

HOVEDOPPGAVE

Kandidatens navn: Erlend Naalsund, Ole Anders Walseth
Fag: Systemutvikling, Datateknikk
Oppgavens tittel (norsk): Beslutningstaking i Komponentbasert Utvikling
Oppgavens tittel (engelsk): **Decision-Making in Component-Based Development**
Oppgavens tekst:

This thesis aims at improving the reuse process at Ericsson AS, concerning reuse of code / design components. It also aims at exploring how the current process is working and is viewed by employees. A small survey will be conducted to achieve this. There are four main objectives: The first is to study the existing support in RUP for decision-making regarding evaluation and selection of COTS or internally developed components. The second is to study the GSN RUP adaptation regarding support for reuse and component-based development. The third is to define the criteria for evaluating components in the GSN RUP adaptation where an architecture is already developed. And the fourth and final objective is to evaluate and perform either a case study or an experiment on using a checklist used in different stages of the life cycle.

Oppgaven gitt: 21. januar 2002
Besvarelsen leveres innen: 17. juni 2002
Besvarelsen levert: 14. juni 2002
Utført ved: NTNU, Gløshaugen, Trondheim
Veileder: Reidar Conradi

Trondheim, 14. juni 2002

Erlend Naalsund

Ole Anders Walseth

Faglærer

Abstract

The definition of our diploma is written by our external advisor, Parastoo Mohagheghi at Ericsson AS.

Title: Decision-making in component-based development

Background: When developing software, several decisions should be taken in different stages of a project. The software development process in the GSN project at Ericsson is an adaptation of the Rational Unified Process (RUP). The GSN project has developed several components that are used internally in two applications and also uses externally developed components. A layered architecture based on reuse is developed and new features are continuously added to the applications. Decisions regarding selecting or developing new components or modifying existing ones are taken based on expertise, experiences from earlier phases of the project and requirements that should be fulfilled. The existing adaptation of RUP should be modified to deal with challenges of component-based development and reuse.

Thesis goals: This is a preliminary list and will be reviewed after a pre-study phase.

1. Study the existing support in RUP for decision-making regarding evaluation and selection of COTS or internally developed components.
2. Study the GSN RUP adaptation regarding support for reuse and component-based development.
3. Define the criteria for evaluating components in the GSN RUP adaptation where an architecture is already developed.
4. A template with criteria for decision-making can include a checklist to be used in different stages of the life cycle. Evaluate and perform either a case study or an experiment on using such a checklist.

Competence: The students have studied component-based development in the previous project during fall 2001. During this work, they should learn more about RUP, GSN RUP adaptation, criteria for future development based on reuse of already developed components and architecture.

Conditions: Students should study in periods at Ericsson in Grimstad. Their stay should be financed by NTNU. Working place at Ericsson during these periods is necessary. Due to conditions it may be necessary to coordinate with other students using the same working place.

In Retrospect:

The goals of the thesis have been changed by both project advisors Parastoo Mohagheghi and Reidar Conradi during the course of the work. This is due to internal Ericsson matter. Goals 1, 2 and 3 were achieved as planned. Goal 4, on the other hand, could be re-written as "Evaluate and perform an experiment by interviewing workers at Ericsson with relation to reuse."

Preface

The diploma is the follow-up to the pre-project from the fall 2001, entitled “Component-Based Development – Models for COTS / Software Reuse”. That paper serves as an introduction to the concept of reuse with a known architecture in mind. This paper focuses more on tangible solutions with reuse.

The main body of this paper explores how we can introduce reuse activities into a sound and stable architecture. We have been engaged by Ericsson AS to see how we can improve their guidelines concerning reuse by introducing reuse activities to fit their current architecture.

In addition, we have performed an experiment based on some hypotheses in a reuse context. The experiment is a survey where we have asked nine participants with experience with the software system from Ericsson to answer a questionnaire we prepared. These answers were later evaluated to find a conclusion on how the reuse practice at Ericsson can be made better.

We wish to thank our student project advisors, our external advisor at Ericsson AS, Parastoo Mohagheghi, and our head supervisor, professor Reidar Conradi at NTNU. We would also like to thank the anonymous participants who helped out with answering our questionnaire for the experiment.

