Incorporating Functional Design Patterns In Software Development

Master Thesis
Business Information Technology

Yoran Maxim Bosman

31.08.2007

Under supervision of
University of Twente
Maria Laura Ponisio
Jos van Hillegersberg

Quinity B.V.
Jeroen Snijders
Robert Guitink

Revision History

Date	Description
28.03.2007	- Basic template
16.05.2007	- Concept/work in progress for review by Quinity supervisor
24.05.2007	- Concept/work in progress for review by University supervisors
01.06.2007	- Completely rewritten structure - Shortened introduction - Processed other comments and suggestions.
15.06.2007	- First step to incorporate Laura's feedback, making the thesis more scientific instead of a tutorial
20.06.2007	- Textual improvements - Started with conclusion - Removed some literature appendices
02.07.2007	- Laura's comments on the method. - Quinity's comments on the thesis. - Moved several steps from requirements to functional design phase
12.07.2007	- Lots of textual improvements (spelling, grammar) - Added case studies
23.08.2007	- Method in dependency graph form - Case studies revamped - Designer opinions - Discussion on case studies - Removed figure "distribute functional design document
31.08.2007	- Textual improvements (spelling, grammar) - Fixed errors in figures - Restructured and shortened some sections - Combined the example in one paragraph - Revamped validation and conclusion

Summary

When developing software, clients often ask for the same kind of functionality which solves a certain problem they are faced with. Reuse of solutions for recurring problems is possible at many levels. The functional level has not been given as much attention as the technical level. It is wasteful to engineer similar functionality from scratch every time it is required. Therefore the concept of Functional Design Patterns was identified to describe recurring problems and solutions of applications.

From interviews it became clear that currently there is no unified way in which the Functional Design Patterns are applied during the software development process. Although a small library of Functional Design Patterns exist, it is not clear how to integrate them in the development process. Every functional designer decides for himself how and when the pattern repository is consulted and which parts to use.

This thesis proposes a method for the incorporation of Functional Design Patterns in the software development process. The goal of the method is to enable functional and technical designers to make more efficient use of Functional Design Patterns at different phases of development. The method does not focus solely on functional design but ranges from acquisition all the way to maintenance.

The method was developed by combining the state of the practice with an analysis of the state of the literature regarding the integration of design patterns. It provides guidelines for recognizing the applicability of patterns using conceptual decomposition, pattern selection, the combination of multiple patterns in a functional design, the transition to technical design and defines the responsible persons. The result is a systematic method that clarifies the when, where, how and what questions regarding Functional Design Patterns and software development for all parties involved.

This systematic approach to develop software using Functional Design Patterns is explored and demonstrated with an example.

Application of the method in real world projects was not feasible during the execution of this research, therefore empirical evaluation of the presented method remains future work. The method is validated by matching it against the functional reuse approaches of other organisations and a review by designers which showed their support for the method.

We believe that the use of the presented method to incorporate Functional Design Patterns in software development will lead to an improved and more efficient use of Functional Design Patterns in particular at Quinity. Novice designers will know when, where and how to use Functional Design Patterns in the development process thereby bridging a large gap between them and more experienced designers.

Acknowledgements

"and times move so fast, now there does not seem to be any
and once it felt that there was more than plenty"
I Am Kloot – Astray

First and foremost I would like to express my gratitude to my parents who made it possible to attend University and always stood behind me and my decisions.

I would like to thank all my supervisors, especially Laura who very patiently spent hours answering questions and giving feedback.

My sincere appreciation goes to Quinity for providing a very pleasant work environment and all the great colleagues I have had the opportunity to meet, in particular Fleur, Jacob and Wouter; I wish you all the best. Huge thanks to all my friends, in no particular order... Tom, Hugo, Stephan, Jos, Rutger, Mike, Sjoerd, Dian, Frank, Ron and the ones I forgot. We always have a great time whatever we do.

A special thanks goes to the organisations who allowed a peek into their software development kitchen. Last but not least special thanks fly out to all musicians who provided me with listening pleasure, deprived from which I would certainly have gone insane.

Table Of Contents

1. Introduction

1.1 Background

"A pattern is an idea that has been useful in one practical context and will probably be useful in others" [11] [1]

Patterns make it easier to reuse successful designs and expressing these techniques allows other developers to make use of proven solutions [12].

Reuse of solutions is possible at different levels of abstraction and many different types of patterns have emerged in the past [2] [12] [13] [41] [11]. The technical level gained a lot of popularity through the book of Gamma et al. [12]. Reuse of functionality on the other hand has not been given as much attention.

Recurring problems can be identified when developing custom software systems. Clients often demand similar types of functionality. Until recently there were no patterns that allowed the description of recurring functionality in applications. Administrative software for banking and insurance organisations for example share many of the same functions for working with personal records and time dependent loans and insurance policies [17] This is the reason Functional Design Patterns were introduced by Guitink [17] and described in detail by Snijders [37].

Functional Design Patterns describe recurring functionality of applications where functionality is defined as:

"All behaviour of an information system concerning the storage, manipulation and display of data"

In previous studies the foundation for Functional Design Patterns were laid out. The reasoning behind Functional Design Patterns, their notation and relation were all explored [37][34][26]. The most important reasons behind Functional Design Patterns are the improvement of the predictability of the software development process and the delivery of high quality software [34]. Because patterns are proven solutions, quality can be improved. The risk of creating something new and forgetting a critical component is lower. Predictability increases because the complexity of the functionality and the time it takes to implement are already known due to past experience. In [26] an approach to implementing the patterns in technical design documents and programming code is presented.

The research is conducted at Quinity B.V. located in Utrecht, The Netherlands. Quinity is a supplier of custom software applications based on internet technology. Quinity's focus lies with applications in the insurance and banking domain. The software Quinity designs and builds is partly based on a framework. The framework is extended with custom parts and components, which are developed for each client separately. The design process is streamlined by using their own development method

[1] Numbers in brackets refer to entries in the bibliography at the end

which amongst other techniques includes reuse of software and automatic code generation.

The development method is a combination of Linear Application development (LAD) [10] and Dynamic Systems Development Method (DSDM) [38]. Quinity uses the clearly phased structure of LAD but also design and build iterations as proposed by DSDM.

1.2 Problem Statement

Although patterns in essence already capture reusable solutions and have been around for years, less devotion has been given to develop methods to truly integrate (functional) design patterns throughout the software development life cycle.

After Snijders [37] there have been a handful of other researchers [34][26][27] who attempted to broaden the knowledge of Functional Design Patterns. As for the reuse of Functional Design Patterns in software development the work of Van Helden en Reyngoud [34] further specified introduced specialisation and generalisation of similar patterns. Next to this, they also made a head start with describing the use of Functional Design Patterns in the functional design phase and a possible method to combine multiple patterns.

They noted that one of the most important questions left unanswered is how to allow the use of Functional Design Patterns in software development.

Even though a catalogue of smaller Functional Design Patterns has been described in [32] and well-documented domain level Functional Design Patterns exist in various internal Quinity documents, the fundamental problem Quinity is experiencing lies in the fact that at this moment there is no structured way of reusing the developed Functional Design Patterns. Both designers and application developers do not know how to make optimal use of Functional Design Patterns during software development.

1.3 Research Questions

In the context of making the best use of Functional Design Patterns, the problem statement leads to the following main question:

"How can reuse of Functional Design Patterns be incorporated in the software development process?"

To discover a satisfying answer to the research subject the following research questions were developed:

- What is the current way Quinity uses Functional Design Patterns in software development?

To be able to define what should be improved, it is a requisite to find out the current state of the practice.

- To what extent and in which way is pattern reuse implemented in current software development methods?
- How can the current Quinity Method be extended with Functional Design Patterns?
- When should the patterns be applied?
- Which elements should be used?
- How do we recognise the possibilities for reuse of Functional Design Patterns?
- How can we combine multiple patterns?
- In which context can Functional Design Patterns be used?

1.4 Contribution

The contribution of this thesis can be summarised as follows:

- An analysis of existing methods to integrate design patterns in software development.
- A systematic and practical method for the incorporation of Functional Design Patterns in the software development process.
- A survey of the state of the practice regarding functional reuse at external organisations.

1.5 Approach

To engineer a method for the incorporation of Functional Design Patterns in the software development process the following steps were taken.

To fully understand Functional Design Patterns we attend review sessions for the pattern “time dependence of data” [6] and create new Functional Design Pattern for Authorisation ourselves [3].

The desired state of the practice is found by enquiring functional designers at Quinity on their expectations and experiences with Functional Design Patterns and by means of a thorough literature study on current pattern integration methods. These will lead to criteria the method for incorporating Functional Design Patterns should meet.

The results from the interviews and the literature study are merged to establish a theoretical method for making more efficient use of Functional Design Patterns in software development.

After developing the method, it is validated by matching it against structural reuse methods at external companies, designer opinions and the original criteria.

1.6 Thesis Structure

The structure of the thesis is as follows:

Chapter 2 (p. 11) provides an extensive overview of the development method at Quinity. It describes Functional Design Patterns in detail, the process and the link between these two. It lists the criteria a development method for the incorporation of Functional Design Patterns in the development process needs to adhere to.

Chapter 3 (p. 21) analyses existing methods that attempt to systematically reuse patterns in the software development process. To the best of our knowledge no earlier research exists in this area.

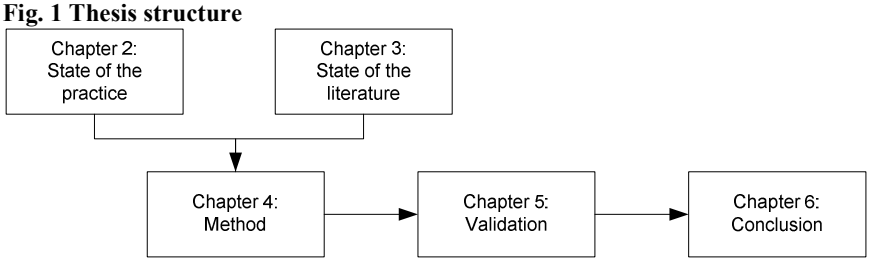
Chapter 4 (p. 30) contains a specifically engineered development method with guidelines to incorporate Functional Design Patterns in software development.

Chapter 5 (p. 57) validates the developed method in various ways.

Chapter 6 (p. 75) contains the conclusion of the thesis. It presents the results, practical implications and ends with new research questions that are currently left unanswered.

The Appendices (p. 85) contain examples of Pattern Oriented Analysis And Design's diagrams, the case study interviews and an example of visual Functional Design Pattern composition 0

Fig. 1 Thesis structure



2. State Of The Practice

This chapter introduces the state of the practice concerning Quinity's current view on software development, Functional Design Patterns and the link between them. First Quinity's development method and phased approach is described. After that, Functional Design Patterns are explored and in the last part the current link between Functional Design Patterns and software development is discussed, which leads to criteria for a new method.

2.1 Software Development At Quinity

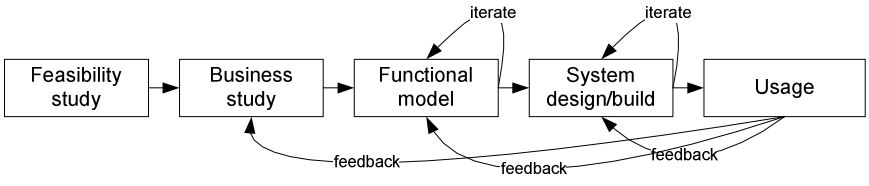
A myriad of different software development methods exists. Quinity's development method is based on a combination of the Linear Application Development [10] and iterative Dynamic Systems Development Method [38].

Linear Application Development is a methodology based on the System Development Method. LAD is a method very suitable for the development of complex administrative systems. It is essentially a waterfall model consisting of five phases; definition study, basic design, detailed design, realisation and implementation. In all these phases there are linear and parallel tracks which together result in the completed information system. LAD does not feature iteration. The next phase only begins after the preceding phase is completed.

Fig. 2 LAD overview

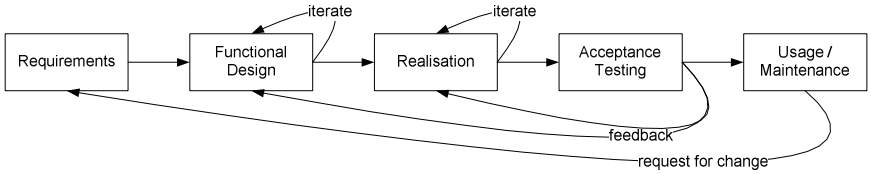
The Dynamic Systems Development Method is an agile development method originally based on Rapid Application Development. DSDM is meant to iteratively and incrementally develop interactive information systems that meet the business needs. It makes use of continuous user involvement and allows requirements to change over time. DSDM works with time boxes to prevent projects going over budget and exceeding their expected development time. DSDM stresses that nothing is built perfectly the first time, but states that early delivery to end-users can improve the quality in the end [38].

Fig. 3 DSDM overview



Quinity uses the clear phases from LAD, but also adds business analysis, functional and technical design iteration and implementation from DSDM. The process is depicted Fig. 3.

Fig. 4 Quinity development process



2.1.1 Requirements

In this phase all requirements are gathered from and with clients. The client also prioritises the requirements according to the MoSCoW rules from DSDM. (Must have, should have, Could have and Won't have) [38]. This phase also includes business analysis as specified by DSDM, which gives insight in business processes that have to be automated. The requirements documented states what an information system will deliver.

2.1.2 Functional Design

Functional design as defined by Quinity [9] is a phase during software development in which documents are created that elaborate on how the information system will fulfil its requirements. The functional design phase delivers:

- The functional description
  This describes the information functions the system will support.
- Mock-ups
  A graphical representation of the user interface which gives an idea how the end users will interact with the system.
- Data model
  Entity Relationship Diagram [7] of the conceptual data structure, which later will make up the system's database.

2.1.3 Realisation

The realisation phase consists of creating the technical design document and implementing this. The technical design document as defined by Quinity [31] explains the way the functional design should be realised. All design decisions are made explicit in a technical design. This includes the memory model. Class diagrams, exception handling, core algorithms which require explanation.

The technical design is realised by software developers. Multiple software engineers work on different areas of functionality, which are tested using unit tests. Periodically all parts of the system are integrated. On the integrated product a system and performance test is run.

2.1.4 Acceptance Testing

After the realisation phase the information system is deployed and the client tests if all functionality works like it should according to the functional design. The Acceptance tests provide feedback to the designers and as such the functional design and realisation phase are repeated.

2.1.5 Usage And Maintenance

When the client approves all functionality, the software will go live. The development now enters a maintenance state. Maintenance is done via request for change proposals, which brings development back in the requirements phase.

2.2 Functional Design Patterns

When developing multiple applications for several clients, there is functionality that is desired by more than one client, this can be captured in a Functional Design Pattern.

"Functional Design Patterns describe recurring functionality problems in (domain specific) applications"[37]

Functional Design Patterns capture domain knowledge and define structure on an abstract level. The main motives of Functional Design Patterns are:

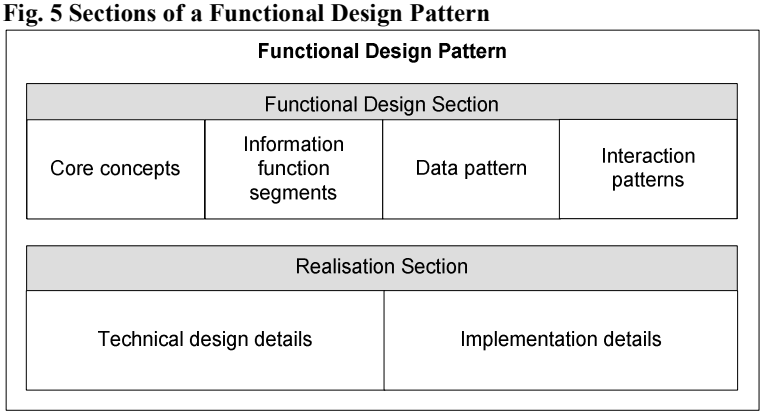
- Transfer knowledge
  Functional Design Patterns contain knowledge how to solve functional problems. The advantage is that new functional designers have a quick way of getting a grasp on the way functionality problems are solved instead of having to rely on more experienced designers to help them.
- Increase predictability
  It is believed that using Functional Design Patterns makes it possible to improve the predictability of the software development process. Predictability increases because the complexity of the functionality and the time it takes to implement are already known due to past experience.
- Improve quality
  Because Functional Design Patterns describe proven solutions for recurring problems, the risk of developing a new untested solution is lower.
- Save time and cost
  Reusing solutions in functional and technical designs is expected to save time and cost.
- Increase maintainability
  Reusing functional and technical designs will ultimately create similar code, which increases maintainability.

The main difference with Analysis Patterns [11] is mentioned by Reyngoud and Van Helden [34]. They state that analysis patterns focus on the static representation of the conceptual data model whereas Functional Design Patterns show how interaction between entities takes place to attain functionality.

In essence Functional Design Patterns combine analysis patterns with interaction [41] and data patterns.

A Functional Design pattern consists of a functional design section and a realisation section. Both sections are explained below. Fig. 5 shows the composition of a single Functional Design Pattern.

Fig. 5 Sections of a Functional Design Pattern



Functional Design Section

- Core concepts
  Core concepts describe the most important constructs and definitions used. The core concepts allow a common vocabulary to be used amongst designers.

- Information function segments
  Information function segments are the building blocks for information functions. Multiple segments can be combined to form one Information Function but the can also be used separately. Information functions are functions in an information system executed by either a human or computer that display and or manipulate data. [17]. Examples of information functions are “order item”, “print quotation” and “edit person information”.

- Interaction patterns
  Interaction patterns show the way users interact with the system [41]. These consist of a mock-up with a description of the interaction.

- Data pattern
  The data pattern describes the entities and their relations, usually an Entity Relationship diagram.

Realisation Section

- Technical design details
  The patterns feature technical details, such as class diagrams that should be used when writing a technical design.

- Implementation details
  Some patterns have small code snippets that show how to solve a particular functional problem with code. These code snippets can be used when implementing the Functional Design Pattern in a project.

