown menu labeled "Function Type to add:" with the selected option "Main / Systemfunction".
- A text label: "If Function Type is Subfunction or Sub-subfunction:"
- A dropdown menu labeled "Parent Function:".
- A text input field labeled "Function Name:".
- A button labeled "ADD Function".
- A hint text: "Hint: Pick **Function Type** = *Subfunction*, **Parent Function** = *Making coffee* and **Function Name** = *Boil water*."

Figure 7.9: tutorial8.php

Short description

The eighth page of the Tutorial is the first page in which the player has to perform a task, the addition of a function, on his/her own for the first time. A hint with possible inputs is shown on screen to help.

Page Transcript

Function Type to add: [drop-down box, options: Main/Systemfunction, Subfunction, Sub-subfunction]

If Function Type is Subfunction or Sub-subfunction:

Parent Function: [drop-down box, options: all Main/Systemfunctions, all Subfunctions]

Function Name: [text box for input] ["ADD function" button (Links to **tutorial9.php**)]

Hint: Pick **Function Type** = *Subfunction*, **Parent Function** = *Making coffee* and **Function Name** = *Boil water*.

7.3.9 Tutorial9.php - <Main Function Name> – Show Overview

Screenshot

Making coffee - Show Overview

Main Function Making coffee	
IN	OUT
Boil water:	Boil water:

Subfunctions

Part of Making coffee: Boil water	
IN	OUT
No inputs defined yet	No outputs defined yet

You can see that the subfunction needs an input and an output. This is shown by the *No inputs/outputs defined yet* messages in the information model and the **Todo List** tasks, marked in **red**. Let's add an input first.

Click on the **ADD Transaction** menu item to add the first input for your subfunction.

Todo List

Complete function table
Add a function input to subfunction
Boil water

Complete function table
Add a function output to subfunction
Boil water

Figure 7.10: tutorial9.php

Short description

The ninth page of the Tutorial shows the effects of the addition of a subfunction. The model now displays the Main function and its subfunction, together with an updated Todo List which urges the player to add a function input and a function output to the newly added subfunction, according to the framework guideline that each function should have at least one input and at least one output, as stated in chapter 6.

A click on the *ADD function* menu item links to **tutorial10.php**.

Page Transcript

[function table for Main function]

[function table for Subfunction]

You can see that the subfunction needs an input and an output. This is shown by the No inputs/outputs defined yet messages in the model and the **Todo List** tasks, marked in **red**. Let's add an input first.

Click on the **ADD Transaction** menu item to add the first input for your subfunction.

7.3.10 Tutorial10.php - <Main Function Name> – ADD Transaction

Screenshot

Making coffee - ADD Transaction

Supplier
Supplier of Input

Transaction
Function input /
Transaction

Customer
of Input

Function output /
Transaction

Customer of Output

Party X

Input

Function 1

Output

Party Y

Supplier
of Output

Transaction

Customer

Remember: **Supplier** -> **Transaction** -> **Customer**

Now, add a transaction to the subfunction that you added in the previous step. The reminder box at the bottom of the menu states that the first subfunction is about Boiling Water. To achieve boiling water, we need at least **Water** from the **Tap**.

Start by adding a **Function Input** named **Water**. The *Function Input* **Water** will have the **Tap** as its *Supplier*, according to the overview displayed above.

Take notice that because **Water** is a **Function Input**, its **Customer** is automatically set to the Function to which the Transaction is added, so you only need to add a **Supplier**.

Function to add Transaction to: Making coffee (Main Function) ▾

Transaction Type: Function Input ▾

Transaction Name:

Only fill in a Supplier if the Transaction Type is *Function Input*.

Transaction Supplier:

Only fill in a Customer if the Transaction Type is *Function Output*.

Transaction Customer:

Hint: Pick **Function to add Transaction to** = *Boil water*, **Transaction Type** = *Function Input*, **Transaction Name** = *Water* and **Supplier** = *Tap*.

Figure 7.11: tutorial10.php

Short description

The tenth page of the Tutorial reminds the player about the connections between the supplier of a transaction and its customer. It then instructs the player to add a transaction named **Water** to the newly added subfunction. **Tap** will be the supplier for *Water*.

Page Transcript

[image]

Now, add a transaction to the subfunction that you added in the previous step.

The reminder box at the bottom of the menu states that the first subfunction is about Boiling Water. To achieve boiling water, we need at least **Water** from the **Tap**.

Start by adding a **Function Input** named **Water**.

The *Function Input* **Water** will have the **Tap** as its *Supplier*, according to the overview displayed above.

Take notice that because Water is a Function Input, its **Customer** is automatically set to the Function to which the Transaction is added, so you only need to add a **Supplier**.

Function to add Transaction to: [drop-down box, options: all Main/Systemfunctions, all Subfunctions, all Sub-subfunctions]

Transaction Type: [drop-down box, options: Function Input, Function Output]

Transaction Name: [text box for input]

Only fill in a Supplier if the Transaction Type is *Function Input*.

Transaction Supplier: [text box for input]

Only fill in a Customer if the Transaction Type is *Function Output*.

Transaction Customer: [text box for input]

["ADD Transaction" button (Links to **tutorial11.php**)]

Hint: Pick **Function to add Transaction to** = *Boil water*, **Transaction Type** = *Function Input*, **Transaction Name** = *Water* and **Supplier** = *Tap*.

7.3.11 Tutorial11.php - <Main Function Name> – Show Overview

Screenshot

Making coffee - Show Overview

Main Function Making coffee	
IN	OUT
Boil water: From: Tap Water	Boil water: To: Boil water Water

Subfunctions

Part of Making coffee: Boil water	
IN	OUT
From: Tap Water	No outputs defined yet

Fantastic! Notice how the Main function automatically duplicates the function input that you entered for the subfunction as a function input **for the Main function**. This is because from an external view, the Water goes into the whole system instead of only into the subfunction!

Click on the **ADD Transaction** menu item to add an output for your subfunction.

Todo List

- Complete function table
- Add a function output to subfunction
- Boil water**

Figure 7.12: tutorial11.php

Short description

The eleventh page of the Tutorial shows the model again. The effects of the transaction addition in the previous step are made visible, as the new transaction is shown in the subfunction's function table. The concept of hierarchical passing of transactions is explained to the player and the Todo List is updated to only require a function output for the subfunction.

A click on the *ADD transaction* menu item links to **tutorial12.php**.

Page Transcript

[function table for Main function]

[function table for Subfunction]

Fantastic! Notice how the Main function automatically duplicates the function input that you entered for the subfunction as a function input **for the Main function**.

This is because from an external view, the Water goes into the whole system instead of only into the subfunction!

Click on the **ADD Transaction** menu item to add an output for your subfunction.

7.3.12 Tutorial12.php - <Main Function Name> – ADD Transaction

Screenshot

Making coffee - ADD Transaction

Supplier
Supplier of Input

Transaction
Function input / Transaction

Customer
of Input

Function output / Transaction

Customer of Output

Party X → Input → Function 1 → Output → Party Y

Supplier of Output

Transaction

Customer

Remember: **Supplier** -> **Transaction** -> **Customer**

Now, add another transaction to the subfunction that you have added in the previous steps. You already added a Function Input, now it's time to add a Function Output. Take another look at the reminder box at the bottom of the menu. The subfunction is **Boil Water**. What do you get if you enter the function input **Water** to the function **Boil water**? Why, of course **Boiled Water**!

Add a **Function Output** named **Boiled Water**.
A Function Output will have the Function to which the Transaction is added as its **Supplier**.
Therefore, we only need to fill in a **Customer** for this Transaction.
The Boiled Water will be held in a Kettle, so the **Customer** will be a **Kettle**.

Function to add Transaction to: Making coffee (Main Function) ▼

Transaction Type: Function Input ▼

Transaction Name:

Only fill in a Supplier if the Transaction Type is *Function Input*.
Transaction Supplier:

Only fill in a Customer if the Transaction Type is *Function Output*.
Transaction Customer:

Hint: Pick **Function to add Transaction to** = *Boil water*, **Transaction Type** = *Function Output*,
Transaction Name = *Boiled Water* and **Customer** = *Kettle*.

Figure 7.13: tutorial12.php

Short description

The twelfth page of the Tutorial lets the player add a function output, just like he/she did before when adding a function input.

Page Transcript

[image]

Now, add another transaction to the subfunction that you have added in the previous steps. You already added a Function Input, now it's time to add a Function Output. Take another look at the reminder box at the bottom of the menu. The subfunction is **Boil Water**. What do you get if you enter the function input **Water** to the function **Boil water**? Why, of course **Boiled Water**!

Add a **Function Output** named **Boiled Water**.

A Function Output will have the Function to which the Transaction is added as its **Supplier**.

Therefore, we only need to fill in a **Customer** for this Transaction.

The Boiled Water will be held in a Kettle, so the **Customer** will be a *Kettle*.