Trondheim, June 10th, 2002

Ole Anders Walseth

Erlend Naalsund

Table of contents

ABSTRACT	I
PREFACE	II
TABLE OF CONTENTS	III
1 INTRODUCTION	1
1.1 SUMMARY	1
1.2 CONTEXT.....	1
1.3 REPORT STRUCTURE.....	2
2 SURVEY OF STATE-OF-THE-ART/PRACTICE	4
2.1 INTRODUCTION TO COMPONENT-BASED DEVELOPMENT.....	4
2.2 THE RATIONAL UNIFIED PROCESS	8
3 CONTEXT	36
3.1 REUSE AND COMPONENT-BASED DEVELOPMENT	36
3.2 CRITERIA FOR EVALUATING COMPONENTS IN THE GSN RUP ADAPTATION	37
3.3 REUSE RELATED ADAPTATIONS TO GSN RUP.....	40
3.4 SUPPORT CHANGES IN SOFTWARE DEVELOPMENT.....	50
4 RESEARCH APPROACH	51
4.1 WORK TO BE DONE.....	51
5 SURVEY OF REUSE ASPECTS IN SOFTWARE DEVELOPMENT	52
5.1 HYPOTHESES RELATED TO THE SURVEY.....	52
5.2 A SURVEY OF THE SOFTWARE DEVELOPMENT PROCESS AND REUSE ASPECTS AT ERICSSON AS	53
6 EVALUATION OF RESULTS	54
6.1 THE GENERAL QUESTIONS	54
6.2 COMPONENT QUESTIONS.....	56
6.3 REQUIREMENTS.....	60
6.4 APPLICATION FRAMEWORKS	61
6.5 GSN RUP WORKFLOWS	63
6.6 ARCHITECTURE COMPLIANCE.....	64
6.7 EVALUATION OF THE PROPOSED HYPOTHESES	66
7 CONCLUSION AND FURTHER WORK	69
APPENDICES	70
A. REFERENCES	71
A.1 RECOMMENDED READING	71
B. TECHNICAL INFORMATION	73
B.1 QUESTIONNAIRE OF THE SOFTWARE DEVELOPMENT PROCESS AND REUSE ASPECTS AT ERICSSON, GRIMSTAD.....	73
B.2 WORK SCHEDULE.....	82

C. ERICSSON RELATED INFORMATION 83

- C.1 GSN RUP..... **ERROR! BOOKMARK NOT DEFINED.**
- C.2 THE FIS SKELETON..... **ERROR! BOOKMARK NOT DEFINED.**
- C.3 CHECKLISTS..... **ERROR! BOOKMARK NOT DEFINED.**
- C.4 CHANGES ON THE FIS WITH RELATION TO REUSE..... **ERROR! BOOKMARK NOT DEFINED.**
- C.5 RESULTS OF THE SURVEY OF REUSE ASPECTS **ERROR! BOOKMARK NOT DEFINED.**

1 Introduction

1.1 Summary

The diploma explores the field of component-based development, and particularly reuse of software. Our joint venture partner, Ericsson AS, has engaged us to continue the pre-project we finished in December 2001.

The study we have done is both a literature survey and an empirical study (based on answers from a questionnaire).

The literature study part introduces component-based development briefly, as well as the Rational Unified Process (RUP). Knowledge about RUP is needed to fully understand GSN RUP, which is an adaptation of standard RUP at Ericsson AS. GSN RUP is then thoroughly described. This part is found in appendix C.1.

We later propose changes to GSN RUP to solve reuse issues, changes to the requirement specification scheme is also conducted. All this is taken care of in chapter 3.

The empirical study mentioned above is an interview with Ericsson personnel (with experience in GSN RUP applications) as a questionnaire with multiple-choice answers. These answers are documented and later evaluated. This study can be found in chapters 5 and 6. The conclusion of the work is found in chapter 7.

1.2 Context

The goals of this project is divided in four:

1. Expand our understanding of RUP
2. Use our acquired knowledge of RUP to understand more of an Ericsson adaptation of RUP, GSN RUP. Explore GSN RUP further.
3. Define the criteria for evaluating components in the GSN RUP adaptation where an architecture is already developed.
4. A template with criteria for decision-making can include a checklist to be used in different stages of the life cycle. Evaluate and perform either a case study or an experiment on using such a checklist.