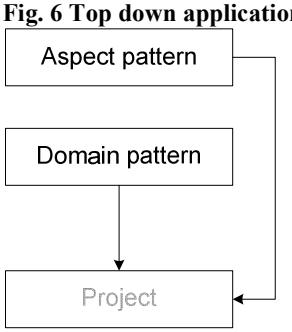
Some Functional Design Patterns also define extensions to the basic pattern, which might only be applicable in certain situations, with certain requirements. Other patterns may define variants when it is not clear yet which is the superior solution.

Functional Design Patterns can be categorised in aspect level and domain level patterns. Design Patterns at the aspect level describe functionality which is the same for all administrative applications independently of their domain, they are domain transcending.

Thus, patterns at the aspect level do not capture domain knowledge. They describe standard implementations for information functions. Examples of aspect level design patterns include browsing entities and Entity Management [17]. Domain level Design Patterns describe abstract functionality for a specific domain or problem area. Examples of domain level design patterns are pension calculations and chain integration in the insurance field [17].

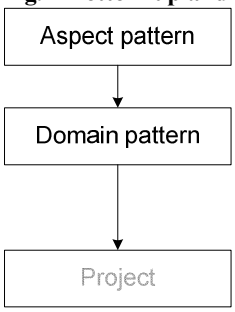
When Functional Design Patterns are used in a functional design they will be altered to the project's context, this is the lowest level. Domain level patterns are used in projects. When no domain level pattern is available, an aspect level pattern is directly applied in a project. The design pattern applied at the project level is not really a pattern, that is why it is greyed out in the figures. The arrows represent *can be used in*.

Fig. 6 Top down application of Functional Design Patterns



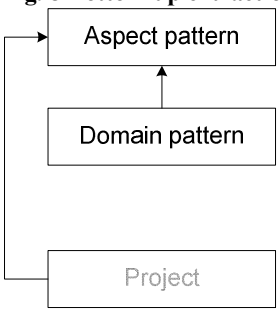
To create domain patterns, the Functional Design Patterns from the aspect level are made specific using knowledge from executed projects.

Fig. 7 Bottom up and top-down development of Domain level pattern



When functional problems occur in multiple projects, they are abstracted to either the domain or the aspect level. It is also possible to further abstract from a domain level pattern to the aspect level when similar functionality is identified across domains is discovered later on.

Fig. 8 Bottom up extraction of Functional Design Patterns



2.3 Link Between Process And Patterns

This section describes the current link between Quinity's software development method and Functional Design Patterns. To engineer a method for the incorporation of Functional Design Patterns preliminary interview session were organised to enquire about the way these patterns are used right now.

These interviews with functional designers at Quinity showed that the overall feeling is that the usage of Functional Design Patterns is not standardised. Apart from small Functional Design Patterns which are already embedded in components, the experience in applying Functional Design Patterns is limited.

Practitioners know the definition and there also appears to be a consensus on the advantages of Functional Design Patterns. Designers are convinced that Functional Design Patterns help to comprehend subject matters more rapid, reduce complexity and improve the overall quality of delivered software.

Following is a list with the most important points extracted from the interviews.

Important Phases

The phases of development in which Functional Design Patterns play an important role are the requirements, functional design and realisation phase. After the technical design phase, the usage of Functional Design Patterns decreases. Instead the effort that goes into creating Functional Design Patterns and updating the repository increases. Feedback from implementing a pattern in a project is used to update the pattern documentation and new patterns are derived. This is implicit; it is assumed people will act on their own. When a Functional Design Pattern at the aspect level exists without technical details Quinity assumes that developers which implemented the pattern will have the discipline to update the pattern.

Cost Estimation

Experienced functional designers identified that it would be helpful if Functional Design Patterns could be used in the acquisition phase to support the creation of more accurate time and cost estimations. Because the patterns are created from existing projects it is already known approximately how much time their implementation takes.

Common Vocabulary

Some designers already use Functional Design Patterns as a structuring mechanism in interviews and design sessions with clients. The core concepts that are explained in each Functional Design Pattern give designers and clients a common vocabulary.

Clients often come up with their own definition of certain words. A Functional Design Pattern helps the functional designer to recognise the real concept. For example, users familiar with a different kind of authorisation will frequently call a set of users with some similarity in their permissions a group, while in Quinity's Functional Design Pattern for authorisation this is called a role. Interestingly not all

functional designers agree that the core concepts from a Functional Design Pattern should be communicated to the client. Quinity also encountered occasions where the client had already made an analysis of their own product workflow, presenting fifteen different states the product could be in. Without the guidance of a Functional Design Pattern they might have thought that the analysis the client came up with was correct, but due to experience and the core concepts described in the Functional Design Pattern it could be discovered that there were in fact only three states which carried along five process dimensions [30].

Some functional designers believe it may cause confusion because the concepts can be quite difficult to understand and in the end the customer is interested solely in a working product, how problems are solved is not there concern. Therefore they feel it would be wise for the designer to just keep the concepts in his head, using the Functional Design Pattern as a tool.

Guide Technical Design

Furthermore an important role of Functional Design Patterns is defining the mapping from functional design to technical implementation. When a Functional Design Pattern is distilled from a successfully executed development project, the creator of the Functional Design Pattern is already intensely submerged in the subject matter that it would be wasteful to neglect writing down hints for the actual implementation of the described functionality. Therefore, current Functional Design Patterns also contain technical documentation that is used by technical designers in the creation of a technical design in the implementation phase.

Unclear Usage Of Parts

Which parts of Functional Design Patterns should be transferred to a functional design document and how this should be done is acknowledged by practitioners to be ambiguous. Mostly the diagram styles used in the Functional Design Pattern are used. The text is not directly usable in the functional design document. Thus it is not a matter of copy and pasting Functional Design Patterns from their document to the functional design.

Pattern Recognition

Recognizing patterns is not a mechanical task. The knowledge of patterns is currently located in the designers head. When they encounter requirements which seem to be very generic they check if members of other projects may have defined usable patterns. The lack of a knowledgebase or pattern repository which can be consulted to search for patterns is recognised by the designers. The current tactic is to come up with a keyword for the problem and search for documents describing these.

Domain level patterns are occasionally fully shown to clients, but no guidelines exist which parts should explicitly be shown or when a designer should refrain from showing documentation.

Deciding on the pattern to use is based on the experience of the designer. The patterns currently do not contain description of situations when it not advised to apply them.

Pattern Documentation

The way patterns are documented was said to slightly vary between patterns. Not all patterns use the same structure, which sometimes makes them difficult to read. The documentation is very extensive which makes it necessary to read the entire pattern to get a good grasp of the concept.

Pattern Combining

When multiple patterns are found which could possibly be of use in the functional design, designers admit that a structured way of combining the patterns is not available. No real order exists in applying the patterns. One designer noted that it is probably best to start with the pattern which has the most significant impact on the design.

Pattern Repository Growth

The functional designers identify that in the future a lot more patterns will be developed and when that growth is achieved it would be beneficial if a structured approach was available to make optimal reuse of developed patterns. The future as envisioned by Quinity's management is the total incorporation of Functional Design Patterns in software development. This will hopefully allow them to maintain their competitive edge.

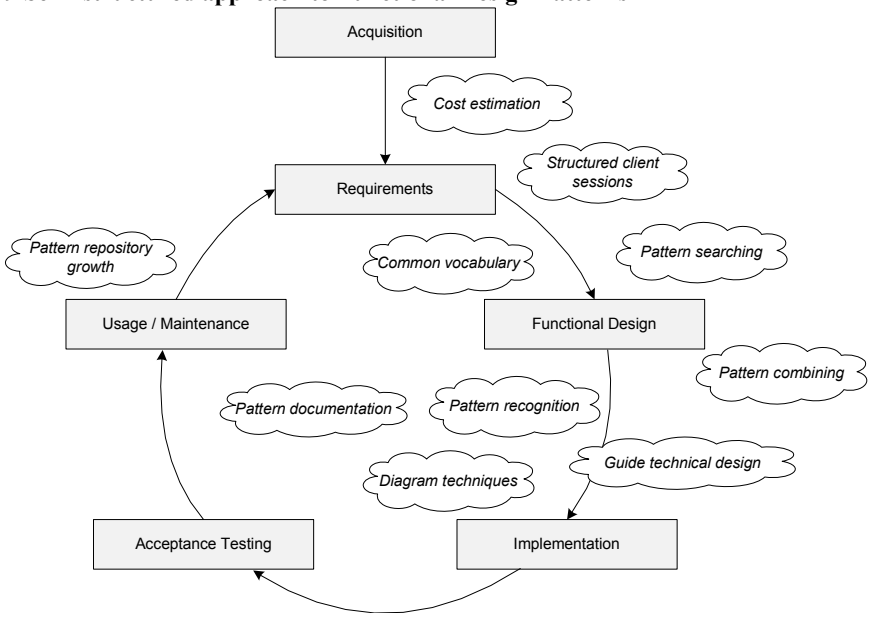
2.4 Conclusion

In this section we showed the software development process at Quinity, Functional Design Patterns and the current link between the development process and patterns.

Summarizing, the state of the practice shows that functional designers see a bright future for Functional Design Patterns but a standardised way of putting them to use is missing. Not all functional designers agree on the precise incorporation of Functional Design Patterns in the development process. Fig. 9 represents the current state of the practice, a semi structured approach with uncertainty and ambiguity. This is the way a new functional designer would feel when starting at Quinity.

The method that will be described in chapter 4 transforms the semi structured approach to Functional Design Patterns into a structured one by introducing the important additions found in literature in the state of the practice.

Fig. 9 Semi structured approach to Functional Design Patterns



3. Existing Pattern Integration Methods

This chapter presents an analysis of the existing methods to integrate the use of patterns into software development. The purpose of the analysis is to find ideas from integrating design patterns in development which are also applicable to Functional Design Patterns.

The methods were chosen because they attempt to integrate patterns in a systematic way. First off the Pattern Oriented Analysis And Design method from Yacoub and Ammar [2] is explained. Software reuse with analysis patterns is covered in 3.2, after which a small extension to POAD is shown [18] that makes use of analysis patterns as well. An alternative, building software with patterns [4] is presented in paragraph 3.4.

3.1 Pattern Oriented Analysis And Design

Yacoub and Ammar [2] state that designing applications by deploying technical design patterns is not a straightforward task. Although several pattern composition techniques have been proposed, they do not have a systematic process, which is what POAD proposes. Fig. 10 presents the POAD process overview.

POAD uses so called *constructional design patterns*. They represent design components in the application design. Constructional design patterns can be glued together because they are object oriented patterns, have interfaces for composition and their solution has a class model of collaborating classes. This thus excludes the composition of patterns which do not have a class diagram.

Yacoub and Ammar view technical design patterns as building blocks that solve a particular problem. The requirements of an application are transformed into conceptual components. Once the problems of the applications have been written down and assigned to conceptual components the patterns are applied like building blocks. A system in POAD consists entirely of technical design patterns.

POAD presents pattern diagrams at different levels, allowing tracing back and forth between diagrams at a higher or lower level. The visual representation is viewed as essential to understand the role patterns play in development. Examples of these pattern diagrams can be found in appendix 7.2.

Yacoub states that although they describe their process in a linear fashion, incremental and iterative development is encouraged. Iteration is made possible because POAD documents all steps.

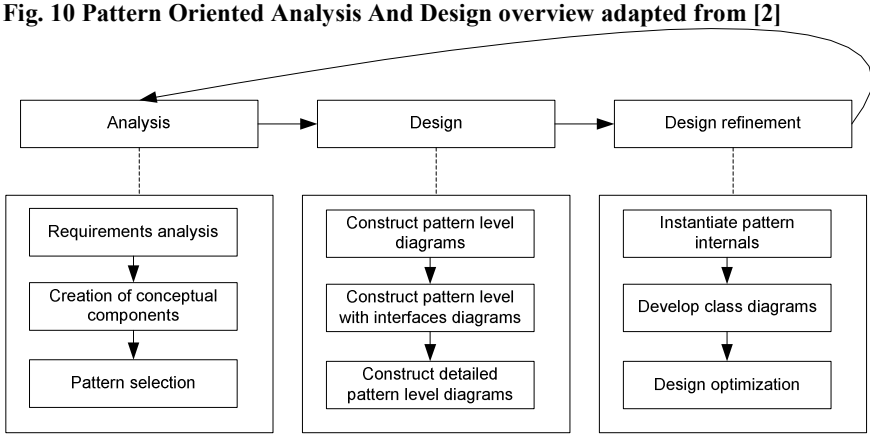
After the design process POAD follows a traditional object oriented detailed design process and implementation.

Two interesting additions are

- the identification of an acquaintance process
  during which a developer acquaints himself with available solutions to use, this is something we can relate to Functional Design Patterns because the existence and a global idea of what the pattern does should be know by the designer.

- merging similar responsibilities
  The initial class diagram is transformed into an optimised diagram. Because all patterns are selected and instantiated without looking into details, it is probable that there is some overlap. Designers might implement Observer patterns multiple times. The abstract Observer class can thus be shared. Next to overlap there will be classes which are very trivial responsibilities. These can be merged. Using either pattern documentation, application specific documentation or by studying pattern relationships the designer finds classes that can be merged.

Fig. 10 Pattern Oriented Analysis And Design overview adapted from [2]



3.2 Software Reuse With Analysis Patterns

Geyer-Schultz and Hahsler [13] suggest using patterns in the analysis phase of software development because it has the potential to reduce development time significantly because reuse is introduced early on in the process and the interface between the analysis and design phase is improved.

Fig. 11 Analysis patterns in the software development process

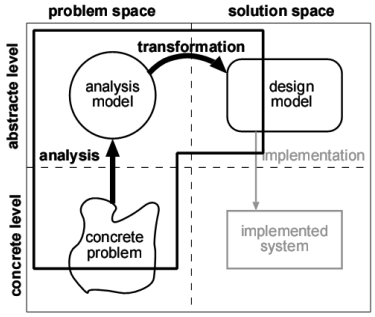


Fig. 11 shows the main tasks where analysis patterns contribute to software development.

- Analysis patterns speed up modelling of abstract analysis models by providing base models with examples and their limitations
- Analysis patterns facilitate the transformation of the analysis model to the design model.

Geyer-Schulz stresses a consistent format for describing analysis patterns, in contrast to Fowler's free form [11].

They also mention that a key challenge is establishing a common vocabulary between authors and users of patterns. A common vocabulary is important to achieve efficient communication.

Geyer-Schulz start off with a desire to create a simple application. This application is turned into an analysis pattern. At the beginning of each new project the analysis pattern is used as the base. Gradually the analysis pattern is extended because the bigger projects need more functionality.

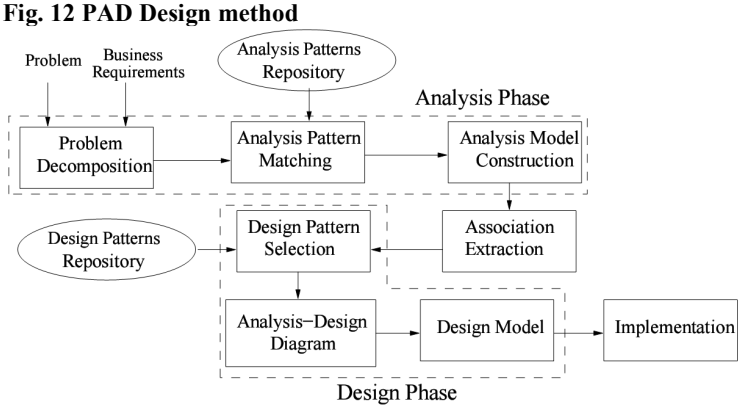
The abstraction of an analysis pattern from an executed project so it can be reused in subsequent projects after which the original pattern is updated is very similar to Functional Design Patterns. The challenge of a common vocabulary was also mentioned in the interviews with Quinity designers.

3.3 Pattern Driven Analysis And Design

Hamza and Chen [18] base their design method on POAD but state that it can be improved upon by using analysis patterns, just like Geyer-Schultz. PAD uses analysis patterns to develop an analysis model for the problem at hand which is then

transformed into a design model using design patterns. This is the basic idea which was explained in the previous section.

Fig. 12 PAD Design method



The problem the software should solve is decomposed into sub problem using existing analysis techniques. A solution for every sub problem is sought for in an Analysis patterns repository. These analysis patterns are integrated to an analysis model. Details on how this is achieved are not given. The relations between classes in the analysis model are annotated and based on these relations design patterns groups are selected from a design pattern repository. Patterns within groups are studied to select the best suitable ones. From this a so called Analysis-Design diagram is developed. The design model can be obtained by adding the detailed design of the design patterns to the model.

Hamza does not elaborate on the proposed method in detail. The essence is that the transformation from problem decomposition to design model is aided by using core concepts from analysis patterns. This is a useful idea in the light of Functional Design Patterns.

3.4 Building Software With Patterns

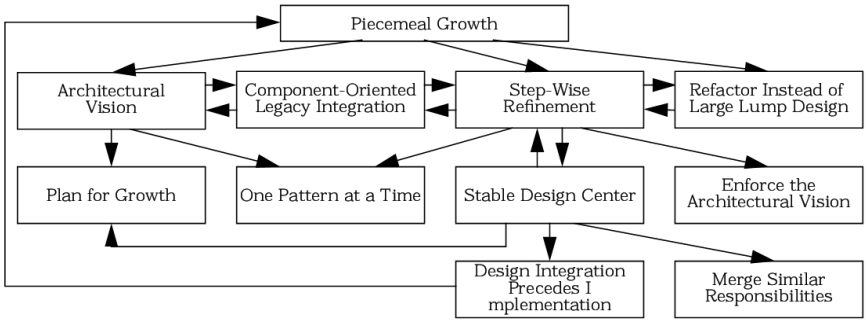
Buschmann [4] states that next to pattern specific implementation details we need general guidelines to construct pattern based software. What is lacking in the current software architecture field is the answer to the question how we can combine technical design patterns into partial or larger structures.

The application of patterns he says, is not a mechanical task. Experience is needed to compose them to large structures in a meaningful way. Guidelines to help us do this cannot be simple two-line answers, they need structure. Every guideline describes its context, the problem, a solution and a clear example. This structure makes the guidelines patterns themselves.

Buschmann presents eleven of these patterns of pattern based software development. Each pattern makes most sense when used in the context of the patterns

that precedes it or the one that it completes and are overall very dependent on each other. Therefore the eleven patterns together form a pattern language.