Function to add Transaction to: [drop-down box, options: all Main/Systemfunctions, all Subfunctions, all Sub-subfunctions]

Transaction Type: [drop-down box, options: Function Input, Function Output]

Transaction Name: [text box for input]

Only fill in a Supplier if the Transaction Type is *Function Input*.

Transaction Supplier: [text box for input]

Only fill in a Customer if the Transaction Type is *Function Output*.

Transaction Customer: [text box for input]

["ADD Transaction" button (Links to **tutorial13.php**)]

Hint: Pick **Function to add Transaction to** = *Boil water*, **Transaction Type** = *Function Output*, **Transaction Name** = *Boiled Water* and **Supplier** = *Kettle*.

7.3.13 Tutorial13.php - <Main Function Name> – Show Overview

Screenshot

Making coffee - Show Overview

Main Function Making coffee	
IN	OUT
Boil water: From: Tap Water	Boil water: To: Boil water Water

Subfunctions

Part of Making coffee: Boil water	
IN	OUT
From: Tap Water	To: Kettle Boiled Water

Marvelous, we have now added a Function Input and a Function Output to the subfunction. Now that you know how to ADD functions and transactions, let's go through the **removal** of functions and transactions.

Click on the **REMOVE Transaction** menu item to remove the subfunction input.

Figure 7.14: tutorial13.php

Short description

The thirteenth page of the Tutorial shows the model again. The function output is added to the subfunction and the Todo List is cleared, as the subfunction now has both a function input and a function output. The player is now instructed to remove a transaction.

A click on the *REMOVE transaction* menu item links to **tutorial14.php**.

Page Transcript

[function table for Main function]

[function table for Subfunction]

Marvelous, we have now added a Function Input and a Function Output to the subfunction.

Now that you know how to ADD functions and transactions, let's go through the *removal* of functions and transactions.

Click on the **REMOVE Transaction** menu item to remove the subfunction input.

7.3.14 Tutorial14.php - <Main Function Name> – REMOVE Transaction

Screenshot

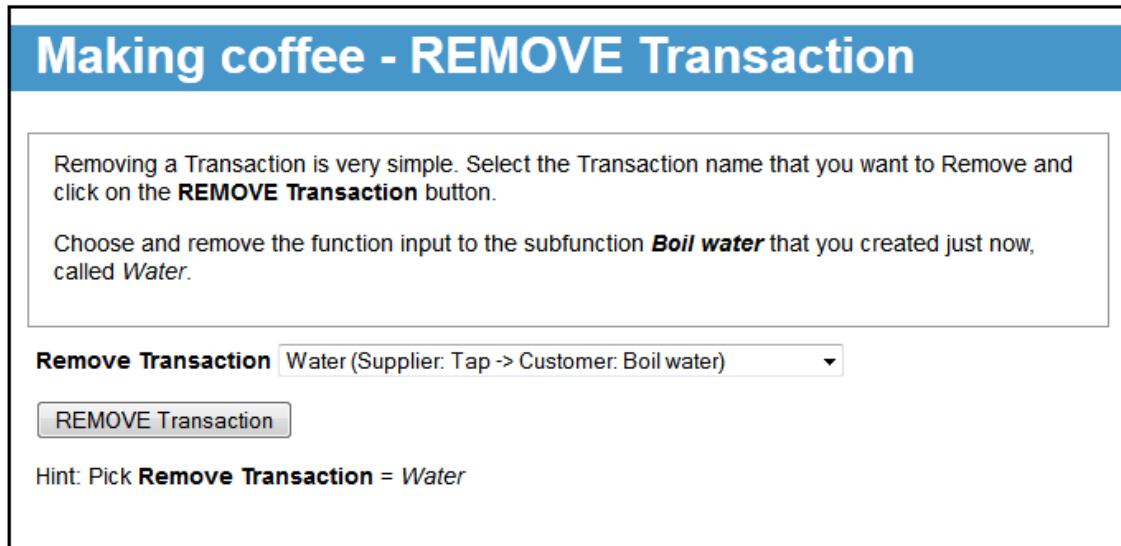


Figure 7.15: tutorial14.php

Short description

The fourteenth page of the Tutorial lets the player pick a transaction and remove it.

Page Transcript

Removing a Transaction is very simple. Select the Transaction name that you want to Remove and click on the **REMOVE Transaction** button.

Choose and remove the function input to the subfunction **Boil water** that you created just now, called *Water*.

Remove Transaction [drop-down box, options: all Transactions]

["REMOVE Transaction" button (Links to **tutorial15.php**)]

Hint: Pick **Remove Transaction** = *Water*

7.3.15 Tutorial15.php - <Main Function Name> – Show Overview

Screenshot

Making coffee - Show Overview

Main Function Making coffee	
IN	OUT
Boil water:	Boil water:

Subfunctions

Part of Making coffee: Boil water	
IN	OUT
No inputs defined yet	To: Kettle Boiled Water

Okay, now you have removed a Transaction. Take notice that the **Todo List** has been updated and provides you with a task to add a new function input to the subfunction. The last instruction is to **remove a function**.

Click on the **REMOVE function** menu item to remove the subfunction.

Todo List

Complete function table
Add a function input to subfunction
Boil water

Figure 7.16: tutorial15.php

Short description

The fifteenth page of the Tutorial is another Overview. The transaction/function input to the subfunction has been removed, and the Todo List is updated once more to urge the player to add a new function input, as the last one is removed now.

A click on the *REMOVE function* menu item links to **tutorial16.php**.

Page Transcript

[function table for Main function]

[function table for Subfunction]

Okay, now you have removed a Transaction. Take notice that the **Todo List** has been updated and provides you with a task to add a new function input to the subfunction. The last instruction is to **remove a function**.

Click on the **REMOVE function** menu item to remove the subfunction.

7.3.16 Tutorial16.php - <Main Function Name> – REMOVE Function

Screenshot

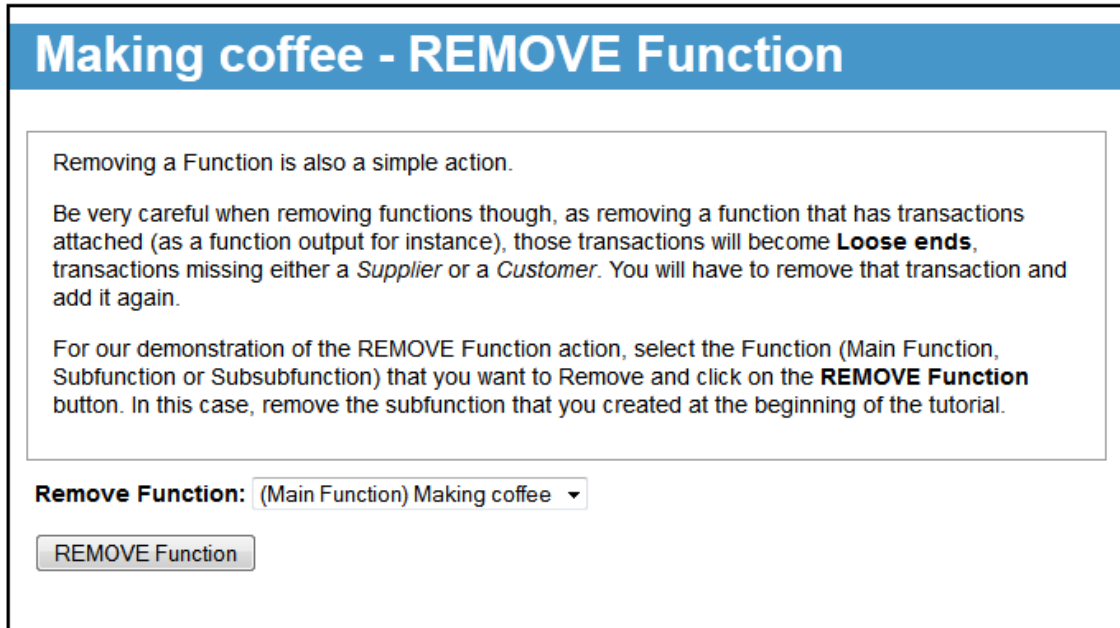


Figure 7.17: tutorial16.php

Short description

The sixteenth page of the Tutorial instructs the player about the danger of removing a function which has transactions attached. These transactions will become loose ends when the function is removed. The player is then instructed to remove the subfunction that he/she created before.

Page Transcript

Removing a Function is also a simple action.

Be very careful when removing functions though, as removing a function that has transactions attached (as a function output for instance), those transactions will become **Loose ends**, transactions missing either a *Supplier* or a *Customer*. You will have to remove that transaction and add it again.

For our demonstration of the REMOVE Function action, select the Function (Main Function, Subfunction or Subsubfunction) that you want to Remove and click on the **REMOVE Function** button. In this case, remove the subfunction that you created at the beginning of the tutorial.