As part one and two are rather theoretical, the main emphasis is on parts three and four, where the bulk of our own contribution is.

1.3 Report structure

The report is structured as described:

Abstract – Here is the definition of the diploma, our task.

Preface – A brief description on the contents of the paper, including the acknowledgment of the people that helped out with the work.

Introduction – The project is introduced with a summary, a context and a description of the structure of the report.

Survey of state-of-the-art/practice – Here is most of our literature study part, divided in two:

- Part one is an introduction to component-based development, with pros and cons of reuse, different types and so on.
- Part two is an introduction to the Rational Unified Process (RUP)

Context – This is the first chapter of our own contribution, where we try to find solutions to the goals of our diploma definition. The chapter is divided this way:

- First, a short introductory subchapter entitled “Reuse and Component-Based Development” angled on Ericsson’s situation
- “Criteria for Evaluating Components in the GSN RUP Adaptation”. Here we explore component selection and evaluation. Also, here is a general discussion on requirements and component selection/evaluation, and some discussion on how this could be adapted in GSN RUP.
- “Reuse-related adaptations to GSN RUP” contains the proposal for changes that should be made to the GSN project’s adaptation of RUP, GSN RUP.

Research Approach – Here is our project plan. It shows how we planned our work week by week. This plan was written early in the work.

Survey of Reuse Aspects and Software Development – Here is the information from the practical part of the diploma, the interview. Here are the hypotheses we were testing, along with a description of a survey that was conducted at Ericsson.

Evaluation of Results – Here, the results from the survey are evaluated and compared.

Conclusion and further work – This chapter concludes our work and gives our recommendation for further work.

Appendices – The Appendices gathers literature references, technical reading (like check lists and more), recommended reading and so on. Included in the appendices is most information from the survey conducted at Ericsson. The questionnaire we used for our interview based on the hypotheses is represented here. The results are also in the appendices, but in a separate chapter Ericsson Information chapter

(appendix C). That part of the appendices also includes an exploration of an adaptation of RUP at Ericsson, GSN RUP.

The reason for dividing the information like this is to shield Ericsson sensitive information that is restricted from the public. This information (appendix C) will not be presented in the publicized version of the thesis.

2 Survey of state-of-the-art/practice

This chapter explains some methods in software development that we will use in our research.

2.1 Introduction to Component-Based Development

[5][6][7][8] In the increasingly competitive area of software development, reuse of high-quality components must be considered not only as an asset, but an imperative part in assuring for your company survival. Reuse of components gives a software product of higher quality, which is faster and cheaper to create [4]. These advantages are considered key factors for success in software development.

The concept of software reuse has existed in software development since the late sixties. In 1968 Doug McIlroy proposed libraries of shared components, and developers have gradually accepted the value and usefulness of this concept. Throughout the seventies and eighties, software reuse and component-based development went through several evolutionary changes, but it was not until the nineties that companies started to take full advantages of these concepts. In the first decade of the third millennium, we finally see these concepts starting to mature.

Component based development differs from traditional development (where most or all of the code is created from scratch) in several ways. The first issue to discuss in context is software architecture. There exist several proposed definitions of software architecture [1], but in brief they encompass the following:

- The organization of the software system
- The selection of structural elements and their interfaces by which the system is composed
- The behavior of these elements as specified in the collaboration between them.
- The composition of elements into progressively larger subsystems

When we perform component-based software development it is important to create a high level architecture early in development. Having clearly defined how components should behave, interface and where they should be placed is an important tool in order for analysts and designers to select and/or design the correct components. In fact there are two issues that play an important part when selecting components to fit into your current project; the first is getting the component to fit and interact within the boundaries of the selected architecture, the second is making sure the component fulfill the requirements (functional and non-functional) it is supposed to.