Fig. 13 Pattern language of pattern based software development



The most important pattern is *Piecemeal growth*. This outlines the overall process for the construction of software using patterns. The software architecture grows by using top-down refinement and bottom-up refactoring until it is complete and consistent.

Architectural vision defines the systems baseline architecture, this step uses analysis- and architectural patterns that help with the specification.

Step-wise refinement describes how to resolve design problems by detailing and extending a given software architecture. This is the top-down process from piecemeal growth.

Refactor instead of large lump design elaborates on how to continue when a design solution does not fit with part of an existing design. This is the bottom-up process of piecemeal growth.

Stable design center talks about specifying an extensible design with the help of patterns.

Plan for growth describes how we can prepare a software architecture for its own evolution.

Component-oriented legacy integration shows how to take advantage of design patterns when 3rd party components have to be integrated in the software architecture.

Enforce architectural vision supports the application of global design principles in every part of the software architecture.

One pattern at a time helps with combining several patterns which together should define the design of a specific part of the system

Design integration precedes implementation introduces a way to implement a pattern in a given architecture.

Merge similar responsibilities combines multiple patterns which have similar responsibilities.

As for the actual selection of patterns he refers back to his own work, Buschmann [5] which presents the basic steps for pattern selection, these are very useful and applicable to Functional Design Patterns as well.

Buschmann [5] states that patterns can be stabilised further by extending the list of known uses whenever it is applied successfully. The more known uses, the higher the chance a functional designer will identify it in the analysis phase of a subsequent project.

The overall pattern based development process presented by Buschmann is an evolutionary process and he states that it will not work in a waterfall-like process model. It is not an entirely new way of development; instead it complements existing approaches with respect to the use of patterns. The only really new addition to existing models is the definition of a baseline architecture at the start of development, even before the specification of the detailed domain model. To integrate this into software development models we can simply add the creation of a baseline architecture right after the requirements analysis phase.

Another condition introduced by Buschmann is the need for a software architect. The *Architectural vision* cannot be defined by all developers; rather it is to be thought up by one individual with vision. This has to be an experienced developer with overview of the entire system as well as insight into specific needs. The defined baseline architecture and vision need to be communicated to the developers that develop individual parts and in the end integrate them to a consistent whole.

3.5 Discussion

The presented integration methods all provide valuable insights to reuse patterns in software development. Although different in their setup there are a few common elements and features of the methods that we will keep in mind for the design of the method for incorporating Functional Design Patterns in the software development process.

Focus On Technical Aspects

The presented methods heavily focus on technical aspects of design. They go directly to class diagrams from requirements without creating a functional design. Class diagrams seem to be the desired end result, this only partially holds for development using Functional Design Patterns.

Acquaintance

Yacoub identified a process in which the designer gains knowledge about existing patterns and the presented solutions. We feel this is important for Functional Design Patterns as well.

Visual Representation

The pattern integration is illustrated using UML in most cases. We believe a visual representation like POAD tries to push is not feasible for large administrative systems, a larger structure cannot be shown without cluttering. It is known that humans can only keep 7±2 things in their immediate memory [25]. Therefore a

diagram of the entire system does not seem as useful. Instead, difficult parts of the system may be elaborated on in more detail.

Another reason why the visual representation of pattern composition is not applicable to Functional Design Patterns is because these patterns are not black boxes. It is not possible to accurately define their input and output. We actually tried this during our research but found that composing Functional Design Patterns is different from applying multiple patterns. This is demonstrated in Appendix 7.3.

No Influence After Requirements And Design Phase

Most methods stop after the design phase this means that no hints concerning the rest of the phases can be extracted.

Merge Pattern Responsibilities After Applying

Both POAD and "Building software with pattern" show that patterns can be merged to reduce overall complexity of class diagrams.

Incremental Design

All approaches reviewed show that reuse is an incremental and iterative process, nothing is built perfectly in the first attempt and during development unforeseen requirements will certainly arise.

Common Vocabulary

The importance of a common vocabulary is stressed solely by Geyer-Schultz [13]. We think this is because technical design is closer to code, where analysis patterns are closer to natural language.

Minimal Impact On Current Development Methods

Currently existing development methods can largely stay intact in all cases. Pattern integration although difficult, in essence just introduces a few more steps in the software development process. This is something we also want to achieve. Radically changing the current development process is not desired and not feasible.

Conceptual Decomposition

All methods decompose problems into smaller concepts. This helps when tackling difficult issues and shows where reuse is possible. Hamza and Geyer-Schultz use analysis patterns to transform the problems into a design model. This idea is applicable to Functional Design Patterns as it will probably be easier to recognise patterns when the problems are clearly divided.

Systematic But Not Mechanical

Buschmann describes problems and solutions but does not provide exact procedures to follow. We believe this way of describing is suitable for Functional Design Patterns as well. A guideline is something completely different from a rule. We do not want to restrict freedom; we want to provide useful information without imposing an extra burden on designers and developers.

A Complementary Process

POAD build software systems from patterns only, they seem to forget that occasionally patterns are not applicable at all. POAD assumes all the conceptual problems we decompose the system in can be solved by patterns. We do not think a system can be built entirely from Functional Design Patterns, especially when the pattern repository, which at the time of writing is not that extensive, does not cover all domains. Therefore the method should be complementary to the existing process.

Pattern Selection

Buschmann [4] refers to himself [5] for pattern selection techniques. The general guideline is to pick the pattern that best matches the problem description with the least liabilities. A mathematical approach to decide between patterns has been proposed by McPhail and Deugo [24]. This is a method which uses weighted criteria to evaluate which pattern deserves preference over another pattern. In a real life situation however finding criteria to evaluate on is as hard as selecting the right pattern.

Intrinsic Analysis Patterns

Analysis patterns are already contained in Functional Design Patterns because the core concepts together with the data model are in fact analysis patterns.

3.6 Criteria

Resulting from the discussion and state of the practice we can state the desired properties and criteria a method to incorporate Functional Design Patterns in software development should fulfil.

C1. Clearly phased
Because the method will be integrated into the existing development method, which is divided in clear phases, the new method should also have this property. It should be clear which contribution Functional Design Patterns have in which phase.

C2. Systematic but not mechanical
Although it should be clear which parts of a Functional Design Pattern can be used; developers should retain a certain amount of design freedom, they are not robots.

C3. Clear division of tasks
To avoid confusion about responsibilities the new method should clearly define who is responsible for which activities.

C4. Facilitate communication
The new method should make the communication between designers as well

as designers and clients easier. In this thesis we will solely focus on the common vocabulary.

C5. Systematic pattern recognition
The method should support systematic pattern recognition. To apply patterns we first have to be able to recognise them. A systematic way of doing this is necessary to allow even novice functional designer to recognise them, albeit somewhat slower.

C6. Pattern combining
A structured approach to combining multiple patterns should be included. Currently such an approach is identified as lacking by functional designers.

C7. Complementary method
The existing development method should largely stay intact. The new method should be complementary to the existing method, when no patterns can be applied it should be possible to follow the normal development method.

4. Systematic Reuse Of Functional Design Patterns

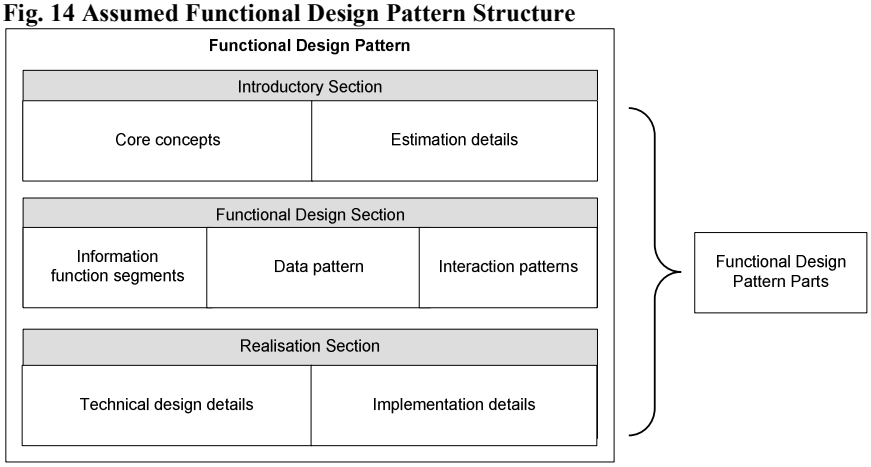
This chapter defines a systematic approach to reuse Functional Design Patterns in software development. The method we present is developed with the criteria from chapter 3 in mind. The first section explains the prerequisites that we assume for the method to be applied successfully. Next we give a short overview and a graphical representation of the method. The fourth section describes the process of developing information systems with the incorporation of Functional Design Patterns in detail. The process is demonstrated using a fictional case depicting the development of an online banking system in the last section.

4.1 Prerequisites

For the method to be applied successfully the following assumptions are made.

- A pattern library with Functional Design Patterns is available.
  This can be anything from a book with patterns, to a folder with documents. In the described process any of the above is called a pattern library to keep things generic.
- It is known how patterns are created and supported. The focus of the method that is defined in this chapter is on the reuse of patterns. The creating and supporting activities are thought of as the basic building blocks for the described process to work.
- A standardised pattern structure. The assumed setup of the structure of a Functional Design Pattern is the following.

Fig. 14 Assumed Functional Design Pattern Structure



4.2 Outline

The overall process is based on the current Quinity development method, with the addition of the acquisition phase. Each phase is introduced here briefly. Functional Design Pattern parts (from Fig. 14) are written in an oblique font.

Acquisition

In the acquisition phase we try to detect Functional Design Patterns as early as possible so we can adjust the quotation based on *Estimation details*. The recognition of patterns in this stage is solely for those with plenty experience in the use of Functional Design Patterns. When the acquisition phase has been completed the next step is the requirements phase.

Requirements

This phase is divided in the actual gathering of requirements in client sessions where we make use of the *Core concepts* of Functional Design Patterns to streamline communication with the customer. From the requirements we distil the information functions the system will support. We group these into conceptual problems which together make up the conceptual architecture. Each conceptual problem is matched against the pattern repository to find applicable patterns.

Functional Design

In the Functional Design phase we deliver a document which contains a description of the information functions, the way end-users will interact with these and the conceptual data model. We copy the applicable *Information function segments* parts of Functional Design Patterns to the functional design document and adjust them where needed. The data model is obtained by applying the *Data patterns* from Functional Design Patterns. *Interaction patterns* are applied on the mock-up to show the way users will interact with the system.

Realisation

The *Technical details* given in Functional Design Patterns are integrated in the technical design. When the technical design is completed the system is implemented, making use of the *Implementation details* in the patterns.

Usage And Maintenance

When the entire information system is accepted by the customer, the technical designers report their experience to the functional designers which in turn update the Functional Design Patterns. New patterns may be found during development as well, these are documented at this time. This brings us in the stable state in which request for change take us back to a new requirements phase.

4.3 Visualising The Method

To visualise the process we use the Artefact Dependency Graph notation as presented by Tekinerdoğan [39]. This model shows the dependencies of the artefacts developed during the execution of the software development process.

The entire process is graphically represented in Fig. 15. In the graph every node represents an artefact and every arrow a dependency. Node A→ Node B means that B needs A, or can only be produced after B. Every dependency has an accompanying rule associated with it. Every rule has one or more persons responsible for the rule. This is denoted as Person : R_x.

The gray nodes represent the parts from Functional Design Patterns as shown in Fig. 14.

In addition to the graphical representation of the method we also included a table for your convenience. The table contains the same information represented in a different format. The tabular process outline is given in Table 1.

Fig. 15 Activities, responsible roles, and involved FDP parts in the method

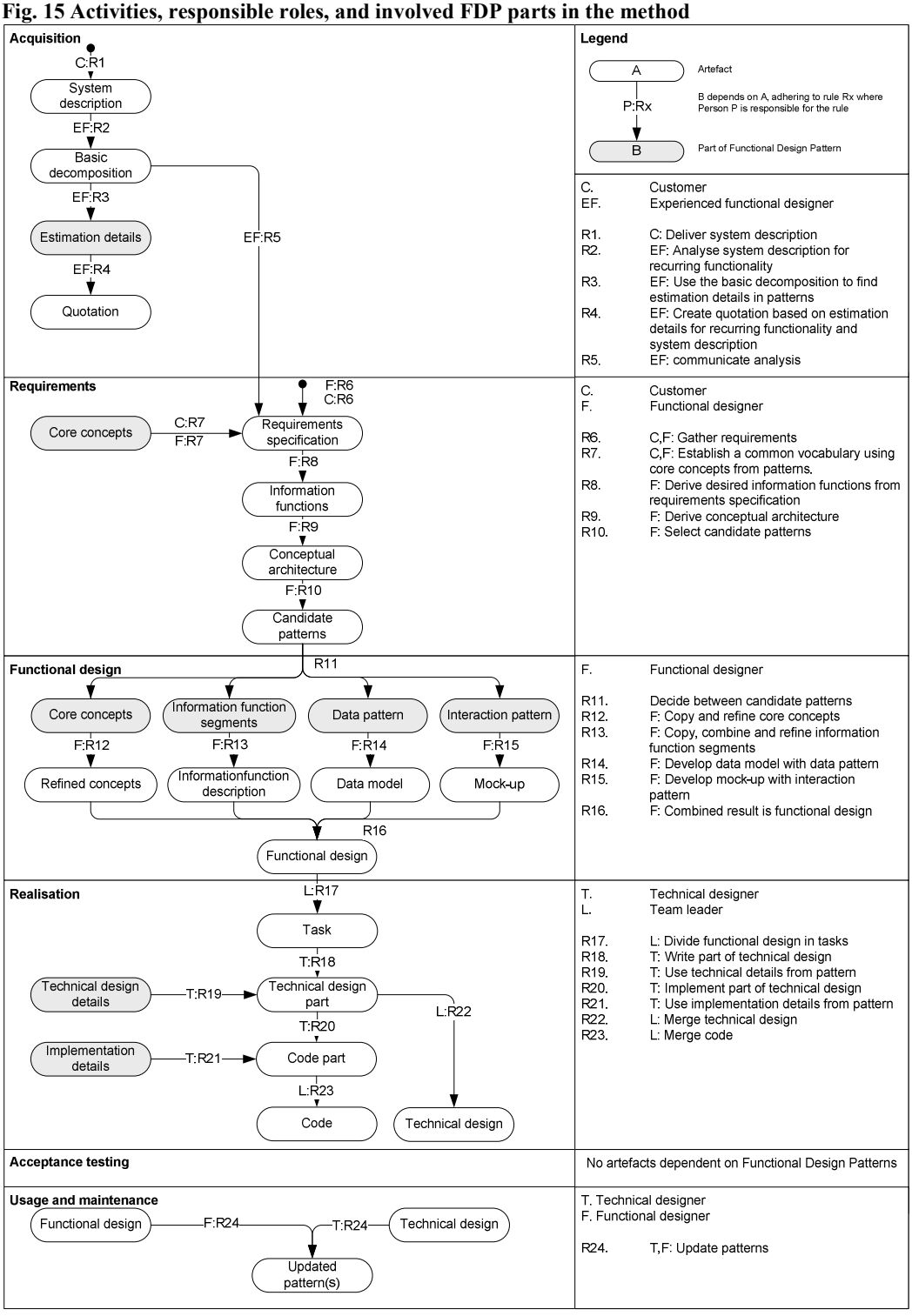


Table 1 Process Outline

Involved Roles	Activities	FDP artefacts involved	Rules
Acquisition			
• Customer • Experienced functional designer	• Deliver system description • Analyse the system description for recurring functionality • Check cost and planning experience in Functional Design Patterns • Create cost and planning estimates for Quotation • Communicate basic analysis to functional designers	• Estimation details	R1...R5
Requirements			
• Customer • Functional designer	• Gather Requirements a. Acquaint with patterns b. Establish A Common Vocabulary • Create A Conceptual Architecture a. Derive information functions b. Decompose In Conceptual problems • Find solutions for conceptual problems a. Recognise patterns b. Select patterns	• Core concepts	R6...R10
Functional design			
• Functional Designer	• Decide Between Candidates And Variants • Describe the information functions a. Copy and refine core concepts (Optionally rename to clients preferred terms) b. Use Prescribed diagram style c. Copy, Combine and refine information function segments. • Develop the data model a. Create basic model b. Sort patterns in order of importance c. Apply one data pattern at a time d. Merge similar responsibilities e. Return to information functions	• Core concepts • Diagram style • Information function segments • Data pattern • Interaction pattern	R11...R16

	f. Extend with pattern independent entities • Develop the mock-up a. List information functions to be supported visually b. Check Functional Design Patterns for interaction patterns c. Apply interaction patterns during design • Deal with open issues in patterns. • Document used patterns for technical designer		
Technical design			
• Team leader • Technical designer/ Software engineer	• Divide functional design • Write the technical design for a specific part a. Check technical design details in patterns b. Assess advantages of copying versus referencing c. Refine technical design details in the technical design • Develop software using implementation details • Merge the technical design parts	• Technical details • Implementation details	R17...R23
Acceptance testing			
	Only indirect influence		-
Usage and maintenance			
• Technical designer • Functional designer	• Evaluate on the use of patterns in this project • Update the pattern library	• Modified sections	R24

4.4 Acquisition

The starting point of any project is its acquisition. Recognition of Functional Design Patterns is reserved to experienced functional designers who already have the knowledge and know-how to recognise the patterns from only a basic description of the systems purpose.

Fig. 16 Dependency graph for acquisition phase

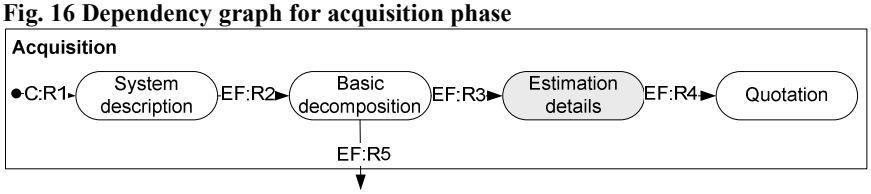


Table 2 Rules and activities in acquisition phase

Rule	Person	Involved activities
R1	Customer	Deliver system description
R2	Experienced functional designer	Analyse the system description for recurring functionality
R3	Experienced functional designer	Check cost and planning experience in Functional Design Patterns
R4	Experienced functional designer	Create cost and planning estimates for quotation
R5	Experienced functional designer	Communicate basic analysis to functional designers

For every Functional Design Patterns that is recognised we can view the most recent cost and planning estimates and accompanying real figures and results. Comparing the estimates with the results to adjust the new estimate allows for more accurate cost and planning estimate. This will position the organisation better in the eyes of the customer.