Remove Function [drop-down list, options: Main Function, all Subfunctions, all Sub-subfunctions]

["REMOVE Function" button (Links to [tutorial17.php](#))]

7.3.17 Tutorial17.php - Tutorial – Getting started

Screenshot

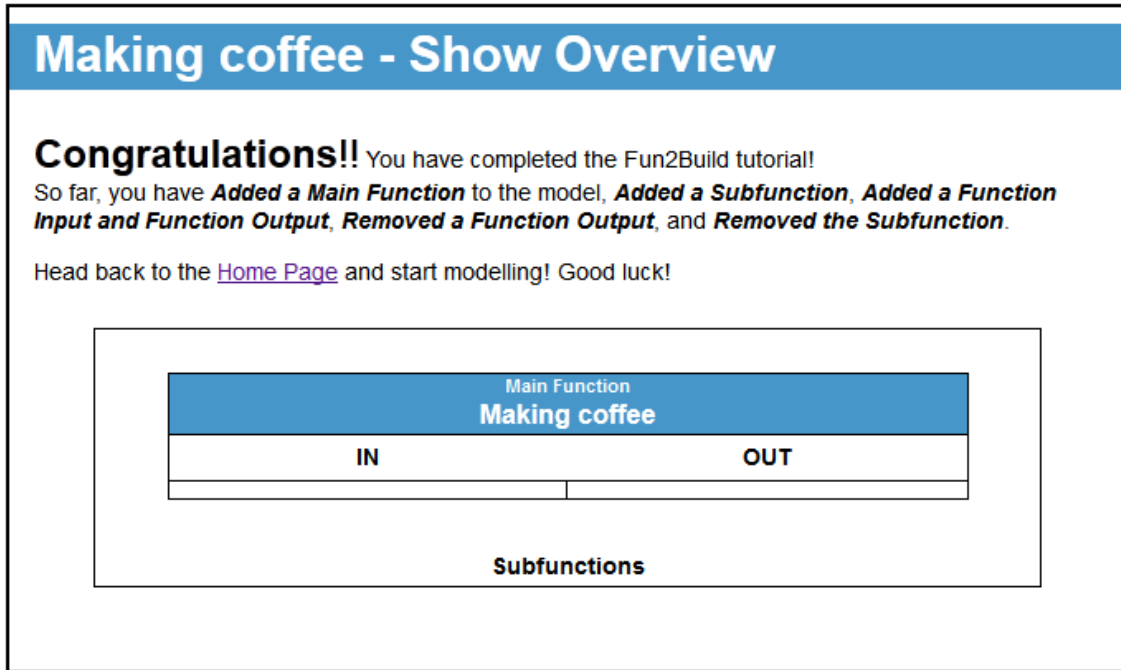


Figure 7.18: tutorial17.php

Short description

The seventeenth and last page of the Tutorial shows an empty function table, now that the subfunction has been removed. It congratulates the player on completing the tutorial and offers a hyperlink back to the Homepage to proceed with actual modelling a model of choice.

A click on the *Home Page* link in the text redirects the player to [index.php](#).

Page Transcript

Congratulations!! You have completed the Fun2Build tutorial!

So far, you have **Added a Main Function** to the model, **Added a Subfunction**, **Added a Function Input and Function Output**, **Removed a Function Output**, and **Removed the Subfunction**.

Head back to the Home Page and start modelling! Good luck!

[function table]

7.4. Start modelling

Instead of following the tutorial first, the player can also decide to start modelling right away. The pages and functions that are used in the modelling part of the Fun2Build game are identical to those used in the tutorial part of the game.

The main differences between both parts of the game are the structured guidance that the tutorial provides and the modelling part doesn't, and the fact that the player is free to choose whatever actions to perform whenever in the modelling part, leaving the player to choose his/her own way of modelling by experimenting with the functions at hand.

The pages of our game that are available in the modelling part of the game will now be elaborated upon:

7.4.1 startmodelling.php - Fun2Build - ADD Main function

Screenshot

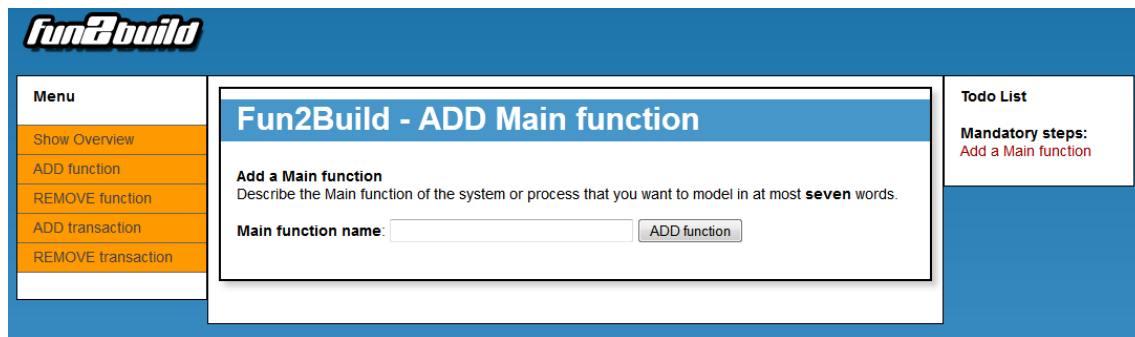


Figure 7.19: startmodelling.php

Short description

As is the case with the Tutorial, the graphical layout of the game remains the same; menu and menu items to the left, main content in the middle and the Todo List to the right of the screen. The first step in modelling is to enter a Main function and name it, so the player can start adding functions and transactions to it.

Page Transcript

Add a Main function

Describe the Main function of the system or process that you want to model in at most **seven** words.

Main function name: [text box for input] [“REMOVE Function” button (Links to [showoverview.php](#))]

7.4.2 showoverview.php - <Main Function Name> - Show Overview

Screenshot

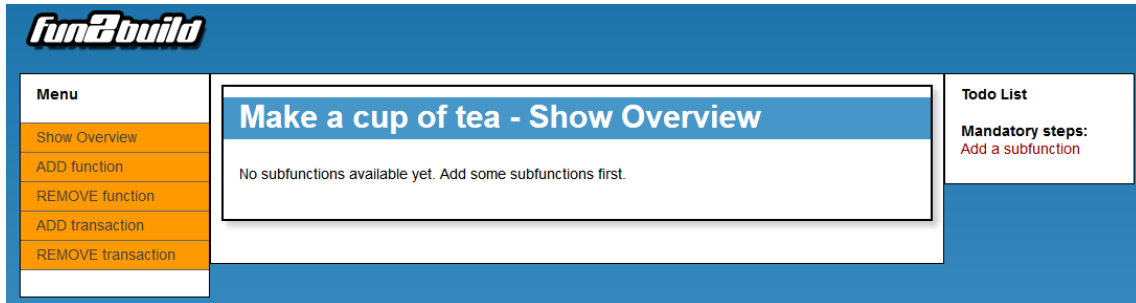


Figure 7.20: showoverview.php initial view

Description

The Show Overview page displays the contents of the entire model, its functions, and transactions and how they all connect together. In a sense, the Show Overview page is the main feedback method to the player, showing the current state of the model and indicating which parts of the model are still missing or should be adjusted.

The page also contains the programming code to process all actions that the player has performed, it adds the functions and transactions or removes them from the model. To this end, all action pages, like ADD function, redirect to the Show Overview page after submitting the data for that action.

All data that together forms the model is stored in an XML file called **Structure.xml**. The Show Overview page parses the information from the XML file into function tables which contain the function name, its optional parent function, its function inputs and its function outputs. It also checks whether system critical model XML nodes are present in the XML file, like the subfunctions node, which is the parent node of all subfunctions. If they are missing, Show Overview will add them to the XML file.

Todo List

While constructing the function tables for display, the Show Overview page interacts with the Todo List. The Show Overview page displays a textual message when an input or output is missing in the relevant function table, while the Todo List sums up all pending tasks that the player has to fulfil to reach the main goal, including those displayed in the function tables, as depicted in figure 7.21.

Part of Boil water: Turn kettle on	
IN	OUT
From: Fill kettle: <i>Water in kettle</i> From: Electricity socket: <i>Electricity</i>	No outputs defined yet

Todo List

Complete function table
Add a function output to sub-subfunction
Turn kettle on

Complete function table
Add a function input to Boil Water

Figure 7.21: Todo List interaction on the Show Overview page

There are three kinds of sub-goals that the Todo List urges the player to complete, providing the player with an incentive to perform iteration by executing the correct menu action to complete the sub-goals.

- **Mandatory sub-goals**

Mandatory goals can be set for the player to provide a general guideline of the tasks at hand, without imposing a direct list of actions to the player (like 1. Click the Add function button, 2. Input a name, etc.). Mandatory sub-goals are usually presented to the player at the beginning of the modelling process and disappear from the list once completed.

An example is the mandatory sub-goal *Mandatory steps* with the accompanying task Add a subfunction, which is displayed directly after adding a Main function to the model, as displayed in figure 7.22.