The need to fulfill the requirements is necessary in all software development (component based or not), but in the context of reuse and the use of existing component we need to approach requirements in a slightly different way. In traditional development (not component-based), the usual approach is for the stakeholders to agree upon a set of requirements and then build a system that satisfies these requirements. The requirements may be changed during development, but these changes are almost always initiated by the customer

(wanting extra features he/she didn't think about initially). In component-based development it is much more likely that the developers initiate negotiations to modify the requirements. The set of existing components may not satisfy all the existing requirements or they may perform the requirements differently than initially planned. When this happens, the developer will have to renegotiate the requirements with the customers and eventually change the requirements, or possibly create new components from scratch. Because renegotiation of requirements is more likely, component-based development encourage developers to keep a closer relationship with the customer.

One typical characteristic of component-based development is that more time and effort is focused on the early phases of development. Finding and evaluating components will have to be performed early (typically during analysis) because your choice of components directly affects the design. As mentioned earlier, more effort will also be used on discovering and (re-) negotiating requirements. Another important feature of component-based development is the strong focus on non-functional requirements and testing. A major effort has to be put into checking how components perform, how well they interact, and in making sure they are stable (by themselves or in combination with other components).

For convenience it is useful to separate between components that are created within the company using them (internal components), and components created by an outside vendor (external components). With internal components you have the advantage of having first-hand experience with the component. This includes knowledge of how it performs, how it is constructed, how it can be modified, how stable it is and so on. Applying components from external vendors also have certain advantages. You get more components to choose from which gives you a higher chance of finding components that fit the requirements. You get standardized components that are often more suited to integrate with other systems etc. You don't have to take the full responsibility of maintaining and supporting the components, and so on. If you apply external components it is very important that the vendor can be trusted. This is particularly important if the components provided are not open source (open source = the source code is available). If the components are not open source, you will be reliant on the vendor to make all changes/adaptations on the components. This can make you very vulnerable, for instance if the vendor goes out of business or refuse to make the necessary adaptations.

For a company to be successfully taking advantage of reuse it is not merely enough to reuse code. One considerate field of reuse is reuse of design. This includes reuse of architecture, models, guidelines, patterns and so on. Quoting Ralph Johnson *"If you reuse code you'll save a load, but if you reuse design your future will shine"*

Virtually any part of a software development process may be reused. You may reuse requirements from earlier processes, you may reuse entire stages or phases from earlier projects, and so on. The potential in reuse is considerate and a company able to harvest this potential will have a great benefit.

2.1.1 Iterative vs. Waterfall paradigm

Traditionally, software development has not been iterative, but has used a sequential process called a waterfall model:

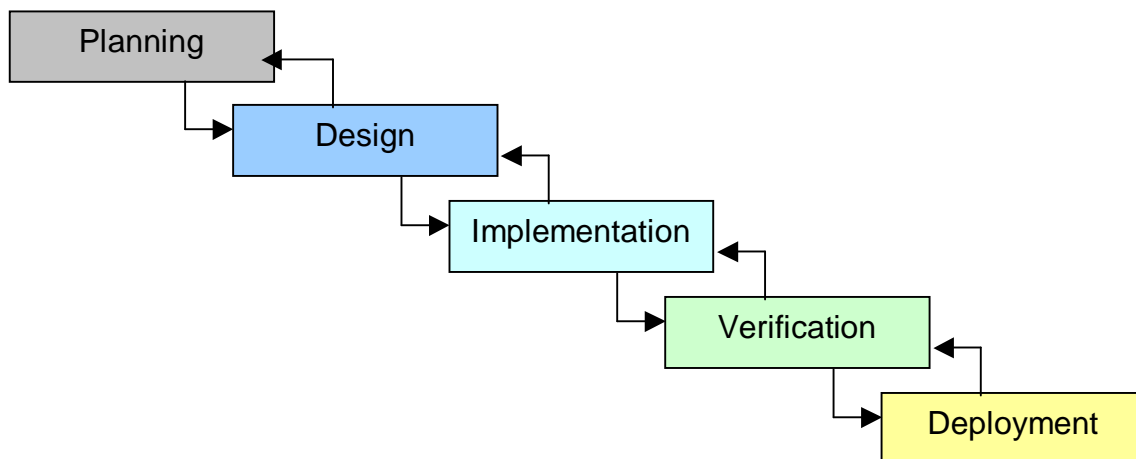


Figure 2.1-1 The Waterfall Model

Planning: Completely understand the problem, its requirements and constraints

Design: Design a complete solution

Implementation: Implement the solution

Verification: Verify that the implementation is correct

Deployment: Deliver/deploy system

This sequential model works well with small to medium size projects, but at a larger scale, this method has several shortcomings:

- Errors made in the planning phase are not discovered until the verification phase leading to a larger cost, exceeded deadlines, unhappy customers etc.
- Larger projects are very difficult to plan all at once, unforeseen events could happen in the course of the project
- Not very flexible in changing requirements, for customers, stakeholders or others

To solve these problems, RUP applies an iterative approach. This involves using a succession of waterfall models iteratively, meaning that the project has time to evolve. Problems that come unexpectedly are handled and fixed before the next iteration. Requirements can be changed continuously throughout the project. Also, reuse is handled at a higher level. An iteration is ideally 3 weeks to 3 months long, and a project of this scale normally has 3-9 iterations, depending on the size of it. The length of iterations stretch over much shorter periods of time than the waterfall project. Where the waterfall model is one big project, the iterative approach is built from a batch of mini-projects.

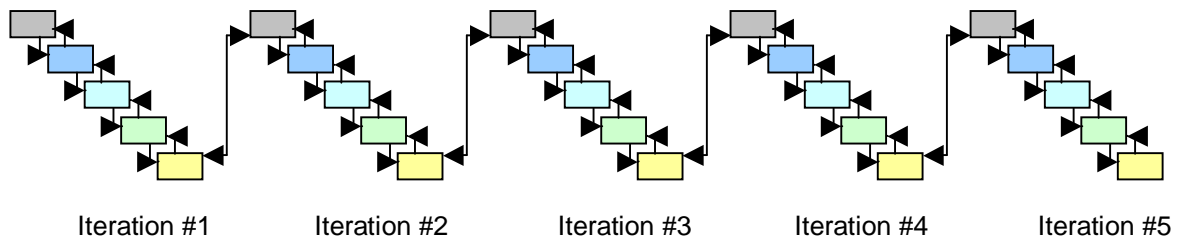


Figure 2.1-2 The waterfall model in iterations

Duration and Number of Iterations

Ideally, an iteration should span from 2-6 weeks, but this is dependent on the number of people working on the project, and the size of the project. For instance:

Lines of code	Number of people	Duration of an iteration
5000	4	2 weeks
20000	10	1 month
100000	40	3 months
1000000	150	8 months

The number of iterations in each phase depends on the size of the project.

LOW: Three iterations [0,1,1,1]

TYPICAL: Six iterations [1,2,2,1]

HIGH: Nine iterations [1,3,3,2]

The numbers indicate how many iterations are done in each phase [Inception, Elaboration, Construction, Transition].

A normal project has 6±3 iterations.

2.2 The Rational Unified Process

This entire subchapter contains information taken from [1].

RUP (Rational Unified Process) is a process for software engineering. It was created and marketed by Rational Software in 1995 from the merging of the Rational Approach and the Objectory Process version 3.8. It was not until 1998 RUP got the name Rational Unified Process, though. In the period 1995-1998 it was known by the merge name Rational Objectory Process.

A software development process, in this case RUP, has four roles:

- To provide guidance as to the order of teams activities in a project
- To specify which artifacts should be developed and when
- To direct the tasks of individual developers and the team as a whole
- To offer criteria for monitoring and measuring the project's products and activities

Having a well-defined and structured software development process ensures quality, a well organized project, budget balance, and upholding the deadlines.

RUP is built on the six best practices of software development:

- Develop software iteratively (as opposed to the traditional waterfall model)
- Manage requirements
- Use component-based architecture
- Visually model software
- Continuously verify software quality
- Control Changes in software

RUP is a process in constant change:

- Rational releases regular updates
- It can be delivered online using web technology
- It can be tailored and configured to suit the specific needs of a development organization
- Integrated with many tools.

RUP is currently used by large scale companies like Ericsson, Alcatel, MCI, Xerox, Volvo, Intel, Visa, Oracle and so on.