It is important that this basic analysis is not lost between this phase and the next when we continue with the next step because obviously we want to optimise efficiency.

The impact of recognizing relatively small Functional Design Patterns like the sorting and displaying of entities in a SearchList [33] is lower than the impact of detecting a Workflow pattern in this phase. The difference is the complexity of the functionality described. Workflow can be a complex matter, but knowing that it has been done before and a pattern exists reduces the risk of creating incorrect estimates. Whereas sorting of entities is not really complex, it is just nice to have a default way of doing it.

Requirements

In the requirements phase the use of Functional Design Patterns is intertwined with gathering the requirements, analysing these and determining the Functional Design Patterns that will be used in the functional design. This section explains the steps to take.

Fig. 17 Dependency graph for requirements phase

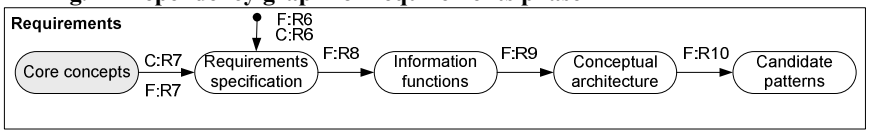


Table 3 Rules and activities in requirements phase

Rule	Person	Involved activities
R6 R7	Customer; Functional designer	Gather requirements • Acquaint with patterns • Establish a common vocabulary
R8	Functional designer	Derive desired information functions from requirements specification
R9	Functional designer	Decompose in conceptual problems to create a conceptual architecture
R10	Functional designer	Find solutions for conceptual architecture • Recognise patterns • Select candidate patters

We will now discuss these activities in detail.

4.4.1 Gather Requirements

The first step in the analysis phase is obtaining the requirements from the client.

Acquaint With Patterns

Before diving in to the requirements gathering sessions at a client a functional designer should at least get himself acquainted with the patterns that:

- were discovered in the acquisition phase
- available patterns in the clients domain

Due to the fact that a pattern library is always under development it would be a good ideas to revisit the acquaintance process for every new project, just to be sure the designers are equipped with the latest information. Although a quick grasp of the pattern library is useful, it is not desirable to study each and every pattern in the entire library [2].
Keeping the focus on the sections of the Functional Design Pattern that describe the essential problems they solve and the core concepts that are explained is the best tactic.

4.4.2 Establish A Common Vocabulary

At this point the functional designer has basic knowledge of the patterns and has studied the core concepts. It is exactly these concepts that help to structure the requirements and design sessions at a client. The usage of the core concept section of Functional Design Patterns gives the functional designer the ability to detect concepts in the current project and at the same time prevent us from drawing the wrong conclusions. The result is a better analysis that will result in a better design as well.

There are two options; both with their own trade-offs.

- **Option A** Explain Functional Design Patterns and the used core concepts
  In this case the designer explains the client about the existence of Functional Design Patterns and the core concepts discovered in previous projects.
  At the time of writing it is not clear when to use a term from the client's vocabulary and when to use the developers preferred term. In practice a common vocabulary evolves from combining client terms and developer terms.

- **Option B** Refrain to tell clients about Functional Design Patterns
  In this case the designer adapts to using the preferred terms of the client and later on in the functional design document create notes and references to the core concepts of a Functional Design Pattern that were actually meant. This could possibly result in miscommunication later on in the project.

Option A is the preferred option, because it lowers the risk of miscommunication. Although there might be times that a client is not willing to cooperate.

4.4.3 Create A Conceptual Architecture

To discover Functional Design Patterns to be applied we first need to define a conceptual architecture of functional problems the information systems is made up of.

The functional designer should create the conceptual architecture by:

- Derive information functions from requirements
- Decompose the system in conceptual problems

Derive Information Functions

Deriving information functions is done by analysing every requirement for verbs and nouns which indicate actions to be taken by end-users of the system. Obviously deriving information functions is not totally straightforward, it takes experience to find all information functions.

Decompose In Conceptual Problems

Using the information functions as our input we look for functionality problems that need to be solved. The conceptual functionality problems are identified by analysing the information functions for similarity and then group them. The decomposition creates manageable pieces of functionality whose core problems corresponds to pattern problem descriptions therefore allowing for better recognition of patterns.

4.4.4 Find Solutions For Conceptual Architecture

At this phase of development we have the conceptual components and want to select patterns that will solve our functional problems.

Recognise Patterns

Not all conceptual components may correspond to a Functional Design Pattern. Some components might capture functionality which is encountered very rarely, thus no pattern is available. Other components might easily be recognizable as suitable for Functional Design Patterns. Recognizing that a pattern might be applicable is made easier by translating our requirements into the conceptual architecture. Luckily we can also systematically query the pattern repository for each conceptual component to find Functional Design Patterns, although slower it is a sure and safe path to succeed.

Select Candidate Patterns

The steps to select a candidate pattern are the following:

- Specify the problem
  We have already done these steps in the previous section where the main problems were captured in the baseline architecture.
- Select the pattern level and domain
  Functional Design Patterns are defined at multiple levels. Determine if the conceptual component is an aspect or domain level pattern. Ask the questions:
  a. What do I want to do? This is the aspect level
  b. Where do I want to do this? The answer is the domain
- Select the sub domain
  Selecting the problem's domain narrows the scope and also ensures that the patterns are made more specific for the current problem. When selecting the domain "insurance" for example, there may also be sub domain patterns defined like "car insurance" and "life insurance". When hesitant about including a pattern in the candidate set, just add it. It can always be discarded later. Compare problem descriptions
  With the set of candidate patterns, the designer should compare the core concepts of the Functional Design Pattern with the problem description to determine if the pattern matches the problem at hand. This process can be sped up if a special tool is available that supports searching and matching problems descriptions with problems.
- Expand search domain
  When there are no patterns in the selected domain we have to expand our search area. Some functional problems are not domain specific and some domains do not have any patterns (yet). These patterns should be searched either at the aspect level or at a nearby domain.

4.5 Functional Design

The purpose of the functional design phase is to develop a functional design of the information system. In this step in the design we not only regard the candidate patterns as black boxes that solve a functional problem but also open it to see the pattern internals.

The Functional Design section of a Functional Design Pattern, Fig. 5 (p. 14) is divided in multiple sections itself. These sections will also be reflected in the functional design. In short the actions to take are shown in Table 4.

Table 4 How elements of a Functional Design Patterns should be used

Element	Action
Core concepts	Copy & refine
Diagram style	Use
Data model	Copy & refine
Information function segments	Copy, Combine & refine
Interaction diagrams	Apply in design, no direct copy

Fig. 18 Dependency graph for functional design phase

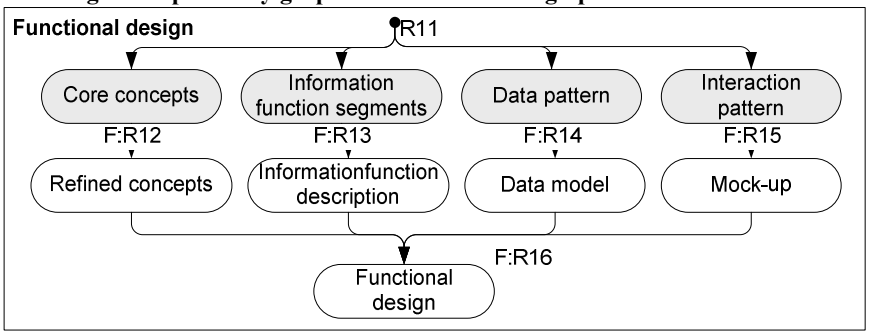


Table 5 Rules and activities in requirements phase

Rule	Person	Involved activities
R11	Functional designer	Decide between candidates and variants
R12	Functional designer	• Copy core concepts, refine when needed • Optionally rename to clients preferred terms • Use prescribed diagram style
R13	Functional designer	Copy, combine and refine information function segments
R14	Functional designer	Develop the data model • Create basic model • Sort patterns in order of importance • Apply one data pattern at a time • Merge similar responsibilities

		• Return to information functions • Extend with pattern independent entities
R15	Functional designer	Develop the mock-up • List information functions to be supported visually • Check Functional Design Patterns for interaction patterns • Apply interaction patterns during design
R16	-	Combined result is functional design
-	-	• Deal with open issues in patterns. • Document the used patterns for the technical designer

We will now discuss the activities in detail.

4.5.1 Decide Between Candidates And Variants

If multiple patterns are recognised to be applicable to a single functional problem we have to evaluate each candidate pattern. The general guideline is to pick the pattern that best solves the problem. By comparing Pick a pattern that has the most benefits and least liabilities. Obviously if a pattern does not define its liabilities then it is up to the analytical skills of the functional designer.

Functional Design Patterns may define variants and extensions to the basic pattern. In particular situations some solution may be more applicable than in others. The variants are described in the patterns and the functional designer should choose the one that is most applicable in the current project.

4.5.2 Copy And Refine Core Concepts

A functional design needs to be a complete document that should be readable without other documents; this is not possible without a clear description of the core concepts. Therefore the definitions of the core concepts should be copied to the functional design document.

If we have chosen not to enlighten the client with the core concepts, rename the core concepts to the clients preferred term.

Not every pattern has core concepts which are “complete”. Some patterns have core concepts that cannot be applied without defining them further. An example of this is the workflow pattern at the aspect level [30] that defines the subject of workflow. The project specific subject of workflow needs to be defined as a new core concept. Furthermore the new concept might also need to be defined in the data model.

4.5.3 Use Prescribed Diagram Style

Functional Design Patterns may define a diagram style which shows the relation between certain core concepts. For example, the Workflow pattern describes how we draw state transitions. We recommend to use this style in the functional design as well because it will create uniform functional designs and reduces the workload of trying

to come up with a way of representing the model with while a sufficient way has already been developed.

4.5.4 Copy, Combine And Refine Information Function Segments

We already have the ingredients to describe our information functions in detail:

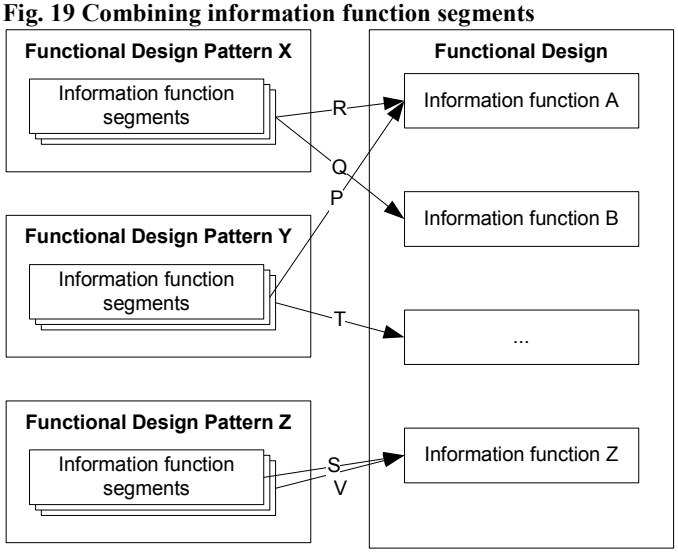
- A list of information functions
- Applicable patterns

In the functional design these need to be combined. Because the information functions needed in the information system may consist of several information function segments we recommend the following approach.

- Copy the list of information functions to functional design document
- Read the selected patterns to define the mapping them and the information functions
- Copy, combine and refine information function segments

Every applicable information function provided in the pattern needs to be copied. Because an information function is built out of information segments, we combine information function segments from multiple patterns into a single information function. This is shown schematically in Fig. 19.

Fig. 19 Combining information function segments



4.5.5 Develop Data Model

An important part of the functional design is the data model. This section explains step by step how Functional Design Patterns are integrated in the data model.

Create Basic Model

Every information system has a few entities which are the most important. Most information functions will perform operations on the data of these entities. These entities make up our basic model. Entities from conceptual problems for which we did not select a pattern should also be added to the basic model. It is wise to think of future requirements when developing the data model. Prepare the data model for extensibility by introducing subtypes for example. A detailed explanation on how to create the basic data model is not part of this thesis.

Sort Patterns In Order Of Importance

Patterns are applied in order of importance. Starting with the most important pattern we incrementally refine and complete the design by replacing all conceptual problems in our conceptual architecture one by one with the data patterns from the Functional Design Patterns. The "most important" pattern is obviously an ambiguous statement, open to interpretation. Guidelines for this sorting process are out of the scope of this thesis and remain future work.

Apply One Data Pattern At A Time

We incrementally apply the Functional Design Patterns that were selected in the requirements phase. By applying the patterns stepwise to the data model we ensure that we only deal with one problem at a time and that the design adheres to the most important structures the best.

Merge Similar Responsibilities

When multiple patterns are applied, overlap might occur in their objects. The authorisation pattern and the workflow pattern both define "User". These responsibilities need to be merged so only one User object remains.

Return To Information Functions

The pattern internals might show concepts we did not address yet. For example, in the authorisation pattern the users have access to information functions but the authorisation pattern also defines "restrictions". Restrictions define that although a role in principle has access to an information function, there are certain conditions under which this access does not hold. The same principle applies to the Workflow pattern which exposes "dimensions" and "process dimensions".

In our functional design we need to address all these remaining issues. So we iterate to be sure we covered all issues.

Extend With Pattern Independent Entities

In the previous section we found solutions for our conceptual architecture, unfortunately there were also problems that could not be solved via Functional Design Patterns. We should not forget to implement these pattern independent entities.

4.5.6 Mock-up Development

Functional Design Patterns should be applied during the development of a mock-up. Functional Design Patterns define interaction patterns that should be incorporated into the design because they give a good example of the way end-users will interact with the system.

The functional designer should:

- List information functions to be supported visually
- Check Functional Design Patterns for interaction patterns
- Apply interaction patterns during design

List Information Functions To Be Visually Supported

Before applying the interaction patterns from Functional Design Patterns we decide which information functions should have their information functions supported visually by basic screen examples.

Check Functional Design Patterns For Interaction Patterns

For every information function selected the Functional Design Patterns should be checked for interaction patterns.

Apply Interaction Pattern During Design

For every information function for which a mock-up screen is created the defined interaction pattern is applied.

There are two types of description formats for interaction patterns

- Text
  Sometimes the interaction pattern just consists of text. It is the responsibility of the functional designer that the behaviour described in the interaction pattern is supported by the mock-up.
- Graphics
  When the interaction pattern contains graphics, these graphics probably cannot be copied directly because a client will wish to see the pattern applied to his specific project. The solution is to mimic the layout and apply the current project's data.

Deal With Open Issues

This is a cross section problem which might be encountered in applying a Functional Design Pattern. Some Functional Design Patterns documents contain open issues which could not be solved at the time the pattern was written or they might deliberately leave a particular design choice up to you. This is exactly what should be done, the designer should make his own design choice. The result of a particular design choice should be documented in the usage and maintenance phase (4.8).

Document The Used Patterns For The Technical Designer

Because a functional design is used by a technical designer when it is completed to derive a technical design, used Functional Design Patterns have to be documented. This ensures that a technical designer will not needlessly re-invent the wheel while trying to solve functional problems for which a technical guideline was already created and documented in the Functional Design Pattern.

4.6 Realisation

This phase of development is concerned with the creation of a technical design document and the actual implementation of the contents of this document. Functional Design Patterns play an important role in speeding up the translation from functional design to technical design and the translation from technical design to code.

Fig. 20 Dependency graph for realisation phase

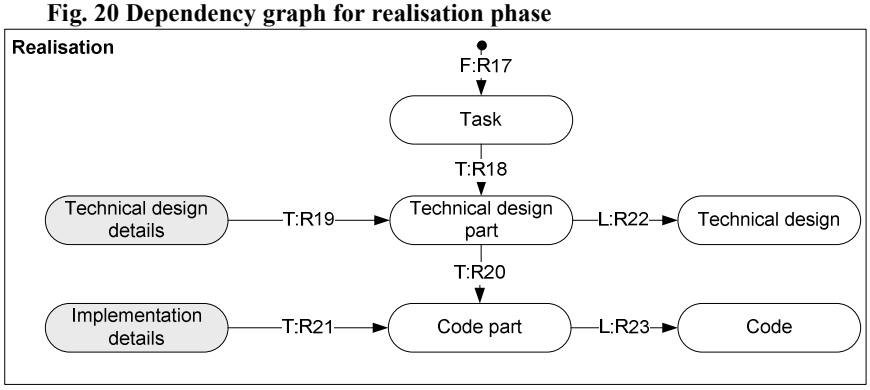


Table 6 Rules and activities in realisation phase

Rule	Person	Involved activities
R17	Team leader	Divide functional design in tasks
R18 R19	Technical designer	Write the Technical Design for a specific part • Check technical design details in patterns • Assess advantages of copying versus referencing • Refine technical design details in the technical design
R20 R21	Technical designer	Develop software using implementation details
R22	Team leader	Merge technical design parts into technical design
R23	Team leader	Merge code parts in code

We will now discuss the activities in detail.

4.6.1 Divide Functional Design

Although there is no strict need for the software engineer to know about functional requirements when they implement a technical design, Lauesen [22] showed that engineers who better understand functional requirements make better decisions when programming software that should fulfil these requirements.

Therefore we propose the team leader should communicate the functional design as well to the software engineers.

At Quinity this is already in effect because writing the technical design is not done by a single person. Usually there are a number of software engineers which create the technical design collectively. A software engineer receives the assignment from the team leader to build certain functionality and the accompanying functional design. The developer will write the technical design for this specific part. All these parts are merged later on.

Functional Design Patterns have impact on the division of the functional design document. Because we want the software engineer to work on a more or less isolated part of the design, The team leader should take into account *where* and *if* a Functional Design Patterns can be split and which parts should be left intact.

4.6.2 Write Technical Design

This part applies to every software engineer and is executed in parallel.