Todo List

Mandatory steps:
Add a subfunction

Figure 7.22: Todo List mandatory sub-goal

- **Missing inputs/outputs**

According to the framework rules, a function should have at least one function input and one function output. The Todo List checks the model after the completion of each action to see whether the model still complies with these rules. If an input or output is missing from a function, it will be shown in both the relevant function table at the Show Overview screen and in the Todo List, which states the sub-goal *Complete the function table* and the accompanying task *Add a function input/output to sub/sub-subfunction <function name>*, as seen in the example in figure 7.21.

- **Loose ends**

The third kind of sub-goal that the Todo List keeps track of together with the Show Overview screen, is the checking for Loose ends. As explained before, a loose end is a transaction that misses either a supplier or customer. When a Loose end is detected, for instance after removing a function that had a transaction connected to it, after which the transaction becomes a loose end, the transaction is listed in the Todo List as a warning. It is already explained that the possibility might occur that the modeller wants to add a loose end to the model on purpose, which is why it is shown as a warning and not as a sub-goal that the player has to solve by all means. An example of a loose end and how it is displayed in both the Show Overview function tables and the Todo List can be seen in figure 7.23.

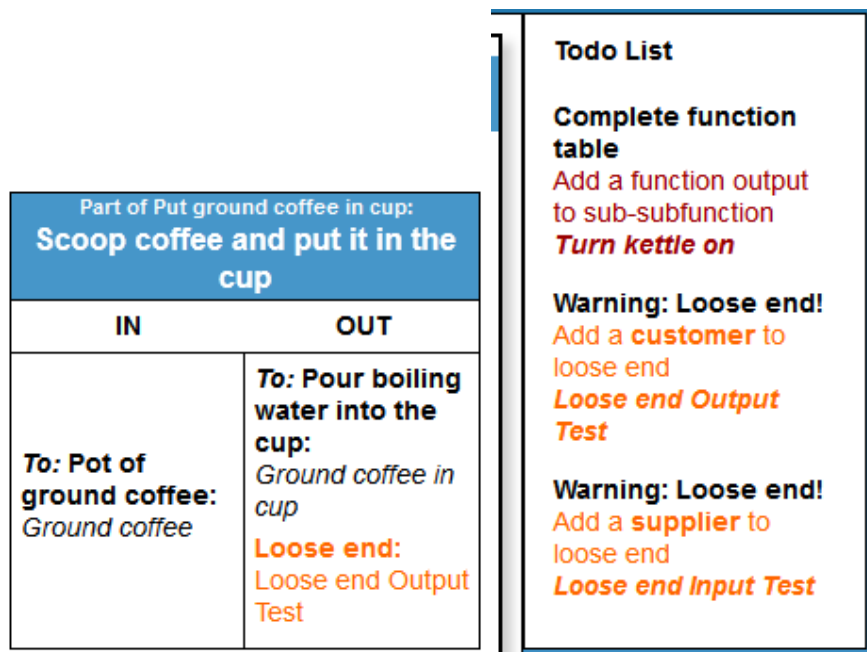


Figure 7.23: Todo List – Loose ends

7.4.3 addfunction.php - <Main Function Name> - ADD Function

Screenshot

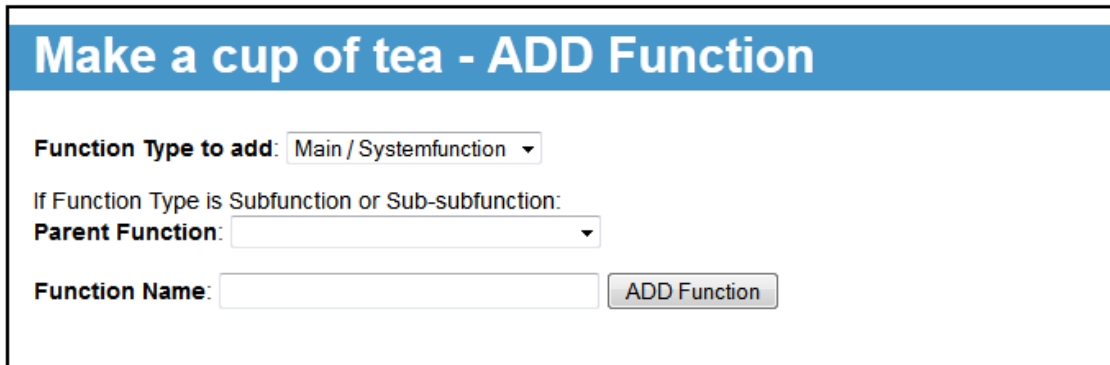


Figure 7.24: Add Function

Description

The Add Function page adds a function to the model.

Page transcript

Function Type to add: [drop-down list, options: Main/Systemfunction, Subfunction, Sub-subfunction]

If Function Type is Subfunction or Sub-subfunction:

Parent Function: [drop-down list, options: Main Function, all Subfunctions]

Function Name: [text box for input] ["ADD Function" button (Links to [showoverview.php](#))]

7.4.4 removefunction.php - <Main Function Name> - REMOVE Function

Screenshot

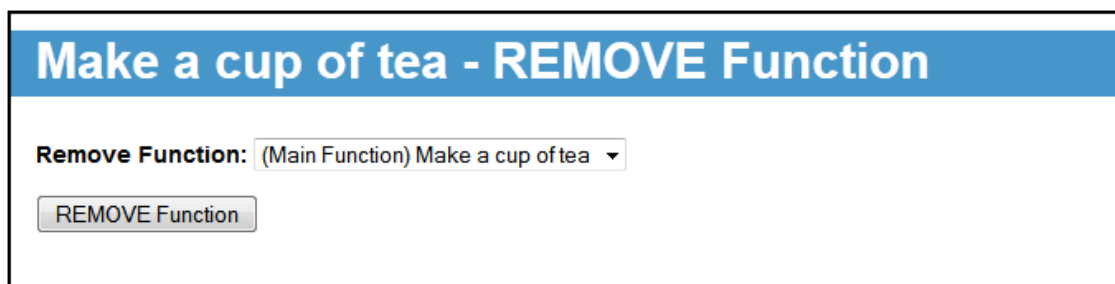


Figure 7.25: Remove Function

Description

The Remove Function page lets the player remove a function from the model.

Page transcript

Remove Function: [drop-down list, options: Main Function, all Subfunctions, all Sub-subfunctions]

["REMOVE Function" button (Links to [showoverview.php](#))]

7.4.5 addtransaction.php - <Main Function Name> - ADD Transaction

Screenshot

Make a cup of tea - ADD Transaction

Supplier
Supplier of Input

Transaction
Function input / Transaction

Customer
of Input

Function output / Transaction

Customer of Output

Party X → Input → Function 1 → Output → Party Y

Supplier of Output

Transaction

Customer

Remember: **Supplier -> Transaction -> Customer**

Function to add Transaction to:

Transaction Type:

Transaction Name:

Only fill in a Supplier if the Transaction Type is *Function Input*.

Transaction Supplier:

Only fill in a Customer if the Transaction Type is *Function Output*.

Transaction Customer:

Figure 7.26: Add Transaction

Description

The Add Transaction page lets the player add a transaction to the model.

Page transcript

[image]

Remember: **Supplier -> Transaction -> Customer**

Function to add Transaction to: [drop-down list, options: Main Function, all Subfunctions, all Sub-subfunctions]

Transaction Type: [drop-down list, options: Function Input, Function Output]

Transaction Name: [text box for input]

Only fill in a Supplier if the Transaction Type is *Function Input*.

Transaction Supplier: [text box for input]

Only fill in a Customer if the Transaction Type is *Function Output*.

Transaction Customer: [text box for input]

["ADD Transaction" button (Links to **showoverview.php**)]

7.4.5 removetransaction.php - <Main Function Name> - REMOVE Transaction

Screenshots

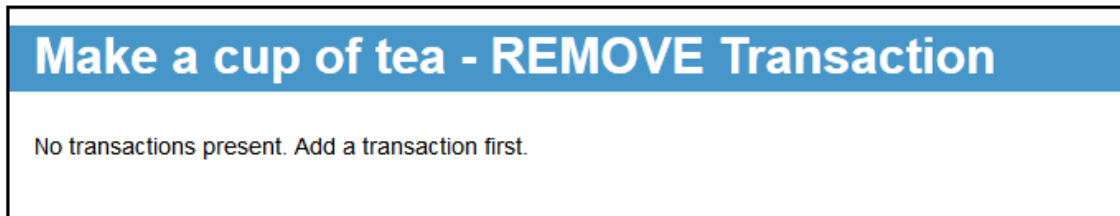


Figure 7.27: Remove Transaction with no transactions present



Figure 7.28: Remove Transaction with transactions present in the model

Description

The Remove Transaction page lets the player remove a transaction from the model. If there are no transactions present to remove, it will inform the player of that.