2.2.1 RUP Static Structure: Process Description

RUP is represented using four primary modeling elements:

- Workers: The *who*
- Activities: The *how*
- Artifacts: The *what*
- Workflows: The *when*

2.2.1.1 Workers

A worker defines the behavior and responsibilities of an individual or of a group of individuals working together as a team. The behavior is expressed in terms of activities that the worker performs.

Example of workers:

An individual acting as a “Worker: System Analyst” leads and coordinates requirements elicitation and use-case modeling by outlining the functionality of the system and delimiting the system.

It must be noted that the workers are not individuals per se; they are merely describing how individuals should behave in business. This means that a single individual may act as several workers, and for a worker to be performed by several individuals.

An example of this is:

BRIAN can be a “Worker: System Analyst” in the morning and as “Worker: Test Designer” in the evening.

BRIAN and SUSAN can both be “Worker: Test Designer”, but they are likely to be responsible for different classes etc.

2.2.1.2 Activities

Activities define the work performed by the workers. An activity is a unit of work that produces meaningful result in the context of the project. The activity typically includes creating or updating artifacts, such as models, classes or plans. To ensure that the activity is neither too small (which makes it easy to neglect), nor too big (hard to express), it should be in the range: a few hours -> a few days.

Examples:

Activity: Find use cases and actor -> performed by Worker: System Analyst

Activity: Plan an iteration: performed by Worker: Project Manager

2.2.1.3 Artifacts

Activities have input and output artifacts. An artifact is a piece of information that is produced, modified, or used by a process. Artifacts may take various shapes and forms:

- A model: such as the use case model
- A model element: such as a use case
- A document: such as a performance report
- Source code
- Executables

2.2.1.4 Workflows

A workflow is a sequence of activities that produces a result of observable value. There are six core engineering workflows and three core supporting workflows.

Phases Core Process Workflows	Inception		Elaboration		Construction			Transition	
	Business Modeling	[Dashed line rising]		[Dashed line falling]		[Dashed line flat]			[Dashed line flat]
Requirements	[Dashed line rising]		[Dashed line rising]		[Dashed line falling]			[Dashed line falling]	
Analysis & Design	[Dashed line rising]		[Dashed line rising]		[Dashed line falling]			[Dashed line flat]	
Implementation	[Dashed line rising]		[Dashed line rising]		[Dashed line rising]			[Dashed line falling]	
Test	[Dashed line rising]		[Dashed line rising]		[Dashed line rising]			[Dashed line rising]	
Deployment	[Dashed line rising]		[Dashed line rising]		[Dashed line rising]			[Dashed line rising]	
Core Supporting workflows									
Configuration & Change Mgmt	[Dashed line rising]		[Dashed line rising]		[Dashed line rising]			[Dashed line rising]	
Project Mgmt	[Dashed line rising]		[Dashed line rising]		[Dashed line rising]			[Dashed line rising]	
Environment	[Dashed line rising]		[Dashed line rising]		[Dashed line rising]			[Dashed line rising]	
Iterations	Preliminary iteration	Iteration #1	Iteration #2	Iteration #k	Iteration #k+1	Iteration #k+2	Iteration #m	Iteration #m+1	

Each core workflow covers a lot of ground. To break them down, RUP uses workflow details to express a specific group of closely related activities. It is also useful to express workflows related to iterations, expressed as “iteration plans”.

2.2.2 RUP Dynamic Structure: Iterative Development

The dynamic structure specifies how the process runs over time. Central issues in the RUP development process include: phases, milestones and iterations.

2.2.2.1 Phases and Milestones

The iterative RUP process is organized in phases, and each phase is concluded by a major milestone.