Check Technical Design Details In Patterns

As explained in the previous section the used Functional Design Patterns are documented for the technical designer. For every Functional Design Patterns that is mentioned in the functional design, the technical designer needs to check the Functional Design Pattern document if specific technical details are given. These details may include class diagrams, memory models and code snippets and the reasoning behind them.

The technical designer should copy or reference the reasoning because they clarify a lot for developers. These two possibilities are discussed below.

Assess Advantages Of Copying Versus Referencing

There are a number of forces that should be taken into account to decide between copying the described technical design details and just referring to the Functional Design Pattern document from within the technical design.

- Pattern changes
  If the used pattern is not final and still under heavy modifications because of pattern design sessions it will be wiser to copy the current documentation to the technical design.

- Amount of details
  When there are only a few technical remarks, it might not be worth the effort to create a reference. On the other hand when there are a lot of details an external document may be better to prevent cluttering.

Refine Technical Design Details In The Technical Design

Although technical details are given, applying them in the current project will demand further refining in most cases. This includes renaming specific attributes from the class diagram for instance.

The class diagram is usually a further specification of the data model from the functional design. When creating the (or a part of) the class diagram the recommendations from the Functional Design Pattern should be followed

4.6.3 Develop Software Using Implementation Details

The software engineer actually implementing the technical design is usually the same person who wrote the part of the technical design. The software can be developed by programming as done normally but the code should be implemented keeping the implementation details as mentioned in the technical design in mind.

4.6.4 Merge Technical Design Parts

After the implementation of a technical design part, the team leader should merge the parts into the complete technical design document.

4.7 Acceptance Testing

Functional Design Patterns are of limited use during acceptance testing. Their influence is indirect. Functional Design Patterns indirectly guide testing because the way the software should behave when implemented correctly is described in the Functional Design Patterns which are combined in the functional design. The incorporation of patterns to develop a functional design will probably reduce the amount of errors that will be found during testing, because the patterns describe proven efficacious methods.

4.8 Usage And Maintenance

The last phase of the development cycle of the information system is the actual usage by the client.

Fig. 21 Dependency graph for usage and maintenance phase

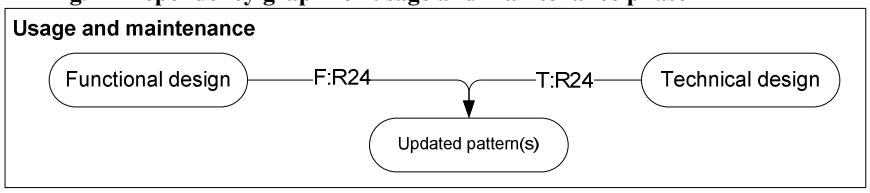


Table 7 Rules and activities in usage and maintenance phase

Rule	Person	Involved activities
R24	Team leader; Functional designer	• Update the pattern repository with project experience

We will now discuss the activities in detail.

4.8.1 Update the pattern repository with project experience

There are a number of possibilities that may have occurred during the execution of a project:

- Functional Design Patterns were chosen during analysis and design but their usage did not bring about the intended result.
  It is important to document the experience within the pattern
- A perfect fit with chosen patterns was achieved
  Next to documenting negative aspects, like when not to use the pattern it is also vital to note when a pattern can best be applied. The more known uses, the higher the chance a functional designer will identify it in the analysis phase of a subsequent project.
- The used patterns were altered
  The implementation that was built using Functional Design Patterns may have altered the pattern to better fit the need of the specific project.
  In this case it is wise to update the pattern library, maybe a domain level pattern can be distilled from the alterations.
- No patterns were implemented. If this is the first time a project was executed in a particular domain, it is to be expected that no patterns were available to guide functional design. This is the time to try and extract new generic functionality and develop a new Functional Design Pattern.

This concludes the software development process using Functional Design Patterns. After the deployment of the application, the client will definitely come up with

request for changes and new features; these will cause the entire process to be restarted. A new cycle will begin.

4.9 Example Of An Online Banking System

In this section we demonstrate some phases of the method we just described on a fictitious example of an online banking system.

4.9.1 Acquisition

For the acquisition phase we describe which functionality can be recognised by an experienced functional designer.

Suppose we have been given the assignment:
"Develop a very basic online banking system for client X"

The assignment is accompanied by the following case description:
"The banking system should support multiple users: customers, managers and tellers. Customers can login to view their own accounts. They are able to view the transaction history and to execute new transactions, like transferring money from one account to another. Bank tellers can do these transactions for every customer. Transactions are not executed directly because they need to be approved first. On top of this bank managers can add new accounts and remove superfluous ones."

We have to be able to tell the client if we want to accept the assignment. To do this we have to know which kind of complex requirements we can anticipate and how much time they will cost to implement.

When we analyse the case description an experienced designer might see the following concepts:

- Functionality discriminated for multiple users
- A flow of a transaction object.

Using only these statements it is already quite possible to recognise functionality that has been built before. Functionality discriminated for multiple users is something which returns in a lot of administrative applications as well as workflow.

An experienced functional designer will recognise this functionality as Authorisation and Workflow. He should check the latest information regarding the time it costs to implement this functionality in the respective Functional Design Patterns.

4.9.2 Requirements

From the requirements phase we demonstrate how gathered requirements are transformed into a conceptual architecture which in turn results in a set of candidate patterns.

Gather Requirements

The results from the requirements sessions at clients result in the following requirements (Table 8). Requirements are denoted as R_x

Table 8 Requirements for the online banking system

Requirement	Description
R1	The system is accessible online
R2	The system supports multiple users. Initially the system has three types of users: customers, tellers and managers.
R3	Every user must login before having access
R4	Each customer can have multiple bank accounts.
R5	Customers are able to view their transaction history
R6	The transactions can be searched and sorted
R7	Customers can transfer money from their own account to another account. This also includes transfers between their own accounts.
R8	Customers can edit or delete previously declined transactions.
R9	Tellers can do everything a customer can, but for every customer account.
R10	Tellers can approve or decline transactions.
R11	Managers can do everything a Teller can.
R12	Managers can add new accounts
R13	Managers can delete accounts
R14	No transaction is executed immediately. Transactions need approval of Tellers.
R15	Cancelled transactions are not deleted, but may be altered by Customers.

Conceptual Architecture

Next we analyse these requirements and create a conceptual architecture that will serve as a starting point for the functional design.

The first step is deriving information functions from the requirements. R3, every user has to login to have access to the system defines the information function "Login". Iterating over Table 8 results in Table 9. Recognise that because Tellers can do the same actions as a Customer although for every Customer, a list of customers will be needed so a Teller can select a customer (IF12).

Information functions are denoted IF_x.

Table 9 Information functions for the online banking system

Information function	Description
IF1	Login
IF2	View transaction history
IF3	Search transaction history
IF4	Sort transaction history
IF5	Approve transaction
IF6	Transfer money
IF7	Edit declined transaction
IF8	Delete declined transaction
IF9	Add account
IF10	Delete account
IF11	View account overview
IF12	View customer overview
IF13	Search customers
IF14	View transaction

Conceptual problems are found by grouping similar information functions from Table 9. We see multiple information functions for accounts which can be grouped as "account management"

Multiple information functions work on transactions as well. Add and deleting transactions are basic actions that are grouped as "transaction management"

Approve and decline though, are no basic actions on an entity, when we think about it, they do nothing more than change the state of a transaction, it is either declined, or accepted. This delivers a new conceptual problem "status tracking"

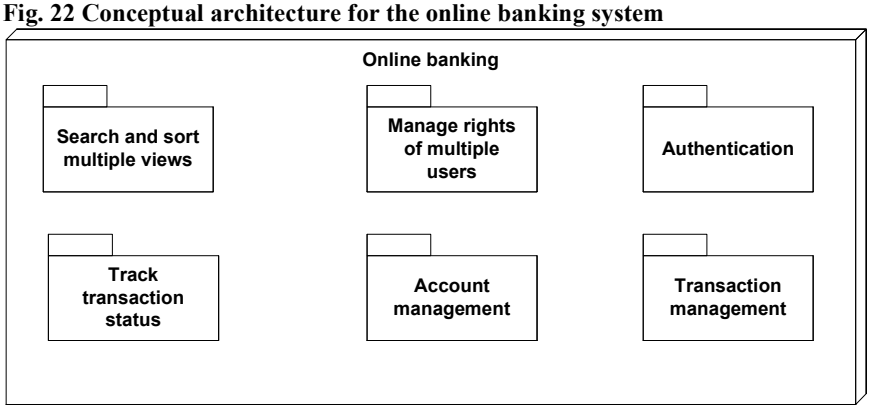
As can also be seen from Table 8, customers, tellers and manager cannot perform the same functionality within the system we will also need some kind of functionality that manages this. Functionality concerning logging in users is not part of this same group, because authorisation and authentication are two different problems.

Multiple views need to be presented for example account overviews and transaction history. All views need the same search and sort functionality. This can be grouped as the conceptual problem "search and sort multiple views"

The different users of the system are not conceptual components themselves because they are not responsible for actions within the system, the can execute them, but organisational responsibility is left outside of conceptual components.

We derive the conceptual architecture by creating a diagram of the conceptual problems

Fig. 22 Conceptual architecture for the online banking system



To demonstrate the selection of patterns we return to the online banking case. The domain for this project is banking for every conceptual problems, this might be different when a project spans multiple domains. We match the conceptual problems against our pattern library.

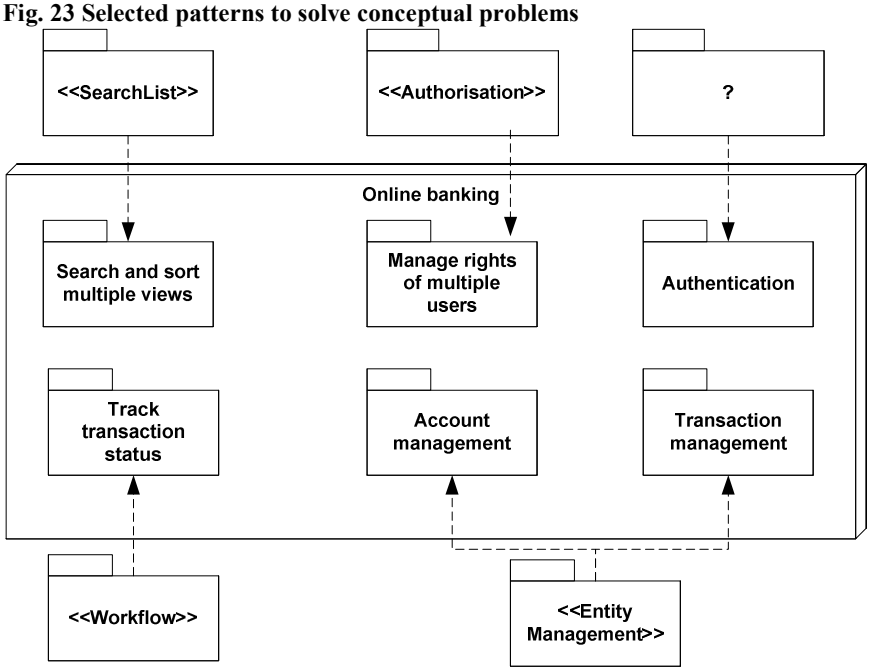
- Manage rights
  The problem is that we need multiple users who have different relations with accounts and transaction. The pattern aspect level can be classified as "security" or "authorisation". In this area we find the "authorisation" [3] pattern.
  Its description reads: "describes the common elements in administrative applications where not every user is allowed to execute the same functions"
  We add this pattern to our candidate set.
- Authentication
  The problem is letting users login to the system. The pattern aspect level can be classified as "security". Unfortunately there is no pattern yet which describes a general solution for this problem. The Authorisation pattern does not include functionality for logging in users. Therefore this conceptual problem will need to be solved using normal functional design.
- Account management
  The problem is creating and deleting accounts. The essence is the creation and deletion of an entity. Although a functional problem in the domain of banking, is really an aspect level problem. The current pattern library does not contain a functional design pattern at the domain level, but does provide an aspect level pattern: Entity Management [32].
- Transaction management
  Similar to account management, the Entity Management pattern applies here.
- Transaction status tracking
  The problem identified here is status tracking (where accept and decline manipulate this status).
  The pattern categories might be status tracking or transaction management. Unfortunately there is no pattern available in this domain and sub domain, that's why we look at the aspect level. This is where we find the "workflow" pattern

[30]. Its description reads “An entity in an information system passes through one or more states while multiple users treat the entity each from their own role (…)” A perfect match, we add this pattern to our candidate set.

- Search and sort multiple views
  The problems to be handled, searching, sorting of multiple entities are not particularly domain bound. We look at the aspect level for “search” or “display” or “sort” We can select the Functional Design Pattern “SearchList” [32] which handles these problems.

In this section we gathered requirements, analysed them and demonstrated the use of the pattern selection technique. We ended up with a candidate set of Functional Design Patterns which is passed on to the next phase of developing, the functional design phase.

Fig. 23 Selected patterns to solve conceptual problems



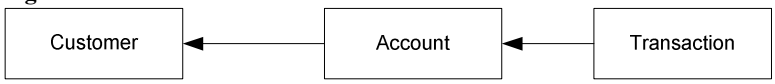
4.9.3 Functional Design

From the functional design phase we demonstrate how a data model can be developed by applying the patterns in order of importance and give an example of mock-up development.

Data Model

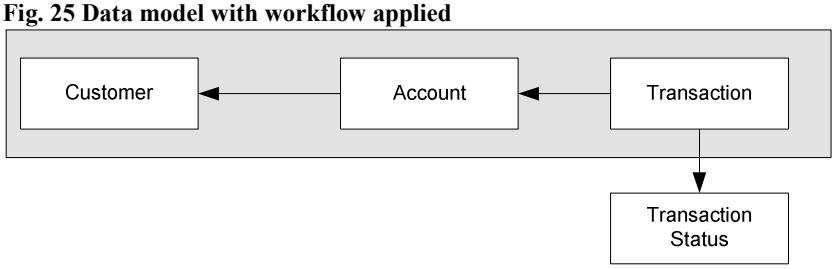
First we develop the basic data model. A customer can have multiple accounts which in turn can be associated with multiple transactions. The arrows denote a foreign key relationship.

Fig. 24 Basic data model



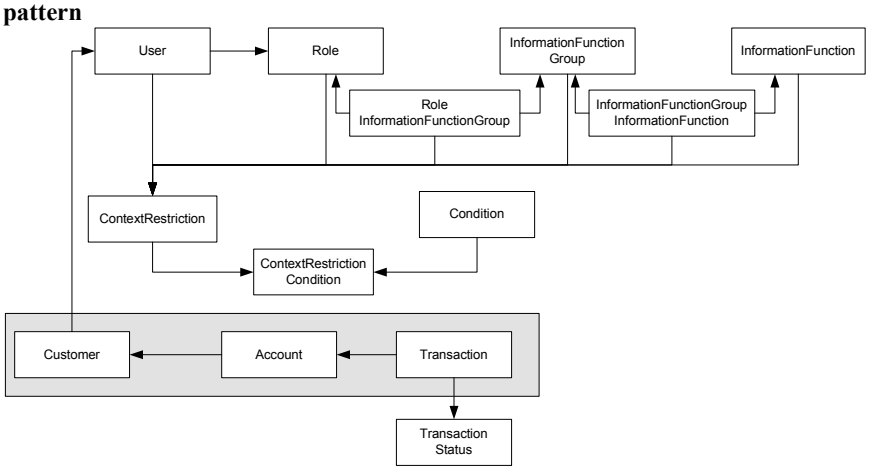
Now we apply the patterns in order of importance. We feel workflow is the most important pattern because it is responsible for transaction management. A banking system is nothing without transaction management. Workflow defines a subject and a subject status. The subject of workflow is the transaction. The only thing we need to add here is a transaction status.

Fig. 25 Data model with workflow applied



The next Functional Design Pattern to be applied is the Authorisation pattern. This pattern has considerable impact on the data model.

Fig. 26 Data model extended with authorisation pattern



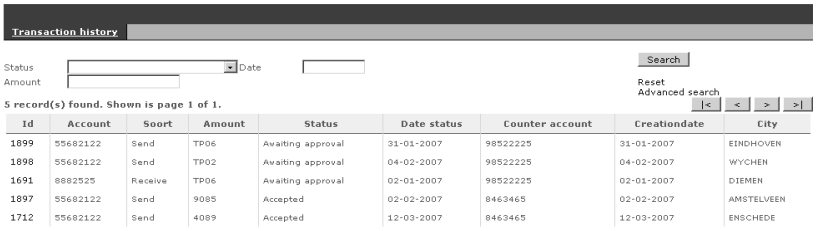
The third pattern that should be applied is the SearchList. The SearchList pattern does not contain any implications for the data pattern. The same goes for the Entity Management pattern.

After applying the patterns we should continue adding remaining entities that were not solved yet. In our case this was the Authentication part. This does not require a new entity, because login functionality can be part of the User entity.

Mock-up Development

For our banking system we would like to show an example of the way end-users will see their accounts history. The information functions from Table 9, Show transaction history is contained in the conceptual problem "search and sort" multiple views, which is solved by the SearchList pattern. We apply this pattern by mimicking the layout provided in the pattern.

Fig. 27 Mock-up example with SearchList applied



4.9.4 Realisation

From the realisation phase we give a short example of writing the technical design.

Technical Design

The team leader distributes the functional design amongst developers. Suppose we as a developer, get assigned to the status tracking information functions.

In the functional design it is documented that this functional problem can be solved using information from the Workflow pattern. When we open this pattern we see that a class diagram and technical details have already been defined.

We assess between copying and referencing the reasoning about the details by concluding that the workflow pattern is already quite old and will not be updated or changed that much, therefore the risk of including outdated information is not that high. Also the amount of details is quite large, so we choose to reference the details instead of copying them.

The class diagram is copied and adapted to the online banking case. Please note we do not have a figure here because the workflow pattern is an internal document and the class diagram is not available publicly.

5. Validation

This chapter validates the method from the previous chapter by comparing it with the criteria developed earlier, matching it with experiences of functional reuse at other software development companies and by evaluating it with the opinion of Quinity designers.

5.1 Approach

Validating the method we are faced with a dilemma. We want to validate if the method would work in practice but due to time constraints a real evaluation is not possible.