Page transcript

Remove Transaction [drop-down list, options: all Transactions]

["REMOVE Transaction" button (Links to **showoverview.php**)]

8. Conclusion

We started off this thesis with the intention to design a theoretical framework based on game-like elements to guide a player through a Serious Game in which an existing system or process is declaratively specified through functional decomposition. Following the completion of the framework, we built a prototype of such a game, based on the principles of the framework. In doing so, we have fulfilled the research goal of this thesis. This final chapter discusses the outcome of the research, the final conclusion and suggestions for future research.

8.1 Summary

After deep-diving into the subjects of Serious Games, iteration, functional decomposition and the Chinese Boxes principle, we combined aspects from all of these subjects to form a theoretical framework. This framework provides rules and guidelines to build a Serious Game which enables its player to construct a model that contributes to the functional specification of a system or process. Our version of the framework is specially tailored to suit our research goal, but with some adjustments, the framework can be implemented to construct a Serious Game which enables modelling of different situations.

A recollection of the framework's guidelines as discussed in chapter 6, Creating the Game, is given:

Technical Environment

The Serious Game will be an accessible web application (based on PHP and XML) that will be accessed through a web browser.

Gaming Environment

The gaming environment will be displayed in the web browser and will be visually attractive.

Goals, sub-goals and tasks

The main goal of the game is to build a model that contributes to a functional specification of a certain system that is modelled.

The main goal can be broken down into sub-goals, which can be achieved by completing the tasks that are linked to the sub-goals.

The main goal and the sub-goals of the game will be communicated to the player in a clear way.

Game elements

The game will present the player with several game elements, which represent elements within the concepts of functional specification and functional decomposition.

Game mechanics and actions

The player will be able to interact with the game environment through actions which allows him/her to manipulate the game elements.

Guidance and Feedback

The player will receive visual and textual feedback while playing the game, to guide him/her towards completing the right sub-goals and the main goal.

Graphical User Interface (GCI)

The GUI elements which enables the player to perform actions and provide feedback for the player to process, will be positioned on screen in a logical manner.

Procedure

The game provides a tutorial in which the player learns how to use the game as a modelling tool, before modelling themselves.

These framework guidelines were then implemented in a Serious Game called the Fun2Build game, as a prototype that acts as a proof of concept for the framework.

The Fun2Build game provided the player with a guided walkthrough through the game, explaining the concepts of models, the functional specification and the use of the built in actions to construct a complete and valid model.

8.2 Final Conclusion

The main research goal that we posed in the introductory chapter was described as *the design and construction of an operational functional specification game*.

To achieve this, two sub-goals were defined that had to be fulfilled:

1. The analysis and design of a theoretical framework based on game-like elements to guide a player through a Serious Game in which an existing system is declaratively specified through functional decomposition. The guidance is provided in such a way so that iteration is facilitated.
2. The implementation of this framework in the form of a Serious Game prototype to demonstrate a basic showcase of the framework's rules and guidelines.

Both of these sub-goals have been fulfilled and thoroughly documented in previous chapters, and therefore we can say that we have achieved our main research goal.

Implications

The results from the research done for this thesis are not ground-breaking or totally innovative in the sense that it does not conjure an entire new approach to method engineering and functional specification. The framework and the prototype game make use of concepts that haven't been combined that much before, though, and therefore we believe that it adds new insights and approaches to the development of Serious Games for specification.

Each of the chapters involving the theoretical components on which the framework was based can act as sources for information for future research. This thesis is also one of the first (maybe even the first) works in which the Chinese Boxes principle is combined with functional specification.

The framework that we have proposed and elaborated upon can be used for further research in the field of (iterative) method engineering and Serious Games development, providing researchers with a baseline set of rules and guidelines to refine even more efficient and effective frameworks.

8.3 Recommendations for future research

This thesis provided researchers in the field of Serious Games development with some guidelines on how to find principles on which to base a game. It also provides a proof of concept prototype to see how all that theory can be put into practice.

There are certain topics that aren't addressed in this thesis, but which can be good starting points for further research.

An example is the theoretical framework that we have constructed in this thesis. The rules and guidelines are specifically tailored to suit the purpose of modelling a Serious Game concerning functional specifications. Such a framework could be developed further into a more general version of the framework, so that the rules can apply to each of the moments in systems development that requires user input (e.g. during requirements elicitation, or creating user stories, or UX/GUI design). The resulting framework could be used by researchers who want to further investigate the usage of Serious Games for system development purposes, but also as a practical approach for information architects to design applications or games by.

A second topic for future research can be the further development of the Fun2Build game mechanics and to refine these. An example is the Todo list, that checks for three types of sub-goals that have to be achieved during modelling, checking the mandatory sub-goals, the checking on function inputs and outputs and checking for loose ends. The game mechanics can be developed to also incorporate the logical checking of a model, as described in chapter 4, while explaining the Chinese Boxes principle.

8.4 Final words

We hope that you enjoyed reading this thesis and that you find it a useful addition to the field of Information Science. Serious Gaming is a big, upcoming subject in the field of systems engineering, and we expect to see lots of movement in this area in the coming years. Hopefully in the future, Serious Gaming, its principles and its application in technical education and training will lead to a more accessible IT landscape in which we can all develop and share technical knowledge with others more easily, more attractive and more effective.

Sources

- Corti06 Corti, K. *Games-based Learning: a serious business application*. PIXELearning.com White paper, 2006.
- Finkelstein96 Finkelstein, A. and Dowell, J. *A Comedy of Errors: the London Ambulance Service case study*. In Proceedings of the 8th International Workshop on Software Specification and Design, pp. 2-4, 1996.
- Gartner08 Shiffler, G. *Forecast: PC Installed Base, Worldwide, 2004-2012*. Gartner, Inc., 2008.
- Garris02 Garris, R. et al. *Games, Motivation, and Learning: A Research and Practice Model*. In Simulation & Gaming, Volume 33, Issue 4, pp. 441-467, December 2002.
- Gros07 Gros, B. *Digital Games in Education: The Design of Games-Based Learning Environments*. In Journal of Research on Technology in Education, 40(1), 23-38, 2007.
- Hoppen08 Hoppenbrouwers, S.J.B.A. et al. *Method Engineering as Game Design: an Emerging HCI Perspective on Methods and CASE Tools*. In Workshop proceedings of EMMSAD08: Exploring Modeling Methods for Systems Analysis and Design affiliated to CAiSE08, Montpellier, France. Citeseer, 2008.
- Hoppen09 Hoppenbrouwers, S.J.B.A. et al. *Setting Rules of Play for Collaborative Modeling*. In International Journal of e-Collaboration, Volume 5, Issue 4, pp. 37-52, 2009.
- Jacobson92 Jacobson, I. et al. *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley, 1992.
- Järvinen07 Järvinen, A. *Games without Frontiers, Theories and Methods for Game Studies and Design*. PhD Thesis, University of Tampere, Finland, 2007.
- Keegan08 Keegan, V. *Games can have a serious role to play*. The Guardian, 11 December 2008.
- Kruchten00 Kruchten, P. *From Waterfall to Iterative Lifecycle – A tough transition for project managers*. Rational Software White Paper TP 173 5/00. December 2000.

- Kruchten04 Kruchten, P. *The Rational Unified Process: An Introduction, Third Edition*. Addison-Wesley, 2004.
- Leveson93 Leveson, N. G. and Turner, C. S. *An investigation of the Therac-25 Accidents*. In IEEE Computer, Volume 26, Issue 7, pp. 18-41, July 1993.
- Lions96 Lions, J. L. *Ariane 5, Flight 501 Failure, Report by the Inquiry Board*, 1996.
- Mellor94 Mellor, P. *CAD: computer-aided disaster*. In High Integrity Systems, Volume 1, Issue 2, pp. 101-156. Oxford University Press, 1994.
- Prensky01 Prensky, M. *Digital Game-Based Learning*. McGraw-Hill, 2001.
- Pressman05 Pressman, R.S. *Software Engineering – A Practitioner’s Approach, Sixth edition*, pp. 83. McGraw-Hill, 2005.
- Salomon91 Salomon, G. et al. *Partners in Cognition: Extending Human Intelligence with Intelligent Technologies*. In Educational researcher. Vol. 20, no. 3. pp. 2-9, 1991.
- Sarter97 Sarter, N.B. et al. *Automation Surprises*. Handbook of Human Factors & Ergonomics, Second edition, G. Salvendy (Ed.), Wiley, 1997.
- Stone05 Stone, R.J. *Serious Gaming: Virtual Reality’s Savior?*. In Proceedings of Virtual Systems and MultiMedia Conference, Ghent, Belgium, 2005. pp. 773-786, October 2005.
- Stone06 Stone, R.J., et al. *Serious Gaming Technologies Support Human Factors Investigations of Advanced Interfaces for Semi-Autonomous Vehicles*. In Proceedings of Virtual Media for Military Applications; NATO RTA HFM-136 Workshop, US Military Academy, West Point, NY, June 2006.
- Trusim TruSim Interactive Trauma Trainer.
<http://www.trusim.com/?page=Demonstrations>
- Tygron Tygron RO2 Watergame.
<http://www.watergame.nl>
- Verbraeck09 Verbraeck, A. *Serious Gaming: Tomorrow’s solution for today’s problems*. In Sun Microsystems: Noodpakket voor IT’ers Congres, 2009.
- Whitton07 Whitton, N. *Motivation and computer game based learning*. Proceedings of ASCILITE Singapore, pp. 1063-1067, 2007.
- Zyda05 Zyda, M. *From visual simulation to virtual reality to games*. In IEEE Computer, Volume 38, Issue 9, pp. 25-32, September 2005.