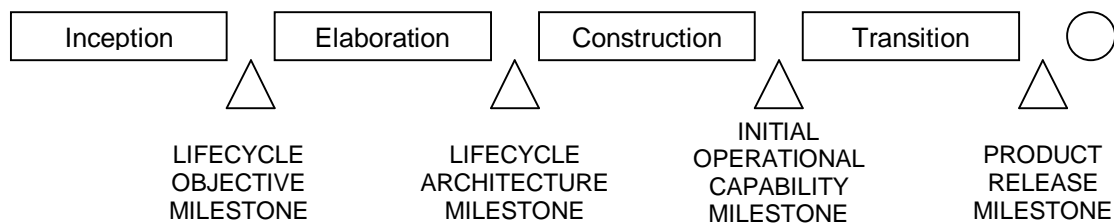


Figure 2.2-1 The four phases and milestones for the iterative process

The Inception Phase

- Establish the software scope and boundary of the project
- Discover the initial use-cases (primary scenarios of behavior)
- Establish overall cost & schedule for the entire project, and a detailed estimate of the elaboration phase
- Estimate risks

In this phase, the company needs to establish a business case for the system and delimit the project scope. All external actors must be identified, and the interaction with these must be defined. In the business case, the resources needed for the project is estimated. A phase plan, showing dates of major milestones is also included, along with success criteria and risk assessments. The outcome of the inception phase covers a project vision (a vision of the core requirements of the project, key features and main constraints), an initial use-case model, an initial business case (with risk assessment, project phase plan and a business model) and one or several prototypes.

MILESTONE: LIFECYCLE OBJECTIVE

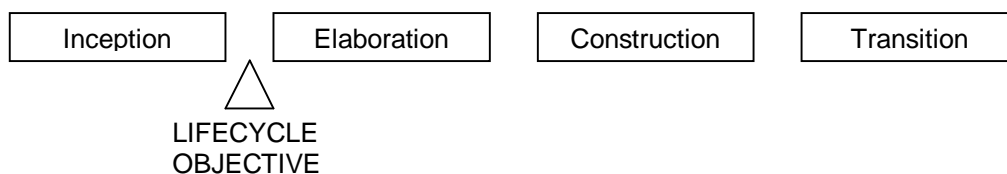


Figure 2.2-2 The lifecycle objective milestone

The first major milestone is the lifecycle objective milestone at the end of the inception phase. The evaluation criteria for the inception phase are:

- Stakeholder concurrence on scope definition and cost and schedule estimates
- Understanding the requirements as they are described in the main use cases
- Credibility of the cost and schedule estimates, priorities, risks and development process
- Depth and breadth of any architectural prototype that was developed
- Actual expenditures vs. planned expenditures

If the project fails to pass the milestone, it may be cancelled or considerably rethought.

The Elaboration Phase

- Analyze the problem domain
- Define, validate and baseline the architecture as rapidly as practical
- Develop project plan
- Eliminate high risk elements

In the elaboration phase, the purpose is to analyze the problem domain, establish an architectural foundation, develop the project plan, and eliminate the highest risk factors of the project. In order to accomplish these objectives, one must have a broad view of the project. Not too detailed knowledge, but a general understanding of the scope, major functionality and non-functional requirements such as performance requirements.

The outcome of this phase is a more complete use case model than earlier (at least 80% complete), supplemental requirements (non-functional and the requirements not associated with the use case), a software architecture description, an executable prototype, a revised business case (with risk list), a development plan for the overall project (replacing and revising the initial project plan) and an updated development case specifying the process to be used.

MILESTONE: LIFECYCLE ARCHITECTURE

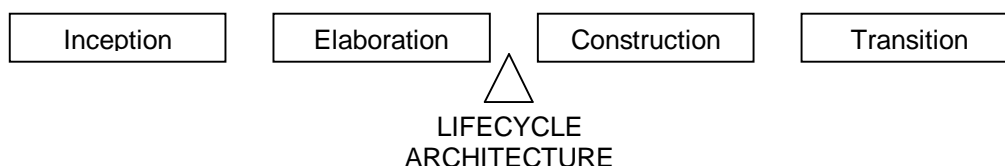


Figure 2.2-3 The lifecycle architecture milestone

The second major milestone is the lifecycle architecture at the end of the elaboration phase. The evaluation criteria from the elaboration phase are:

- Is the vision of the product stable?
- Is the architecture stable?
- Are the major risk elements addressed and resolved?
- Do all stakeholders agree that the current vision can be reached?
- Is actual resource expenditure versus planned expenditure acceptable?

If the project fails this milestone, it can be cancelled or rethought.