Validating the method would require a *test and control* situation where similar projects are executed where one of the project teams makes use of the method and the other does not. Such a test and control validation would show if the method improves the incorporation of Functional Design Patterns; identifying where the method can be further improved. Unfortunately to executing such a validation in the context of this master thesis would demand an unfeasible amount of effort and time.

As an alternative we validate the method by

- Case studies at external companies
  The objective of the case studies is to find out how reuse of functionality is tackled outside of Quinity. We interviewed four external companies and enquire about their experience with functional reuse. The interview questions can be found in appendix 7.1.
- Designer opinions
  We let a functional and technical designer, who just finished a project using a Functional Design Pattern, compare their approach with our proposed method to find possible additions.
- Comparison with the criteria
  Lastly we compare the method to our original criteria to determine if the objectives have been met. This provides a certain validation the case studies cannot.

5.2 Case Studies

Because the method that was developed in chapter 4 could not be used in real world projects during the execution of this research we try to validate the method by comparing it with (structural) reuse methods from other companies[2].

5.2.1 Case Study One

The first case study was conducted at a large business administration software developer. The company employs 30.000 people worldwide, 11.000 of which in the Netherlands. The typical client can be described as bureaucratic governments and semi governments. Although this is the typical client, the company does not specifically target one domain.

The company tries to standardise and certify procedures, processes and wherever possible the people who are responsible for executing them.

At the company new applications are developed but the core competence of the company is maintaining and extending older administrative information systems. For the development of new software a development method based on Rational Unified Process [21] is used.

Reuse plays an important role within the company on all levels. The main reason for reusing different software assets is they believe it ensures higher quality, less work and better maintainability. It is not appreciated when a developer builds something from scratch. Because the company has been involved in an enormous amount of projects in the past, it is highly likely that certain functionality has been built or described before.

Sometimes it happens that some functionality is built twice but this will only be the case if the client had such different requirements that it could not be integrated in the older package. Determining what kind of documentation should be used, when two solutions were made for one problem is a matter of checking in which solution the problem at hand matches best.

Reuse is implicitly embedded in the development process; there is no prescribed method that says how and when to do it.

Most software is assembled by reusing different documents, models and diagrams. Experts in cost estimation try to detect large pieces of functionality that has been built before when they make the time and budget plan.

Essential in the reuse of these documents are the software architects. Each project team has a software architect who is responsible for the composition of the initial documentation set that is given to the software developers. He is the one that searches

[2] As a disclaimer it should be noted that the views and opinions expressed by the interviewees are not necessarily representative for the opinion of their entire organisation.

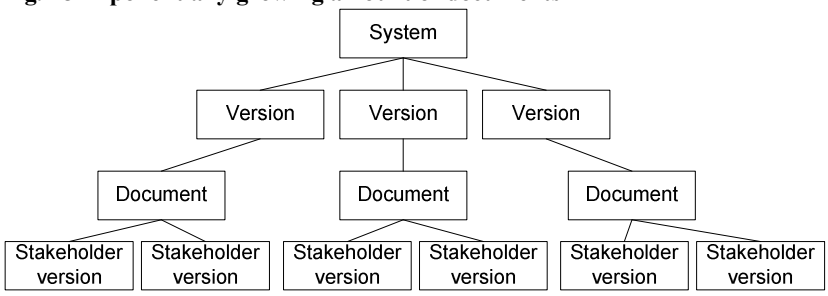
through the current documentation of older systems to see what kind of analysis, descriptions and code templates can be reused.

Although everything is documented, it is hard to find and possibly out of date. The reason is the amount of releases that are done every year for each piece of software. One system has different versions. Every version has many different documents and each document is differentiated for multiple stakeholders.

The vocabulary of the client is used in design sessions with the client and also in the documents that the client will see. But for the company themselves another version of the document is created, there is a mapping between

The amount of documents is illustrated in the figure below

Fig. 28 Exponentially growing amount of documents



It should be noted that although no specific tool for searching through documentation is offered, the architect does not stand alone. A special tool as well as a spreadsheet with a functionality/project matrix is available. To find out where certain functionality was used the company uses a spreadsheet. The sheet contains a matrix that shows what kind of functionality was built in which project. This is a useful source of information when starting a new project. Unfortunately the updating of the spreadsheet is sometimes a bit behind because of the busy schedule of project leaders. They are the ones expected to update this information.

The additional software tool helps the architect to guide the project in the best way. It uses environment variables like the experience of the proposed project team, the client and the type of project to determine the conditions the executing of the project should adhere to. For example when a group of novice developers will work on the project, the software will tell the architect to limit the complexity of the design as much as possible. The tool also delivers documentation for a standard process and recommended default solutions.

As far as the reuse of functionality is concerned we can say that it is mostly project documentation that is reused. There are no special reuse documents created after a project is finished, so documentation is not presented as a pattern or as bundled information as would be the case in a Functional Design Pattern.

Reuse on code level however is done with templates. These templates should be copied and refined where needed.

There is no top management support for reuse. According to the company this is not needed per se. Because of reorganisations the management changes, but the need for reuse stays. Reuse is implicit, it is expected from everyone to reuse as much as possible. The teams are responsible themselves for making the best use of available documents.

For the combination of different documents an interesting approach presented in the IBM book for e-business reuse [1] [19] is used. This is an "outside in" method that defines a linked system of patterns from the high to the low level. IBM defined business patterns that identify the interaction between users, business and data.

Selecting a business pattern (for example extended enterprise) drills down to the application level where an application pattern is selected (for example exposed broker). The application patterns are linked to runtime patterns which define how the application pattern can be implemented (for example via service oriented architecture). The last step offered by IBM is a product mapping from the runtime pattern to actual IBM software, like Websphere. This idea could be implemented for Functional Design Patterns as well by linking related patterns.

In the opinion of the company, experience of developers is very important. A new developer will never be able to achieve the same level of quality by just using documentation. Reusing a template is not enough. Experience is needed for analysis and the detection of reuse possibilities. That is why experts are used to create cost estimates and architects for the assembly of basic project documentation at the start of a project.

Table 10 Summary of case study one

Factor	Quinity	Interviewed organisation
Documentation of functional reuse	Functionality bundled in pattern form	Project documentation and templates
Pattern integration in method	The presented method	Implicit reuse, outside in combination method
Tool support	None	Software architect tool + spreadsheet
Team composition	Mix of experienced novice designers	Mix of experienced novice designers
Client	Medium to large sized Financial organisations	Large (semi)government organisations
Vocabulary	Use developer vocabulary	Two or more separate vocabularies
Company size	Small (<100)	Enterprise (10.000+)
Management support for reuse	Top management support	No management support
Development Standardization	Based on DSDM	RUP, Standardised processes, certified employees

5.2.2 Case Study Two

This interview was conducted at a large multinational company which has over 2500 employees in the consulting area and more than 50000 in the IT area. In the Netherlands these figures are around 1000 and 9000 respectively.

My contact person is an executive business consultant in the unit of public services. This unit is responsible for giving strategic advice to companies in the public service sector. He has previously worked closely in the IT area of the company where software for financial services is developed and recalls how he experienced working there.

Most projects are done for the banking and insurance companies. The typical project size is 900 function points, roughly accounting for 9000 development hours, where gathering requirements is not counted as a development hour.

The company works in standardised ways according to CMMI. Software is mostly written in .NET or Java for both platforms a standardised iterative software development process based on Rational Unified Process [21] is used. The process has the capability maturity level 3: "defined". The company does not restrict itself to RUP per se. When a client demands it, from time to time the linear waterfall model is used or even extreme programming.

Documents delivered during development are all the artefacts as defined by RUP completed with some company specific documents. The technical design is sometimes less detailed or skipped completely. This step is more client driven, if the client would like to see it, it is created. If the requirements specification and functional design, and global software architecture overview are of a sufficient detail and the code is documented thoroughly the technical design is seen as superfluous. The company trusts its developers to be able to develop the software without creating a technical design first.

Functional reuse in the company is done based on experience from previous projects. It is tacit knowledge. There are no specific documents for the reuse of functionality. Reusing functional solutions is left to the analysts and use case developers. Reuse of technical solutions is the responsibility of the software architect.

The company trusts the employees to recognise previously encountered problems and expects them to reuse the solutions if they were successful. The intelligence is located with the employees, not in the documents; therefore documents will never be able to replace experience.

The management stands behind this intelligence and will allow the creation of a knowledge repository if one feels it is necessary to be able to finish new projects quicker. This being said, there is no specific functional reuse incorporated in the development processes.

My contact admits that some generic patterns like authorisation are helpful but he claims there are not that many generic patterns. Patterns on the domain or project level are more useful but it is hard to draw the line, you can describe everything you encounter as a pattern, but it is not always easy to see when functionality can be reused again. If it is in three years, why would we want to put in the effort now?

One problem with functional patterns is that combining them is hard. It is all about a trade-off between different aspects. Using dynamic data structures in a

relation database management system for example make it is possible to add or remove attributes from an object on the one hand increases maintainability, but it greatly increases complexity of some other aspects of information systems. For example high performance management information or authorisation on attribute or even data level. Composition of functional patterns is a challenge and the trade-off will have to be made for every system again based on the client's priorities.

Another problem with the creation of functional pattern documents as identified by the company is that due to the fact that clients become owners of the source code, they do not wish for their expensive solution to be sold to another company for a lower price. They consider their software a key asset in gaining a competitive advantage and wish to keep this position.

My contact sees a risk in trying to fit the client's problem in a pattern. Once you learn about a few patterns, you start to see them everywhere and apply them excessively.

The usefulness of patterns in general depends on the way they are written and understood by the architect. If pattern documentation is clear and unambiguous the architect can apply them faster and in turn they will be used more.

The client's vocabulary is used in documents and requirements gathering sessions. When a client renames concepts from authorisation, like group to sector, this is the vocabulary that will be used in the documents.

The list of requirements is scanned by experienced designers for functionality that has been built before and the cost and time estimation is adjusted accordingly.

If another organisation or open source community develops a set of functional patterns that seem useful my contact feels his company will certainly try to use it in an attempt to gain maximum profit, but as for now they see no real point in developing it on their own and will continue to use ad-hoc reuse.

Table 11 Summary of case study two

Factor	Quinity	Interviewed organisation
Documentation of functional reuse	Functionality bundled in pattern form	Tacit, Project documentation and templates
Pattern integration in method	The presented method	Scan requirements list. Combining based on trade-offs
Tool support	None	Software architect tool + spreadsheet
Team composition	Mix of experienced novice designers	As defined by Rational Unified Process
Client	Medium to large sized Financial organisations	Mostly large financial service providers
Vocabulary	Use developer vocabulary	Always use client vocabulary
Company size	Small (<100)	Enterprise (9.000+)
Management support for reuse	Top management support	No management support
Development Standardization	Based on DSDM	CMM level 3 for different methods

5.2.3 Case Study Three

My contact is an experienced solution architect in the field of enterprise application integration. The company he works for employs approximately 5000 employees. The company does consultancy for the top 40 largest organisations in The Netherlands in the financial sector, telecom but also for (semi) governments. Most of the clients can be categorised as standardised and bureaucratic.

The day to day tasks of my contact range from business process consultancy to the more technical aspects of software architecture.

Although the company develops custom software as well; the business unit my contact is working in tries to minimise custom development. Instead they recommend and implement different types of middleware solutions to integrate applications. The goal is to reuse functionality of entire existing systems while integrating their data exchange via an Enterprise Service Bus. The systems communicate via exposed services; therefore this area is called Service Oriented Architecture.

As far as standardization is concerned, the company has developed its own method to iteratively develop Service Oriented Architectures. This starts at testing if a SOA is suitable for the organisation and goes on with developing the architecture, planning, realizing. The method ends in a continuation phase in which the changes made are kept up and running.

In the standard method for the development of SOA's, functional reuse is not integrated as a standard procedure and not enforced by management. It is implicit and mostly done by reusing solutions seen in other projects.

The company does have some documentation that can be regarded as a reusable artefact, the service descriptions. Services represent the functionality of the system. The documentation of a service is reusable when the same service is encountered at different clients. The documentation is written in a pattern form. Amongst other things it contains a name, goal, pre- and post conditions, non functional quality demands and the technical interface definition containing the way the service should be called and what its return values are.

Next to these service descriptions there is technical documentation to interface with commercial systems like SAP or Oracle because these systems are encountered very often. My contact identifies that it is hard to determine when functionality will be recurring, especially across business units.

The used (implementation) development method depends on the software factory. The company has multiple software development platforms that are supported. The functional description of the SOA is largely platform independent and the choice for a platform is based on what will best fit the client. On the technical level the company makes use of enterprise integration patterns.

The typical team that does Enterprise Service Bus implementations projects consists of a project manager, software architect and three to four software developers. The project manager is responsible for staying on schedule and within budget; the software architect is responsible for the content and quality of the

solution. In the enterprise architecture field these teams consist mostly of experienced employees. Different employees from the client organisation are involved as well. They range from business analysts to IT specialists. These are mostly very experienced people. The vocabulary of the client is used in all documents.

Table 12 Summary of case study three

Factor	Quinity	Interviewed organisation
Documentation of functional reuse	Functionality bundled in pattern form	Service descriptions in pattern form
Pattern integration in method	The presented method	Implicit
Tool support	None	None
Team composition	Mix of experienced novice designers	Mixed team of experienced people with different backgrounds
Client	Medium to large sized Financial organisations	Mostly large financial service providers
Vocabulary	Use developer vocabulary	Always use client vocabulary
Company size	Small (<100)	Enterprise (5.000+)
Management support for reuse	Top management support	No specific management support for reuse
Development Standardization	Based on DSDM	Custom standardised SOA development approach

5.2.4 Case Study Four

My contact is an entrepreneur. She owns a small company that delivers custom solutions for application integration. Service Oriented Architecture is his main interest. The company is hired for consultancy to guide and implement integration projects at clients. Typical clients are financial organisations like banks and insurance companies, utility and telephone companies as well as other companies that deal with mergers and fast changing laws.

Because it is not always possible to send the same person to the client organisations, the consultants need to be exchangeable and versatile. Therefore a standard development approach is needed. The company is still in the progress of developing a standard architecture development method, based on The Open Group Architecture Framework [29]. The framework defines a detailed method and a set of supporting tools to develop enterprise architectures.

Developing a Service Oriented Architecture is evolutionary. The goal is to integrate existing systems, and replace them over time. Creating a web service interface to disclose the legacy system allows the legacy systems to be replaced one by one at a time that is most convenient.

Every insurance company basically has the same basic steps in their core processes, for example invoicing and claims handling. Functional reuse is achieved by:

- Using reference processes.

- Reusing the specification for specific web services.

Good tools are essential to achieve this. All processes are modelled in a tool which can contain the diagrams as well as implementation details. When starting a new project, the reference processes also modelled in the tool can be loaded, be extended and refined. This makes reusing earlier documentation quite efficient and easy.

Reuse where experience is captured in documents is not seen as useful because you have to put in a lot of effort in a document which gets easily outdated. If you can use the documentation in projects immediately that is more useful, that is why the company chooses for model driven approach.

Employees from the client that participate in the project are experienced in their own software although mostly inexperienced in the field of SOA. They know the legacy applications and general information systems well, but are new to Service Oriented Architecture.

Because the client hires the company for advice on a matter they are not that familiar with. The company gets to propose the method that is used to solve the problem at hand.

The company of my contact currently only employs people with years of experience. Normally the company sends one or two consultants who analyse the current situation and design the new architecture. Actually building the interfaces is largely done via generators and by the client themselves. The company will help the client design the software and make sure it fits with the architecture (SOA) and reuses services.

The typical project size is not yet known because the company has just started, but it is assumed that they will be relatively large due to the nature of enterprise application integration.

When starting a new project, the company first organises a workshop to determine the exact tools, vocabulary, diagrams to use. The vocabulary is determined in cooperation with the client.

Table 13 Summary of case study four

Factor	Quinity	Interviewed organisation
Documentation of functional reuse	Functionality bundled in pattern form	Reference processes and service specifications
Pattern integration in method	The presented method	Integrated through model driven development
Tool support	None	Development method based on tool support
Team composition	Mix of experienced novice designers	Only experienced employees
Client	Medium to large sized Financial organisations	Medium to large sized Financial organisations
Vocabulary	Use developer vocabulary	Mixed vocabulary
Company size	Small (<100)	Small (<100)
Management support for reuse	Top management support	Top management support
Development	Based on DSDM	Custom model driven

Standardization		approach based on TOGAF

5.3 Designer Opinions

Around the time we started researching the possibilities to integrate the Functional Design Patterns with the software development method at Quinity a new project was started using a Functional Design Pattern. We enquired the responsible functional and technical designer about their experience with the use of this Functional Design Pattern in the execution of a new project. We compare our method with their approach.

The project is relatively small with about 500 development hours. The goal of the project is to enable employees of a company to claim expenses.

Early on in the project it was discovered that the workflow pattern was applicable. Both the functional designer and technical designer do not know exactly how they discovered that it was applicable.

The diagram style presented in the workflow pattern is not used, rather the functional designer made his own diagram. He pointed out that this was due to the fact that he did not know that he was supposed to and thought project documentation from other projects was the standard instead of the Functional Design Pattern. The result is that in the functional design of the project, we cannot recognise the workflow pattern.

The core concepts from the Functional Design Pattern were not copied into the functional design. Communication with clients was done in the client's terms because the clients find this easier. But gradually a vocabulary emerges which contains both terms from the developers world and the clients world.

The step "Document used patterns for technical designer" was not done. There was a lot of communication between the functional and the technical designer which made explicitly mentioning the workflow pattern in the functional design redundant.

The responsible technical designer had never built anything for workflow before. He was relieved to hear that there was a document available that explained him how workflow can be implemented. The technical design details were copied and refined as described in the method.

The designers support our suggestion that Functional Design Patterns should contain a section with time estimations. The estimation was now taken from other projects that incorporated workflow functionality.

The technical designer alluded that the effort it takes to read, understand and refine the patterns is sometimes equal or even more than the time it would take to build the functionality from scratch. He also identified that the "Usage and maintenance" phase as defined by us would be a welcome addition. He mentioned the need for game developers to do good post mortems of their productions, what went right, what went wrong, so the next time something is built this experience can be reused. He feels this applies to the usage of patterns as well.