Appendix A – Source Codes

The source code for all pages of the Fun2Build game amounts to around 100 pages, therefore it would seem impractical to include them all in this appendix. All files will be supplied with this printed version of the thesis on an accompanying CD-ROM.

Some examples of programming code:

Index.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Fun2Build</title>
<link href="css/fun2build3col.css" rel="stylesheet" type="text/css" />
</head>

<body>
<div class="container">
  <?php include("header.php") ?>
  <div class="menubar">
    <h4>Menu</h4>
    <ul class="nav">
      <li><a href="#">Show Overview</a></li>
      <li><a href="#">ADD function</a></li>
      <li><a href="#">REMOVE function</a></li>
      <li><a href="#">ADD transaction</a></li>
      <li><a href="#">REMOVE transaction</a></li>
    </ul>
    <p>&nbsp;</p>
  </div>
  <div class="content">
    <div class="cadre">
      <h2 class="title">Welcome to the Fun2Build Game!</h2>
      <div class="entry">
        <p><em>Welcome to the Fun2Build Game!</em><br />
          In this game, we will learn you how to construct and refine a model for the functional specification of a
system or a process. Don't be scared if that sounds technical, it's actually quite easy!</p>
        <p><strong>What is a model?</strong><br />
          A model is a graphical representation of information, in this case about the structure of a specific
system or process. The term used to construct a model is called <strong>modelling</strong>.</p>
        <p align=center>Example of a simple model:<br />
          </p>
        <p><strong>What is a functional specification?</strong><br />
          A functional specification is a document which helps the system development team to develop the right
functionality of the system. It focuses on the <strong>functional process of the system</strong>and how it
works.</p>
        <p><em>The model that we will make during this game will tell the team what <em><strong>YOU
</strong></em>want it to be able to do! </p>
        <p><em>Don't be afraid if you have never done this before, we will help you through this!<br />
          Click on the <strong>Follow the Tutorial</strong> button if this is your first time building a functional
specification, or<br />
          Click on the <strong>Start modelling</strong> button to start right away!</p>
        <p><a href="tutorial.php">
```



```

        <button>Follow the Tutorial</button>
      </a></p>
      <p><a href="startmodelling.php">
        <button>Start modelling!</button>
      </a></p>
    </div>
  </div>
</div>
<div class="todolist">
  <h4>Todo List</h4>
  <p>Empty </p>
</div>
<div class="clearfloat"></div>
</div>
</body>
</html>

```

Showoverview.php

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Fun2Build</title>
<link href="css/fun2build3col.css" rel="stylesheet" type="text/css" />
</head>

<body>
<div class="container">
  <?php include("header.php"?>
  <div class="menubar">
    <h4>Menu</h4>
    <ul class="nav">
      <li><a href="showoverview.php">Show Overview</a></li>
      <li><a href="addfunction.php">ADD function</a></li>
      <li><a href="removefunction.php">REMOVE function</a></li>
      <li><a href="addtransaction.php">ADD transaction</a></li>
      <li><a href="removetransaction.php">REMOVE transaction</a></li>
    </ul>
  </div>
  <?php
    /* Load XML file */
    $doc = new DOMDocument('1.0');
    $doc->preserveWhiteSpace = false;
    $doc->load('Specification.xml');
    $doc->formatOutput = true;

    /* Construct an XPath for this document*/
    $xpath = new DOMXPath($doc);

    /* Help variables */
    $subfunctions = $doc->getElementsByTagName('subfunction');
    $subsubfunctions = $doc->getElementsByTagName('subsubfunction');
    $numsubfunctions = $subfunctions->length;
    $numsubsubfunctions = $subsubfunctions->length;
    $specification = $doc->getElementsByTagName('specification')->item(0);

    /* Addition of a Main function */
    if(isset($_POST['MainFunction'])){
      $addmain = $doc->createElement('name', (string)$_POST['MainFunction']);
      $doc->getElementsByTagName('systemfunction')->item(0)->appendChild($addmain);

```

```

        // Save document
        $doc->save('Specification.xml');
    }
    $mainfunctionname = $doc->getElementsByTagName('systemfunction')->item(0)-
>getElementsByTagName('name')->item(0)->nodeValue;

    /* Check if Subfunctions node is present */
    $subfunctionsnode = $doc->getElementsByTagName('subfunctions')->item(0);
    if (empty($subfunctionsnode)) {
        // Construct new subfunctions node
        $newsubnode = $doc->createElement('subfunctions');
        // Add subfunctions node to the model
        $specification->appendChild($newsubnode);
        // Save document
        $doc->save('Specification.xml');
    }
    /* Check if Subfunctions node is present */
    $subsubfunctionsnode = $doc->getElementsByTagName('subsubfunctions')->item(0);
    if (empty($subsubfunctionsnode)) {
        // Construct new sub-subfunctions node
        $newsubsubnode = $doc->createElement('subsubfunctions');
        // Add sub-subfunctions node to the model
        $specification->appendChild($newsubsubnode);
        // Save document
        $doc->save('Specification.xml');
    }
    /* Check if Transactions node is present */
    $transactionsnode = $doc->getElementsByTagName('transactions')->item(0);
    if (empty($transactionsnode)) {
        // Construct new transactions node
        $newtransnode = $doc->createElement('transactions');
        // Add transactions node to the model
        $specification->appendChild($newtransnode);
        // Save document
        $doc->save('Specification.xml');
    }

    /* Get values from previous page, construct function and append it to the model */
    if(isset($_POST['FunctionType'])){
        // If function added is a main function
        if ($_POST['FunctionType'] == 'mainfunction') {
            //echo "MAIN ADDED";
        }
        // If function added is a subfunction
        if ($_POST['FunctionType'] == 'subfunction') {
            // Construct new subfunction
            $newsubfunction = $doc->createElement('subfunction');
            // Construct new name node and append it to newsubfunction
            $subfunctionname = $doc->createElement('name', (string)$_POST['FunctionName']);
            $newsubfunction->appendChild($subfunctionname);
            // Construct new parent function node and append it to newsubfunction
            $subfunctionparent = $doc->createElement('parent-function',
(string)$_POST['ParentFunction']);
            $newsubfunction->appendChild($subfunctionparent);
            // Add new function to list of functions
            $subfunctionsnode->appendChild($newsubfunction);
            // Save document
            $doc->save("Specification.xml");
        }
        // If function added is a sub-subfunction
        if ($_POST['FunctionType'] == 'subsubfunction') {
            // Construct new sub-subfunction

```

```

        $newsfunction = $doc->createElement('function');
        // Construct new name node and append it to newsfunction
        $functionname = $doc->createElement('name',
(string)$_POST['FunctionName']);
        $newsfunction->appendChild($functionname);
        // Construct new parent function node and append it to newsfunction
        $functionparent = $doc->createElement('parent-function',
(string)$_POST['ParentFunction']);
        $newsfunction->appendChild($functionparent);
        // Add new function to list of functions
        $functionsnode->appendChild($newsfunction);
        // Save document
        $doc->save("Specification.xml");
    }
}

/* Get values from previous page, construct transaction and append it to the model */
if(isset($_POST['TransactionName'])){
    // Construct new transaction
    $newtransaction = $doc->createElement('transaction');
    // Construct new name node and append it to newtransaction
    $transactionname = $doc->createElement('name', (string)$_POST['TransactionName']);
    $newtransaction->appendChild($transactionname);
    // Construct new customer name and supplier name nodes
    if($_POST['TransactionType'] == 'input') { // Handle function inputs
        if(isset($_POST['TransactionFunction'])){
            $transactioncustomername = $doc->createElement('name',
(string)$_POST['TransactionFunction']);
        }
        $transactionsuppliername = $doc->createElement('name',
(string)$_POST['TransactionSupplier']);
    }
    if($_POST['TransactionType'] == 'output') { // Handle Function outputs
        if(isset($_POST['TransactionFunction'])){
            $transactionsuppliername = $doc->createElement('name',
(string)$_POST['TransactionFunction']);
        }
        $transactioncustomername = $doc->createElement('name',
(string)$_POST['TransactionCustomer']);
    }
    // Construct the new transaction node
    $newtransactionsupplier = $doc->createElement('supplier');
    $newtransactioncustomer = $doc->createElement('customer');
    $newtransactionsupplier->appendChild($transactionsuppliername);
    $newtransactioncustomer->appendChild($transactioncustomername);

    $newtransaction->appendChild($newtransactionsupplier);
    $newtransaction->appendChild($newtransactioncustomer);

    // Add new function to list of functions
    $transactionsnode->appendChild($newtransaction);
    // Save document
    $doc->save("Specification.xml");
}

/* Get values from previous page and remove transaction from the model */
if(isset($_POST['TransactionRemove'])){
    $removename = $_POST['TransactionRemove'];
    // If the name of the transaction matches the removename, remove the transaction
    foreach ($transactionsnode->childNodes as $transaction) {
        if ($transaction->getElementsByTagName('name')->item(0)->nodeValue ===
$removename) {

```

```

        while ($transaction->hasChildNodes()){ // Remove all Transaction child nodes
            foreach($transaction->childNodes as $transactionchild) {
                while ($transactionchild->hasChildNodes()){ // Remove all
Transaction child child nodes
                    $transactionchild->removeChild($transactionchild-
>childNodes->item(0));
                }
            }
            $transaction->removeChild($transaction->childNodes->item(0));
        }
        // Remove the Transaction from the Transactions node
        $transactionsnode->removeChild($transaction);
    }
}
// Save document
$doc->save("Specification.xml");
}

/* Get values from previous page and remove (sub/sub-)function from the model */
if(isset($_POST['FunctionRemove'])){
    $removename = $_POST['FunctionRemove'];
    // If the name of a subfunction matches the removename, remove the subfunction
    foreach ($subfunctionsnode->childNodes as $subfunction) {
        if ($subfunction->getElementsByTagName('name')->item(0)->nodeValue ===
$removename) {
            while ($subfunction->hasChildNodes()){ // Remove all subfunction child nodes
                $subfunction->removeChild($subfunction->childNodes->item(0));
            }
            // Remove the Subfunction from the Subfunctions node
            $subfunctionsnode->removeChild($subfunction);
        }
    }

    // If the name of a subsubfunction matches the removename, remove the subsubfunction
    foreach ($subsubfunctionsnode->childNodes as $subsubfunction) {
        if ($subsubfunction->getElementsByTagName('name')->item(0)->nodeValue ===
$removename) {
            while ($subsubfunction->hasChildNodes()){ // Remove all Sub-subfunction
child nodes
                $subsubfunction->removeChild($subsubfunction->childNodes-
>item(0));
            }
            // Remove the Subsubfunction from the Subsubfunctions node
            $subsubfunctionsnode->removeChild($subsubfunction);
        }
    }
    // Save document
    $doc->save("Specification.xml");
}

echo "
<div class='content'>
<div class='cadre'>
    <h2 class='title'>$mainfunctionname - Show Overview</h2>
    <div class='box'>";

if ($numsubfunctions < 1)
    echo "<p>No subfunctions available yet. Add some subfunctions first.</p>";

else {
echo "<div class='functions'>
<table class='overview'>

```

```

<tr>
  <td colspan='2'>
    <table class='function' id='mainfunction'>
      <th colspan='2' class='title'><span class='parentname'>Main
Function</span><br/>". $mainfunctionname. "</th>
      <tr>
        <td class='inout'>IN</td>
        <td class='inout'>OUT</td>
      </tr>
      <tr>
        <td class='inputs'>";
foreach($subfunctions as $subfunction){
  /* Collect inputs */
  $subfunctionname = $subfunction->getElementsByTagName('name')->item(0)->nodeValue;
  $subinputquery = '//transaction[customer/name="'. $subfunctionname.'"]'; // Find all transactions that
have this subfunction as customer
  $subinputs = $xpath->query($subinputquery); // All subfunction inputs

  $subsubinputquery = '//subsubfunction[parent-function="'. $subfunctionname.'"]'; // Find subfunctions
for this subfunction
  echo "<span class='subname'>" . $subfunctionname . "</span><br/>";

  /* Find all inputs for this subfunction */
  foreach($subinputs as $subinput) {
    echo "<div class='item'><span class='supplier'><i>From: </i>". $subinput-
>getElementsByTagName('supplier')->item(0)->getElementsByTagName('name')->item(0)->nodeValue . "
</span><br/>
<span class='name'>" . $subinput->getElementsByTagName('name')->item(0)->nodeValue . "</span>
</div><br/><br/>";
  } // end foreach

  /* Inputs for this subfunction's subfunctions */
  $subsubs = $xpath->query($subsubinputquery);
  foreach($subsubs as $subsub) {
    $subsubname = $subsub->getElementsByTagName('name')->item(0)->nodeValue; // Get the
name of the sub-subfunction
    $subsubtransactionsxpath = '//transaction[customer/name="'. $subsubname . '"]';
    $subsubtransxpin = $xpath->query($subsubtransactionsxpath); // Find all transactions that
have this sub-subfunction as its customer / all subsub inputs

    if ($subsubtransxpin->length > 0) { // If there are indeed inputs for this sub-subfunction, show
these
      foreach ($subsubtransxpin as $subsubtransin) {
        $subinputsupp = $subsubtransin->getElementsByTagName('supplier')-
>item(0)->getElementsByTagName('name')->item(0)->nodeValue;
        if (empty($subinputsupp)) { // If it's a loose end, display it like one
          echo "<div class='item'><span class='looseend'><b>Loose end:</b><br/>" .
$subsubtransin->getElementsByTagName('name')->item(0)->nodeValue . "</span></div>";
        }
        else {
          echo "<div class='item'><span class='supplier'><i>From: </i>". $subsubtransin-
>getElementsByTagName('supplier')->item(0)->getElementsByTagName('name')->item(0)->nodeValue .
"</span><br/>";
          echo "<span class='name'>" . $subsubtransin->getElementsByTagName('name')->item(0)-
>nodeValue . "</span></div>";
        }
      }
    }
  } // end foreach sub-subfunctions
} // end foreach subfunctions

```

```

echo"</td>
<td class='outputs'>";

foreach($subfunctions as $subfunction){
    /* Collect outputs */
    $subfunctionname = $subfunction->getElementsByTagName('name')->item(0)->nodeValue;
    $suboutputquery = '//transaction[supplier/name="'. $subfunctionname.'"]'; // Find all transactions that
have this subfunction as supplier
    $suboutputs = $xpath->query($subinputquery); // All subfunction outputs

    $subsuboutputquery = '//subsubfunction[parent-function="'. $subfunctionname.'"]'; // Find
subfunctions for this subfunction

    echo "<span class='subname'>" . $subfunctionname . " :</span><br/>";

    /* Find all outputs for this subfunction */
    foreach($suboutputs as $suboutput) {
        echo "<div class='item'>
            <span class='customer'><i>To: </i>"
            . $suboutput->getElementsByTagName('customer')->item(0)->getElementsByTagName('name')-
>item(0)->nodeValue . "</span><br/>
            <span class='name'>" . $suboutput->getElementsByTagName('name')->item(0)->nodeValue .
"</span>
            </div><br/><br/>";
    } // end foreach

    /* Outputs for this subfunction's subfunctions */
    $subsubs = $xpath->query($subsuboutputquery);
    foreach($subsubs as $subsub) {
        $subsubname = $subsub->getElementsByTagName('name')->item(0)->nodeValue; // Get the name of the
sub-subfunction
        $subsubtransactionsoutxpath = '//transaction[supplier/name="'. $subsubname . '"]';
        $subsubtransxpout = $xpath->query($subsubtransactionsoutxpath); // Find all transactions that have
this sub-subfunction as its supplier / all subsub outputs

        if ($subsubtransxpout->length > 0) { // If there are indeed outputs for this sub-subfunction, show these
            foreach ($subsubtransxpout as $subsubtransout) {
                $subsubtransout = $subsubtransout->getElementsByTagName('customer')->item(0)-
>getElementsByTagName('name')->item(0)->nodeValue;
                if (empty($subsubtransout)) { // If it's a loose end, display it like one

                    echo "<div class='item'><span class='looseend'><b>Loose end:</b><br/>" . $subsubtransout-
>getElementsByTagName('name')->item(0)->nodeValue . "</span></div>";
                }
                else {
                    echo "<div class='item'><span class='customer'><i>From: </i>". $subsubtransout-
>getElementsByTagName('customer')->item(0)->getElementsByTagName('name')->item(0)->nodeValue .
"</span><br/>";

                    echo "<span class='name'>" . $subsubtransout->getElementsByTagName('name')->item(0)->nodeValue .
"</span></div>";
                }
            }
        } // end foreach sub-subfunctions
    } // end foreach subfunctions
} // end foreach subfunctions
echo "</td></tr></table></td>
</tr>
<tr>

    <td align='center'><b>Subfunctions</b></td>";
if ($numsubsubfunctions > 0) { // If at least one sub-subfunction present, display heading
    echo "<td align='center'><b>Sub-subfunctions</b></td>";
}