From the designer opinions it becomes clear that: the direction we took is one appreciated by functional and technical designers. The conclusions we can draw from the interviews are

- The structure and direction of the method is appreciated by both functional and technical designers. They did not identify major revision points.
- Technical and functional designers approach Functional Design Patterns differently
- Some employees need to be convinced of the use of the extra effort using Functional Design Patterns take.
- The natural way of creating a vocabulary is combining terms from both worlds.

5.4 Comparison With Criteria

In this section we compare the presented method to the criteria from 3.6. We explain how well the method complies with each criterion; to provide a certain validation the case studies could not.

C1. Clearly phased
The presented method keeps the clear structure of the current Quinity development method and as such fulfils this criterion. It does however add the acquisition phase as a new step in development which was not really recognised as a phase in the explanation documents of Quinity.

C2. Systematic but not mechanical
The process is explained in a systematic way, but alternative paths are possible and where Functional Design Patterns falls short it is still up to the functional designers skill. Therefore this criterion is fulfilled.

C3. Clear division of tasks
For every phase the responsible persons are defined, as a result confusion regarding the division of tasks is clearly eliminated.

C4. Facilitate communication
In this thesis we solely focused on the common vocabulary to facilitate communication. We recommended combining the vocabulary of the client and the developer like we encountered in real world situations. We believe that guidelines for establishing the common vocabulary might need some more attention.

C5. Systematic pattern recognition
A systematic way of recognizing patterns has been described by proposing the deduction method from information functions to conceptual architecture and a matching process between the conceptual problems and the pattern library.

C6. Pattern combining
The method describes how multiple Functional Design Patterns can be combined by specifying which parts can be put together and how this should be done. Especially for combining multiple data patterns, the process of applying one pattern at a time has been described in detail. It was discovered that visually combining Functional Design Patterns does not work and does not bring about the desired effects.

C7. Complementary method
The phases of the current development method can be followed. The existing method can be applied when no patterns are available or the project does not lend itself for the use of Functional Design Patterns. The proposed method is an addition, it describes the contact points between the process and Functional Design Patterns but these steps can also be left out.

We believe the presented method adheres to all criteria that were developed in advance.

5.5 Discussion

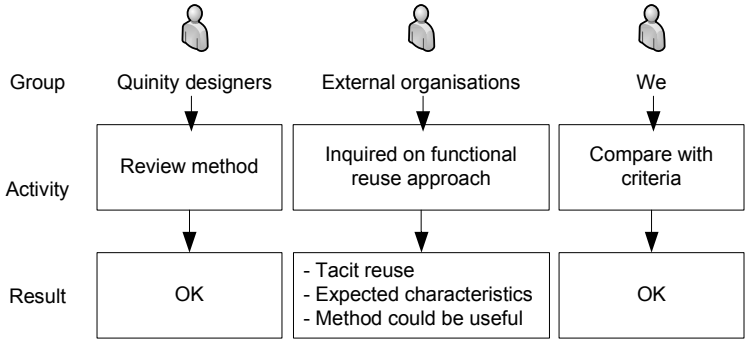
This section discusses the most important conclusions we can draw from the validation and the reasons behind them.

Originally we wanted to find out how other companies approach functional reuse. During our research it became clear that although all case studies show a standardised development method most do not have a structured approach to functional reuse. We found that technical design patterns and template reuse are embedded in the interviewed organisations but functional reuse is not. Thus, the interviews showed the real state of the practice regarding functional reuse.

This observation does however not validate our method. The only conclusion regarding our method is that the interviewees can see why functional reuse is useful and that a structured method as proposed by us would be helpful. Therefore we also went back to the real experts, designers at Quinity, who did validate the method by comparing their experience to the described method. Lastly we

The fact that the case studies showed no structured functional reuse approach makes the validation of our method harder, but does allow us to think about the reasons why organisations do or do not use Functional Design Patterns.

Fig. 29 Summary of validation



5.5.1 The Risks Of Tacit Knowledge

We observed that at the interviewed companies most functional reuse is based on experience: tacit knowledge.

Detecting reusable functionality is not standardised and mostly done by experienced designers. Only one organisation had a spreadsheet with projects offset against functionality; to quickly determine where functionality had been used. Unfortunately it went out of date because of the lack of a structured method that prescribes who should update this and when.

Tacit memory obviously poses a risk; when the experienced designer leaves the company the teams are left with nothing but the project documentation and no efficient way of recognizing where certain functionality was used.

Reuse based on project documentation limits the search space of the solution domain tremendously. In a large enterprise divisions will not know about each other's work and will as such miss reuse opportunities.

5.5.2 Service Specifications Are Similar Functional Design Patterns

The results seem to indicate that the functionality created for Service Oriented Architecture is well suited to document in a pattern form. The two organisations involved with application integration via web services have documented the services in a pattern form.

A possible reason is that web services are in essence developed to be reused in multiple application by different organisations. Designers do not wish to explain the way they work multiple times. In this sense the patterns are used as tool for knowledge transfer, just like Functional Design Patterns.

5.5.3 Possible Additions

The case studies showed two additions which might improve Functional Design Pattern documentation.

- Pattern trade-offs
  Adding certain trade-off characteristics to Functional Design Patterns might prevent implementing a pattern when it is not suitable.

- Pattern linkage
  The linking of patterns as hinted by case study one is a good idea and certainly a direction in which Functional Design Patterns could be evolved. This would involve adding related patterns to the Functional Design Patterns

5.5.4 Problems With Functional Reuse

Here we discuss the observed issues preventing widespread adoption of Functional Design Patterns.

- Documenting is time consuming
  Several organisations pointed at the problems of the pattern form. Developing separate documents is time consuming and they can get out of date quickly. The model driven approach as presented in the last case study ensures that the documents do not get out of date, because the documents are directly used in projects and if the projects change, the document changes as well.
  Functional Design Patterns are separate documents as well but we believe that we accurately counter this problem. Our method states that at the end of the project the team leader should be given the time to reflect on the usage of the patterns and update the pattern repository.

- Process improvement is not a top priority
  Because the development in organisations is going well using ad-hoc reuse there is no drive for further improvement.

- Identifying *when* functionality is reusable is complex
  The case studies showed organisations have difficulties determining when functionality will be reusable. This might be due to the fact that most organisations target multiple domains so functionality might not occur again for months or even years; estimating the advantage of creating a pattern is heavily influenced by this.

- Lack of top management support
  Management support helps to provide a companywide functional reuse integration method. At the companies which rely on implicit reuse, every team can implement reuse in another way, for example by only reusing their own projects. We saw no evidence of structured functional reuse at these companies as opposed to case study four which does have top management support.

- Copyright issues
  Another problem with the creation of functional pattern documents as identified by the company is that because clients become owners of the source code, they do not wish for their expensive solution to be sold to another company for a lower price. They consider their software a key asset in gaining a competitive advantage and wish to keep this position. We feel a feasible solution is introducing financial benefits for clients who do allow reuse.

- A common repository is lacking
  We believe the widespread adaptation of Functional Design Patterns suffers from the "chicken or the egg problem". When we explained the reasoning behind Functional Design Patterns and there possible advantages it became clear that most software companies did in fact thought about it, but were afraid to invest in it for various reasons but were willing to try them. Unfortunately a repository with Functional Design Patterns is not publically available.
  We can say Quinity is leading the way in Functional Design Patterns and cooperation with other software companies might be profitable for all involved.

5.5.5 Expected Organisational Characteristics

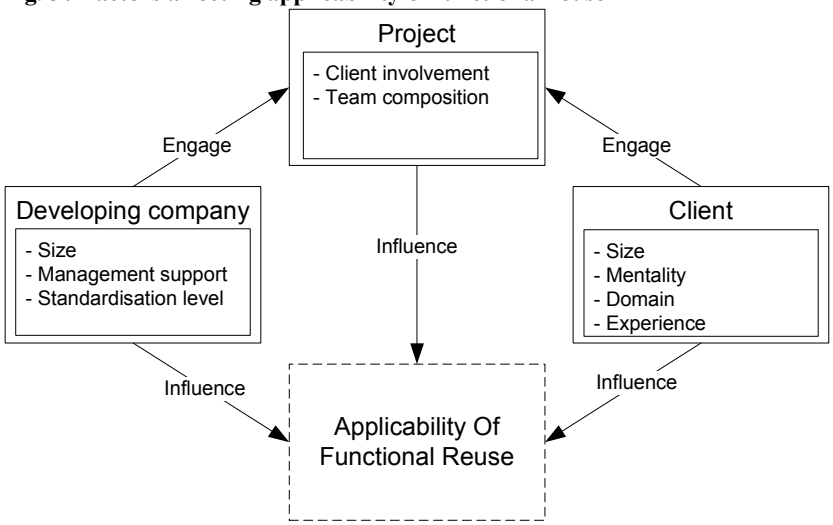
The fact that functional reuse is not commonly embedded yet gives us the opportunity to think about the ideal organisational characteristics.

The presented development method cannot be applied in all contexts. There are a number of inhibiting and enabling factors that determine in what extent a company or specific project will benefit from the use of Functional Design Patterns. To make reasoning about the context and the applicability of the method easier we divide these factors in three groups:

- the developing company
- the client company
- the project they engage together

The factors are displayed in
Fig. 30 and can be both inhibitors and enablers.

Fig. 30 Factors affecting applicability of functional reuse



Developing Company

- Size
  The size of the development company affects the applicability of the proposed method. Small companies have smaller projects in general and might not need a strong division of labour or can permit themselves to work in a less structured way. In our case studies the small companies could easily introduce a companywide policy concerning reuse. The large enterprises showed diversion between divisions.

- Management support
  The results seem to support [15]: Supportive management is essential when using a systematic reuse development method. Adopting systematic reuse takes upfront investments and therefore management support to provide the financial backing that is needed. Case study 4 shows that the organisation with the most structured approach has specific top management support for reuse.

- Level of standardisation
  Companies that have a standardised approach to software development will probably have more advantage of Functional Design Patterns. Companies with an ad-hoc approach to software development will most likely execute every project a bit different. In this case patterns will most likely be superfluous. Every interviewed organisation had a standardised approach to software development; therefore no conclusions can be drawn for this factor. No evidence was found that less standardised development methods can benefit more from Functional Design

Patterns, but the presented method requires a structured approach which will not work in an ad-hoc environment.

Project

- Team composition
  If the team consists of only experienced developers, they will probably already know how to handle certain functional problems and be more reluctant to adjust to the method proposed. Whereas new functional designers will have an advantage when they can make use of the knowledge documented in Functional Design Patterns. In the case studies as well as at Quinity we saw that a mix of experienced and novice developers makes effective reuse possible.

- Client involvement
  The amount of client involvement has influence of the use of Functional Design Patterns. The advantage of user participation and user satisfaction was demonstrated in [23]. Because a large part of Functional Design Patterns is about preventing common pitfalls and establishing a common vocabulary it would not be wise to exclude the client. The organisations we interviewed all struggled with the vocabulary to use; this explains all case studies have a different approach. We feel the most natural approach is to determine the vocabulary at the start by combining terms from the client and developer vocabulary so all parties can understand it. This is also endorsed by the designer opinions. For Functional Design Patterns this means that although some core concepts have already been defined they might have to be renamed.

Client Company

- Size
  Larger companies have a tendency to be more rigid and with a more strict hierarchy this influences the flexibility of a client. A small client will easier adapt to a method proposed by the developing company. The case studies showed that large clients might sometimes require certain development methods to be used, the Functional Design Patterns approach should then be adapted to these different methods. We saw no evidence that the size of projects has any impact on functional reuse.

- Mentality
  Client should be willing to accept outside advice and best practices as presented by Functional Design Patterns. The client should allow the solutions invented at their project to be reused at other clients. Case study two showed this as an inhibiting factor, but a solution to this problem might be to introduce a benefits scheme for companies who allow solutions to be reused.

- Domain
  If the client domain is in a known domain, the chance that functionality has been built before is higher. When functional designers and client companies have similar shared experience, this makes it more likely that a common vocabulary is already established. This makes the use of Functional Design Patterns easier.

"When you have an experience sufficiently in common with another person, all you need to do is re-evoke that experience within him" [8].

If the client domain is in a known domain, the chance that functionality has been built before is higher. We saw that case study 4, which has a focus on the insurance domain could reuse and refine reference processes.

- Expertise
  When there is not a lot of expertise on the client side (for example as shown in case study four regarding Service Oriented Architecture) the developing company is seen as the expert and Functional Design Patterns will be welcomed because they are proven solutions.

Summarizing we expect that the following characteristics will be an ideal context for functional reuse and the application of our method.

Table 14 Ideal characteristics for functional reuse

Factor	Ideal
Developer size	Small, one division
Developer management support	Specific top management support for functional reuse
Developer standardisation level	Highly standardised
Project client involvement	Involve client in establishing the vocabulary
Project team composition	Mix novice and experienced
Client expertise	Less experienced in project area
Client size	No guidelines perceived
Client mentality	Flexible, willing to reuse solutions and letting them be reused.
Client domain	Known or fixed domain

6. Conclusion

This chapter summarises the findings presented in this thesis. In the first section (6.1) we revisit the research questions and evaluate their answers. Section 5.2 compares the method with the initially developed criteria from 3.6 .The third section sums up some observations worth mentioning that were discovered during the execution of this research. Section 6.3 shows the practical implications for companies that want to implement the described method. The last part contains future work, research questions that could not be answered in this master thesis.

6.1 Research Questions

In this section we look back at the research questions from chapter 1 and discuss their answers. Before we answer the main research question we first provide answers to the supporting questions:

What Is The Current Way Quinity Uses Functional Design Patterns In Software Development?

Based on interviews with functional designers at Quinity it became clear that there was no unified way in which Functional Design Patterns were applied during the design of an information system. This resulted in the identification of the "semi structured approach" from Fig. 9; which represents the state of the practice.

To What Extent And In Which Way Is Pattern Reuse Implemented In Current Software Development Methods?

Using a literature study we learned that there are very few methods described to fully integrate the use of patterns throughout the software development process, this being especially true in the functional area. The reviewed methods (Pattern Oriented Analysis and Design, Pattern Driven Analysis and Design, Software reuse with analysis patterns and Building software with patterns) lean heavily on the technical side of software development and we have not been able to find actual documented cases of their use in case studies. As to their extent, the methods focus on the design phase and do not mention activities concerning realisation or deployment.

The methods did however provide useful ideas, information and guidelines which were incorporated in the proposed method.

How Can The Current Quinity Method Be Extended With Functional Design Patterns?

The current Quinity Method can be extended by incorporating the darker coloured blocks from Fig. 15. the artefact dependency graph. Only slight modifications to the existing process are needed. It shows that a software development method that is mainly based on LAD, with rigid characteristics is flexible enough to be extended with Functional Design Patterns.

When Should The Patterns Be Applied?

The method we present is clearly phased and shows in which phase and by whom the Functional Design Patterns should be consulted during software development. The method defines the interaction points between the pattern repository and the software process in detail. The most extensive interaction takes place during the phases: requirements, functional design and realisation. Acceptance testing does not involve Functional Design Patterns.

Which Elements Should Be Used?

In section 2.2 the overview of Functional Design Patterns was presented. Here we identified that Functional Design Patterns consist of multiple parts, the core concepts, information functions and the data model.

These correspond to their respective counterparts in the functional design. We believe that core concepts, information functions and data model should be copied in the functional design. This is not to say that no alterations have to be made, core concepts and information functions need to be refined for project specific details. A short overview was presented in Table 4.

How Do We Recognise The Possibilities For Reuse Of Functional Design Patterns?

We proposed the decomposition of problem in conceptual problem components inspired by the POAD process [2]. The resulting conceptual architecture serves as the basis for our pattern recognition process. The decomposition creates manageable pieces of functionality whose core problems correspond to pattern problem descriptions therefore allowing for better recognition of patterns.

Currently this is a manual process but in the future the process could be supported using software, which we will elaborate on in section 6.5.1. The fact that it is a manual process shows that experience cannot be replaced by Functional Design Patterns because experienced designer will be able to decompose functional problems easier.

How Can We Combine Multiple Patterns?

There is no real connection between patterns; it is rather the effect of applying them one at a time that allows multiple Functional Design Patterns to be combined in one functional design.

In the data model patterns should be applied starting from the most important pattern. For information functions we saw that combining multiple patterns is done by merging different information function segments into one information function. This is illustrated in Fig. 19.

This thesis has shown that combining patterns is something very different from composing them. Composing builds the entire software system from patterns where they are connected via interfaces. We tried to visually combine and compose Functional Design Patterns but this does not add useful information for the functional designer. The gain of representing Functional Design Patterns as black boxes of in- and output is questionable.

In Which Context Can Functional Design Patterns Be Used?

Section 5.5.5 revisited inhibiting and enabling factors resulting in Table 14. The ideal context for our Functional Design Pattern approach is a small company which has specific top management support for functional reuse to provide the entire organization with the same view on reuse. The focus should be on a fixed client domain. The project should be executed with a team consisting of experienced designers assisted by novice developers. There should be one single vocabulary, developed in conjunction with the client.

Main Question

"How can reuse of Functional Design Patterns be incorporated in the software development process?"

This question is answered by chapter 4 which describes a systematic method to incorporate Functional Design Patterns in the development process.

We believe that by following the presented method:

- It is possible to combine multiple Functional Design Patterns
- We can leave the current development method largely intact
- Reuse of elements described in Functional Design Patterns can be done without major modification
- It can be clearly pointed out in which phase we should make use of patterns
- Recognition of pattern applicability is possible in a systematic way

6.2 Main Conclusion

We believe that the use of the presented method to incorporate Functional Design Patterns in software development will lead to an improved and more efficient use of Functional Design Patterns in particular at Quinity.

The method is a practical guideline which clarifies the when, where, how and what questions regarding Functional Design Patterns and software development for all parties involved. Novice designers will gain the knowhow to use Functional Design Patterns in the development process thereby bridging a large gap between them and more experienced designers.

6.3 Implications

The method cannot be implemented immediately; we introduce some practical implications that need to be taken into consideration. For the developing company to adopt the incorporating of Functional Design Patterns in the development process this will have the following implications:

Gather Management Support

Without management support any systematic software reuse will fail [15]. Incorporating Functional Design Patterns in the software development process will need investments for the setup of a pattern repository, the development of patterns and the training of employees in the use of the presented method. These investments cannot be made without management support.