```

```

}
echo "</tr>
<tr>
<td>";

/* Subfunction function tables */
foreach($subfunctions as $subfunction){
/* Collect inputs and outputs */
$subfunctionname = $subfunction->getElementsByTagName('name')->item(0)->nodeValue;
$subparentname = $subfunction->getElementsByTagName('parent-function')->item(0)->nodeValue;
$subinputquery = '//transaction[customer/name="'. $subfunctionname.'"]'; // Find all transactions that have this
subfunction as customer
$suboutputquery = '//transaction[supplier/name="'. $subfunctionname.'"]'; // Find all transactions that have this
subfunction as supplier

$subinputs = $xpath->query($subinputquery); // All subfunction inputs
$suboutputs = $xpath->query($suboutputquery); // All subfunction outputs

$subsubinputquery = '//subsubfunction[parent-function="'. $subfunctionname.'"]'; // Find subfunctions for this
subfunction

/* Function table */
echo "<table class='function'>
<th colspan='2' class='title'><span class='parentname'>Part of " . $subparentname . " :</span><br/>" .
$subfunctionname . "</th>
<tr>
<td class='inout'>IN</td>
<td class='inout'>OUT</td>
</tr>
<tr>
<td class='inputs'>";
if ($subinputs->length < 1) {
    echo "No inputs defined yet";
}
/* Find all inputs for this function */
else {
    foreach($subinputs as $subinput) {
        echo "<div class='item'>
            <span class='supplier'><i>From: </i>". $subinput-
>getElementsByTagName('supplier')->item(0)->getElementsByTagName('name')->item(0)->nodeValue . "
            </span><br/>
            <span class='name'>" . $subinput->getElementsByTagName('name')->item(0)-
>nodeValue . "</span>
            </div><br/><br/>";
    } // end foreach
} // end else

/* Start finding inputs from subfunctions */
$subsubs = $xpath->query($subsubinputquery);
foreach($subsubs as $subsub) {
    $subsubname = $subsub->getElementsByTagName('name')->item(0)->nodeValue; // Get the
name of the subfunction
    $subsubtransactionsxpath = '//transaction[customer/name="'. $subsubname . '"]';

    $subsubtransxpinyin = $xpath->query($subsubtransactionsxpath);
    if ($subsubtransxpinyin->length > 0) { // If there are indeed inputs from subs
        echo "<span class='subname'>" . $subsubname . " :</span>";
        foreach ($subsubtransxpinyin as $subsubtransin) {
            $subsubinputsupp = $subsubtransin->getElementsByTagName('supplier')->item(0)-
>getElementsByTagName('name')->item(0)->nodeValue;
            if (empty($subsubinputsupp)) {

```

```

        echo "<div class='item'><span class='looseend'><b>Loose end:</b><br/>" . $subsubtransin-
>getElementsByTagName('name')->item(0)->nodeValue . "</span></div>";
    }
    else {
        echo "<div class='item'><span class='supplier'><i>From: </i>" . $subsubtransin-
>getElementsByTagName('supplier')->item(0)->getElementsByTagName('name')->item(0)->nodeValue .
"</span><br/>"; echo "<span class='name'>" . $subsubtransin->getElementsByTagName('name')->item(0)-
>nodeValue . "</span></div>";
    }
}
}
}
}
echo"</td>
<td class='outputs'>";
    if ($suboutputs->length < 1) {echo "No outputs defined yet"; }
    /* Find all outputs for this function */
    else {
        foreach($suboutputs as $suboutput) {
            echo "<div class='item'>
                <span class='customer'><i>To: </i>" . $suboutput->getElementsByTagName('customer')-
>item(0)->getElementsByTagName('name')->item(0)->nodeValue . "</span><br/>
                <span class='name'>" . $suboutput->getElementsByTagName('name')->item(0)-
>nodeValue . "</span></div><br/><br/>";
            } // end foreach
        } // end else

        /* Start finding outputs from subfunctions */
        $subsubs = $xpath->query($subsubinputquery);
        foreach($subsubs as $subsub) {
            $subsubname = $subsub->getElementsByTagName('name')->item(0)->nodeValue; // Get the
name of the subfunction
            $subsubtransactionsoutxpath = '//transaction[supplier/name="'. $subsubname . '"]';

            $subsubtransxpout = $xpath->query($subsubtransactionsoutxpath);
            if ($subsubtransxpout->length > 0) { // If there are indeed inputs from subs
                echo "<span class='subname'>" . $subsubname . " :</span>";
                foreach ($subsubtransxpout as $subsubtransout) {
                    $suboutputcust = $subsubtransout->getElementsByTagName('customer')-
>item(0)->getElementsByTagName('name')->item(0)->nodeValue;
                    if (empty($suboutputcust)) {
                        echo "<div class='item'><span class='looseend'><b>Loose
end:</b><br/>" . $subsubtransout->getElementsByTagName('name')->item(0)->nodeValue . "</span></div>";
                    }
                    else {
                        echo "<div class='item'><span class='customer'><i>To: </i>" .
$subsubtransout->getElementsByTagName('customer')->item(0)->getElementsByTagName('name')->item(0)-
>nodeValue . "</span><br/>";
                        echo "<span class='name'>" . $subsubtransout-
>getElementsByTagName('name')->item(0)->nodeValue . "</span></div>";
                    }
                }
            }
        }
    }
}
echo"</td></tr></table>";

} // end foreach

/* Sub-subfunction function tables */
if ($numsubsubfunctions > 0) {
    echo "</td><td>";

    foreach($subsubfunctions as $subsubfunction){
        /* Collect inputs and outputs */
        $subsubfunctionname = $subsubfunction->getElementsByTagName('name')->item(0)->nodeValue;

```



```

        $subsubparentname = $subsubfunction->getElementsByTagName('parent-function')->item(0)-
>nodeValue;
        $subsubinputquery = '//transaction[customer/name="'. $subsubfunctionname.'"]'; // Find all
transactions that have this subsubfunction as customer
        $subsuboutputquery = '//transaction[supplier/name="'. $subsubfunctionname.'"]'; // Find all
transactions that have this subsubfunction as supplier

        $subsubinputs = $xpath->query($subsubinputquery); // All subsubfunction inputs
        $subsuboutputs = $xpath->query($subsuboutputquery); // All subsubfunction outputs

        /* Function table */
        echo "<table class='function'>
<th colspan='2' class='title'><span class='parentname'>Part of " . $subsubparentname . " :</span><br/>
. $subsubfunctionname . "</th>
<tr>
        <td class='inout'>IN</td>
        <td class='inout'>OUT</td>
</tr>
<tr>
        <td class='inputs'>;
        if ($subsubinputs->length < 1) {echo "No inputs defined yet"; }
        /* Find all inputs for this function */
        else {
                foreach($subsubinputs as $subsubinput) {
                        echo "<div class='item'>";
                        $subsubinputsupp = $subsubinput->getElementsByTagName('supplier')-
>item(0)->getElementsByTagName('name')->item(0)->nodeValue;
                        if (empty($subsubinputsupp)) { // Loose end
                                echo "<span class='looseend'><b>Loose end:</b><br/>".
$subsubinput->getElementsByTagName('name')->item(0)->nodeValue . "</span>";
                        }
                        else {
                                echo "

                                <span class='customer'><i>From: </i>". $subsubinputsupp . "</span><br/>
                                <span class='name'>" . $subsubinput->getElementsByTagName('name')-
>item(0)->nodeValue . "</span>
                                </div>";
                        }
                } // end foreach
        } // end else
        echo"</td>
<td class='outputs'>";
        if ($subsuboutputs->length < 1) {echo "No outputs defined yet";}
        /* Find all outputs for this function */
        else {
                foreach($subsuboutputs as $subsuboutput) {
                        echo "<div class='item'>";
                        $subsuboutputcust = $subsuboutput->getElementsByTagName('customer')->item(0)-
>getElementsByTagName('name')->item(0)->nodeValue;
                        if (empty($subsuboutputcust)) { // Loose end
                                echo "<span class='looseend'><b>Loose end:</b><br/>". $subsuboutput-
>getElementsByTagName('name')->item(0)->nodeValue . "</span>";
                        }
                        else {
                                echo "<span class='customer'><i>To: </i>". $subsuboutputcust . "</span><br/>
                                <span class='name'>" . $subsuboutput->getElementsByTagName('name')->item(0)->nodeValue .
"</span>
                                </div>";
                        }
                } // end foreach
        } // end else
} // end else

```

```
                echo"</td>
                </tr>
            </table>";

            } // end foreach
        }
        echo "</td></tr></table>
        </div>"; // closing DIV functions
    } // closing else?>
        </div>
    </div>
</div>
<?php include("todolist.php")?>
<div class="clearfloat"></div>
</div>
</body>
</html>
```