Develop A Searchable Pattern Database

A searchable database with Functional Design Patterns should be developed because it is not practical to read all Functional Design Pattern documents when the pattern repository becomes large. A heuristic based pattern search engine is necessary to fully leverage the advantages of Functional Design Patterns. Because there is no guarantee a pattern that matches a keyword in a functional problem.

Standardise And Adjust Pattern Structure

We witnessed that the Functional Design Patterns currently in use do not follow a very strict template. This makes it hard to correctly index the patterns in a searchable database. Furthermore a new pattern section "Estimation Details" (see the prerequisite section of the thesis) should be added to optimally support the acquisition process. Case studies also showed that adding pattern trade-offs and a related pattern section is a desired property of a Functional Design Pattern.

Educate And Convince Employees

Employees should be trained in the use of the proposed method to leverage its maximum potential. When no one is aware of the adaptation of Functional Design Patterns in the development process, no unity will be achieved. Reuse will not just happen because a pattern database is available. In addition we saw that some employees will need to be convinced that the extra effort needed to incorporate Functional Design Patterns in software development will pay off.

6.4 General Observations

Next to the answers to the research questions we made some secondary observations that are explained here.

Functional Design Patterns Communicate Knowledge

When looking at the use of Functional Design Patterns from a higher abstraction point we see that they try to create a more efficient software development process by

allowing functionality to be reused. This increase in efficiency is realised by the fact that senior designers will not have to explain everything to the novice designer; functionality is documented in patterns. The patterns are a way of communicating knowledge between experienced designers and novice developers.

Documentation Does Not Replace Experience

A remark we would like to pose is that Functional Design Patterns do not replace the need for experienced designers. Analytical skills still remain important, something also underlined by the interviews. Although the solution for problems is available, recognizing recurring functionality requires practice, experience and analytical skill. Novice functional designers will be able to learn the solutions to functional problems from them but for the recognition of patterns in a manual way experienced designers will still hold the advantage.

Tacit Reuse Poses A Risk

Tacit reuse as observed in the interviewed organisations poses a risk; when the experienced designer leaves the company the teams are left with nothing but the project documentation. It is highly likely that in a large enterprise divisions will not know about each other's work and will as such miss reuse opportunities.

Web Services Are Suited For Functional Reuse

The web services in a Service Oriented Architecture are well suited to document in a pattern form. A possible reason is that web services are in essence developed to be reused in multiple application by different organisations.

A Common Pattern Repository Is Needed

Functional Design Patterns suffer from the chicken and the egg problem. Although there is a genuine interest in them outside of Quinity, the lack of a common repository prevents widespread acceptance.

Visual Composition Of Functional Design Patterns Does Not Work

During out research we found visual composition like proposed by POAD does not work (see appendix 7.3). Most importantly because Functional Design Patterns are not blackboxes and not all provide class diagrams.

6.5 Future Work

Even though a number of research questions were answered, as always new ideas sprung from these. Questions we haven't answered and interesting ideas that came up but could not be done within the scope of the thesis are described here. We distinguish between research specifically for Quinity and research in general.

6.5.1 Quinity

Evaluation In Projects

Unfortunately the validation of the proposed method by using it in real projects was not within the scope of this master thesis. Empirical research in this area should prove whether or not the application of the proposed incorporation method for Functional Design Patterns is successful. A test and control situation could be created by executing similar projects where the first is executed with designers trained in the use of the method while the latter is executed without the method. The different results can provide reliable information regarding the advantage of the method.

Tool Support

We already identified the need for a searchable pattern database. But perhaps the described method could be supported by a more sophisticated software tool. Possible areas of use for this tool would be:

- Serve as a pattern repository
  All patterns will be stored in a central location
- Heuristic search and evaluation
  The tool allows the input of certain problem keywords, or whole case descriptions and searches matching patterns.
- Create basic data models from selected patterns
  The tool tries to combine the data models from the candidate patterns

The goal would be to support functional designers during the design process to more efficiently create the functional design from multiple Functional Design Patterns. We believe the development of such a tool might be a challenging but interesting task for further research.

Influence Of Pattern Order

In the method we discussed applying patterns one pattern at a time as described by Buschmann in [4]. But "most important" is a very ambiguous and certainly subjective statement. Therefore it would be interesting to see what the influence of applying patterns in reversed order actually would be and examine what are the defining factors that make a pattern "important".

Pattern Linkage

When a large database of patterns is available the internal linkage between patterns at a higher level and lower level can be made explicit in a similar way to the IBM method [19]. This means that when an aspect level pattern is selected,

automatically a number of other patterns will be removed from the list of applicable domain patterns or vice versa. This kind of behaviour would only be possible after a number of projects with Functional Design Patterns have been done so meta patterns will be become visible.

6.5.2 General Research Community

Develop A Common Repository

As identified during the validation of the method, an important reason for organisations to adapt Functional Design Patterns would be the existence of a common repository with Functional Design Patterns just like the ones that are available for technical design patterns. Research in this area might help to spread the knowledge contained in Functional Design Patterns.

Research Influencing Factors

We identified a number of factors that influence the level of applicability of the method. More research will be necessary to determine the exact consequences of the inhibiting and enabling factors mentioned in this thesis. In our method we show there are a number of options when establishing a common vocabulary, but at the time of writing it is not clear when to use a term from the clients vocabulary and when to use the developers preferred term.

Incorporation In Alternative Development Methods

This thesis demonstrated the applicability of Functional Design Patterns in a development process based on LAD and DSDM. It would be interesting to see if a "fit" between Functional Design Patterns and even more agile methodologies such as Extreme Programming can be achieved.

Development Of A Capability Maturity Model

It would be interesting to see if a capability maturity model can be developed for functional reuse. This maturity model could then be used by organisations to measure their progress and see which changes are needed to advance to the next level of maturity.

6.6 Final Remarks

Looking back I can say writing this thesis has been both an intriguing challenge and a satisfactory way of ending my efforts at the university.

The research model of this thesis can be viewed as “exploratory” [35]. To my knowledge there were no existing models for incorporating functional reuse. The outset and direction of the research was not always as strictly outlined as I would have liked it to be. Engineering the method gradually improved our knowledge of software development, Functional Design Patterns and executing research in general.

A point of critique and something I will take on in a different way in successive research is the validation of the engineered method. In my view a test and control situation could have yielded better conclusions although in the current context we did everything possible.

Above all I believe we have provided Quinity with a usable guide to use Functional Design Patterns in software development. The value for the research community outside of Quinity will have to prove itself in time...

Bibliography

[1] Jonathan Adams, Srinivas Koushik, Guru Vasudeva, George Galambos, “Patterns for e-business, a strategy for reuse”, McPress, October 2001

[2] Sherif M. Yacoub, Hany H. Ammar, “Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems”, Addison Wesley Professional, 2003

[3] Yoran Bosman, “Functioneel ontwerppatroon voor het aandachtsgebied ‘autorisatie’”, versie 0.8, May 2007. (Internal document)

[4] Frank Buschmann, “Building software with patterns”, European conference on pattern languages of programs, 1999

[5] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommelad, Michael Stal, “Pattern-oriented software architecture, a system of patterns”, John Wiley & Sons, 1996.

[6] Maarten Brak, “Functioneel ontwerppatroon voor het aandachtsgebied 'Tijdsafhankelijkheid van gegevens"', version 0.7, February 2007, (Internal document)

[7] Peter P. Chen, “The Entity-Relationship Model - Toward a Unified View of Data”. ACM Transactions on Database Systems 1: 9-36, 1976

[8] Alistair Cockburn, “Agile Software Development”, Addison-Wesley Professional, 2001

[9] Patrick van Driel, “Overzicht projectuitvoering, SDE standards and guidelines”, Quinity B.V 2005-2007, (Internal Document)

[10] L. Fokkinga, M.H. Glastra, H. Huizinga, “LAD, Het lineair ontwikkelen van informatiesystemen”, Academic Service Informatica, 2002

[11] Martin Fowler, “Analysis Patterns: Reusable Object Models”, Addison-Wesley, 1997

[12] E. Gamma, R.Helm, R. Johnson and J. Vlissides, “Design Patterns , Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995

[13] Andreas Geyer-Schulz, Michael Hahsler, “Software reuse with analysis patterns”, proceedings of AMCIS 2002.

[14] Henk Gommer, “Sense and nonsense: an evolutionary perspective on thesis writing”, Oxford University Press, 2002

[15] M.L. Griss, “Software reuse from library to factory”, IBM Systems Journal, vol 32, no 4, 1993

[16] Jonathan Grudin, “The Development Of Interactive Systems: Bridging The Gaps Between Developers And Users”, Computer 1991

[17] Robert Guitink, “Omschrijving van het gebruik van Functional Design Patterns”, Quinity B.V. 2006 (Internal document)

[18] Haitham S. Hamza, Yi Chen, “PAD: a pattern-driven analysis and design method”, Conference on Object Oriented Programming Systems Languages and Applications Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2005

[19] IBM, “DeveloperWorks patterns for e-business”, http://www.ibm.com/developerworks/patterns/, 2004 (07.2007)

[20] Jacob Kleerekoper, “Design Of A Pattern Definition Language”, Master’s thesis, Utrecht University, 2007.

[21] Per Kroll, Philippe Kruchten, “The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP”, Addison Wesley Professional, 2003

[22] Soren Lauesen, Otto Vinter, “Preventing Requirements Defects: an experiment in Process Improvement”, Requirements Engineering Journal, Springer-Verlag, London 2001

[23] Winston T. Lin, Benjamin B.M. Shao, "The relationship between user participation and system success a simultaneous contingency approach", Information & Management 37, 2000
[24] Jonathan C. McPhail, Dwight Deugo, "Deciding on a Pattern", IEA 20001
[25] George A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", The Psychological Review, 1956, vol. 63, pp. 81-97
[26] J.J.E. van Montfoort, “Functional Design Patterns, de implementatie van model-gedreven functionaliteit in een object georiënteerde omgeving”, Technische Universiteit Eindhoven, 2006
[27] Peter Nagel, “Patterns of model-based functionality in object-oriented software construction”, Master’s thesis, Utrecht University ,2006
[28] Object Management Group, UML 2 specification, http://www.omg.org/technology/documents/formal/uml.htm
[29] The Open Group Architecture Framework, http://www.opengroup.org/togaf, (17.08.2007)
[30] Quinity, “Functioneel Ontwerppatroon voor het aandachtsgebied workflow”, versie 1.01, 25 oktober 2006 (Internal document)
[31] Quinity, "Technisch Programmaontwerp Standaarden en richtlijnen", version 1.4.1, March 2007 (Internal document)
[32] Quinity Research and Development, “Functional Design and Functional Design Patterns: Theoretical Background”, Quinity B.V. 2003 (Internal document)
[33] Quinity Research And Development, “Functional Design Patterns, User Interface Patterns, Technical Design Patterns, (Quinity Framework And Reference Application v 4.3.8)”, Quinity B.V, 2002-2007 (Internal document)
[34] Niels Reyngoud, Jeffrey van Helden, “Functional Design Patterns”, Master’s thesis, Utrecht University, 2005.
[35] Pentti Routio, “Models in the research process”, http://www2.uiah.fi/projects/metodi/177.htm (30.08.2007)
[36] M. Shaw, D. Garlan. “Software Architecture: Perspectives on an Emerging Discipline”. Prentice Hall, 1996.
[37] Jeroen Snijders, “Functional Design Patterns”, Master thesis, Utrecht University, 2004.
[38] Jennifer Stapleton, “DSDM: The method in practice”, Addison Wesley, 1997
[39] Bedir Tekinerdoğan, “Formalizing Agile Software Development Methods”, Department of Computer Engineering, Bilkent University, Ankara, Turkey
[40] W.S Turner, R.P. Langerhorst, G.F. Hice, H.B. Eilers, A.A. Uijttenbroek, “SDM - System Development Methodology”, Rijswijk, 1990
[41] Martijn van Welie, Gerrit C. van der Veer, Anton Eliëns, “Patterns as Tools for User Interface Design”, Workshop on Tools for Working With Guidelines, 2000
[42] Martijn van Welie, Hallvard Trætteberg, “Interaction patterns in user interfaces”, PLoP 2000 conference

7. Appendices

7.1 Case Study Interview

Company

- Can you give a short introduction to your company and your function? (size, domain)
- How would you describe your typical client?
- How standardised would you call your company?

Software Development

- What development method you use?
- What is the typical project size? (function points)
- What is the default team composition? (experienced/novice mix)

Documentation

- Which artefacts are normally produced during development?
- How is functional experience documented to be reused?
- Do you use functional reuse documentation for cost estimation?
- Do you feel good documentation of recurring functional problems can replace experience?
- How do you ensure employees know all documentation with a large pattern library?

Functional Reuse & Integration

- How does your company attempt to reuse functionality? And why?
- In what manner is functional reuse integrated in the development cycle?
    a. When do you apply functional reuse?
    b. How do you decide between patterns?
    c. How do you combine multiple patterns in a functional design?
    d. Do you feel the order in which patterns are applied is of importance?
- In what way is reuse supported by management?
- Does your company provide tool support for the reuse of functionality?

Client Involvement

- How do you involve clients in the software development process?
- In what way do you communicate with the client?
- Which forces and tensions play a role in client communication?

7.2 POAD Pattern Diagram Examples

Fig. 31 Pattern level diagram

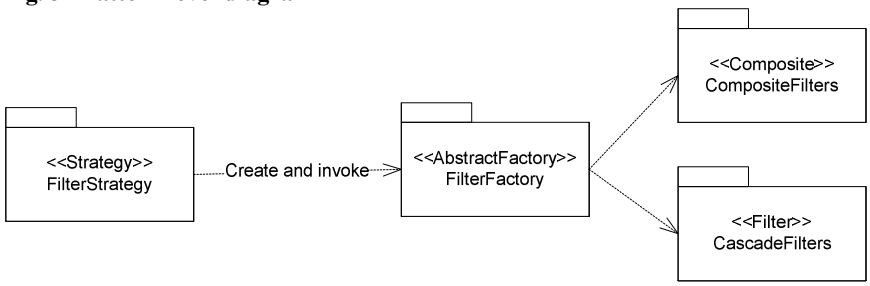


Fig. 32 pattern-level with interfaces diagram [2]

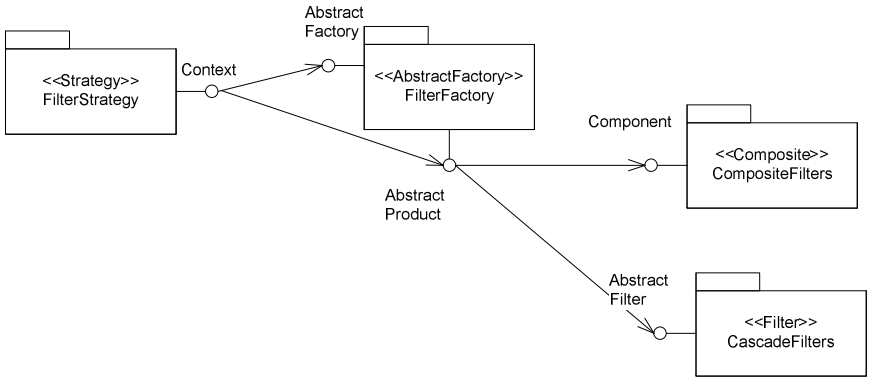
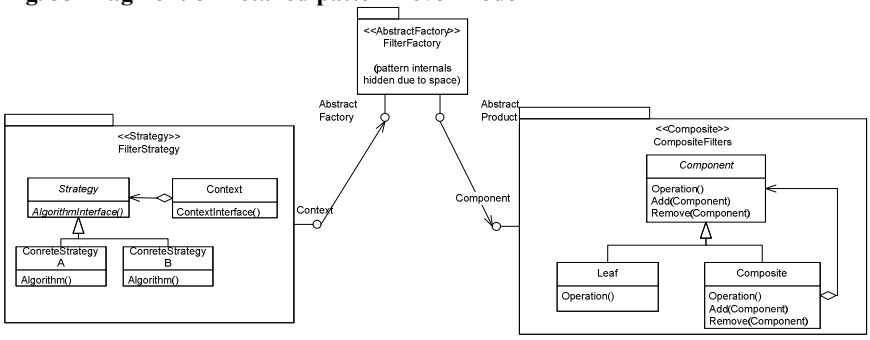


Fig. 33 Fragment of Detailed pattern-level model



7.3 Visual Composition Of Functional Design Patterns

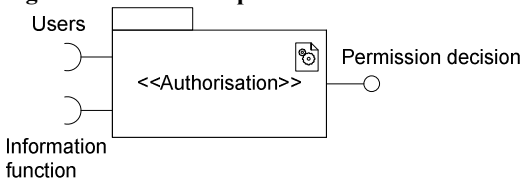
This appendix is complementary to paragraph 3.5 where we reason about the composition of Functional Design Patterns. Functional Design Patterns do not always offer interfaces for other patterns to use. They do however require other patterns or components as input to achieve certain functionality.

When looking at a Functional Design Pattern as a black box it provides services has inputs; core concepts or data; and outputs; for example transformed data and functionality is achieved by combining the concepts or data. The authorisation pattern combines users with information functions to check who has permission to do what.

We can view users and operations as input and the permission or obviously prohibition to execute a (part of) an information function as the result.

We choose to follow the UML2 notation recommendations on composite structure diagrams [28]. Required interfaces are shown with a socket and provided interfaces with a lollipop (or ball) icon. As a result the authorisation pattern can be depicted as follows (Fig. 34)

Fig. 34 Authorisation pattern interfaces



Unfortunately this does not apply for every pattern. There is not necessarily a relation of input and output between patterns where pattern A is input for pattern B. Therefore no interfaces can be defined and composition as defined by Pattern Oriented Analysis And Design offers does not suffice. Also POAD requires class diagrams for composition, not all Functional Design Patterns define these.

Next to this representing the user interface patterns and authorisation pattern in one diagram is difficult because in fact they to not function on the same level. The User interface views entities like Accounts, but every information function on an entity is checked by the Authorisation pattern, leaving certain options out. Showing this indirect influence is hard as can be seen in Fig. 35.

Fig. 35 POAD applied to Functional Design Patterns