The Construction Phase

- Building the product and evolving the vision, the architecture and the plans until the project is completed
- Achieve adequate quality within time limits

In this phase, all remaining components and application features are developed and integrated into the product, and all features thoroughly tested. This phase is a manufacturing process with emphasis on managing resources and controlling operations to optimize costs, schedules, and quality.

The outcome of the construction phase is a product ready for the end-users. At a minimum, it consists of the software product integrated on the adequate platforms, the user manuals and a description of the current release.

MILESTONE: INITIAL OPERATIONAL CAPABILITY

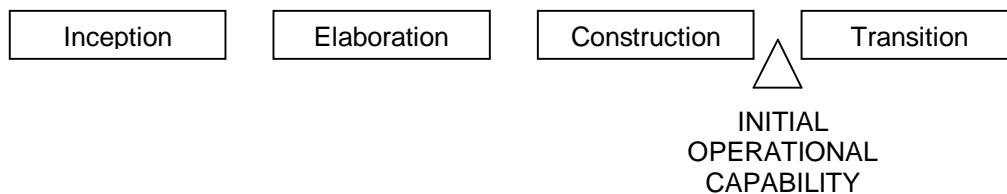


Figure 2.2-4 The initial operational capability milestone

The third major milestone is the initial operational capability at the end of the construction phase. The evaluation criteria from the construction phase are:

- Is this product release stable and mature enough to be deployed in the user community?
- Are all stakeholders ready for the transition?
- Are the actual resource expenditures still acceptable vs. planned expenditures?

Transition may have to be postponed by one release if the project fails to reach this milestone.

The Transition Phase

Move the software project into user community.

- Provide user support
- Train user community
- Market, distribute/sell product
- Achieve user self support

This phase's purpose is to transition the software product to the user community. Once a product is released, issues arise that require the company to develop new releases, correct errors or finish the features that were postponed. The phase is entered when a baseline is mature enough to be deployed in the end-user domain.

The transition phase focuses on the activities required to place the software into the hands of the users. Typically, it requires several iterations, including beta releases, general availability releases, as well as bug fix and enhancement releases. A lot of effort is put in helping user learn the product, by developing user documentation, training users, and also taking user feedback.

MILESTONE: PRODUCT RELEASE

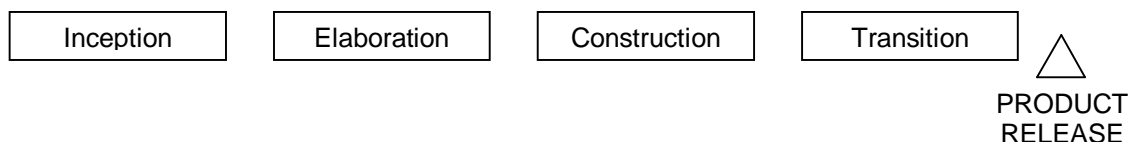


Figure 2.2-5 The product release milestone

The fourth and last major milestone is the product release at the end of the transition phase. The evaluation criteria from the transition phase are:

- Is the user satisfied?
- Are actual resource expenditure vs. planned expenditure acceptable?

2.2.3 Architecture – The Importance of Models

The focus on modeling is very important in RUP. A model is a simplification of reality, and it helps us understand what is too complex to understand in its entirety. We must apply different kinds of models in order to cover all aspects of software development. Care must be taken to ensure that all models are consistent.

2.2.3.1 What is architecture?

The system architecture is a specification of the high-level system structures (aspects/viewpoints to capture), its components, their interrelationship and their strategies and guidelines that governs the design and evolution of the system. The system architecture is an artifact. It is the result of the system design activity.

The purpose of the system architecture is to show what a system does and how it works, enable us to work on individual pieces of the system, make extensions on the system and to enable us to reuse parts of the system.

2.2.3.2 Definition of software architecture

The Rational Unified Process defines software architecture as follows:

- The organization of the software system
- The selection of structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaboration among those elements
- The composition of these elements into progressively larger subsystems
- The architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition.

Software architecture is also concerned with context: usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints, trade offs and aesthetics.

2.2.3.3 The 4+1 view model of architecture

In software development we must use different kinds of models to convey different aspects of the system. RUP suggests a five-view approach:

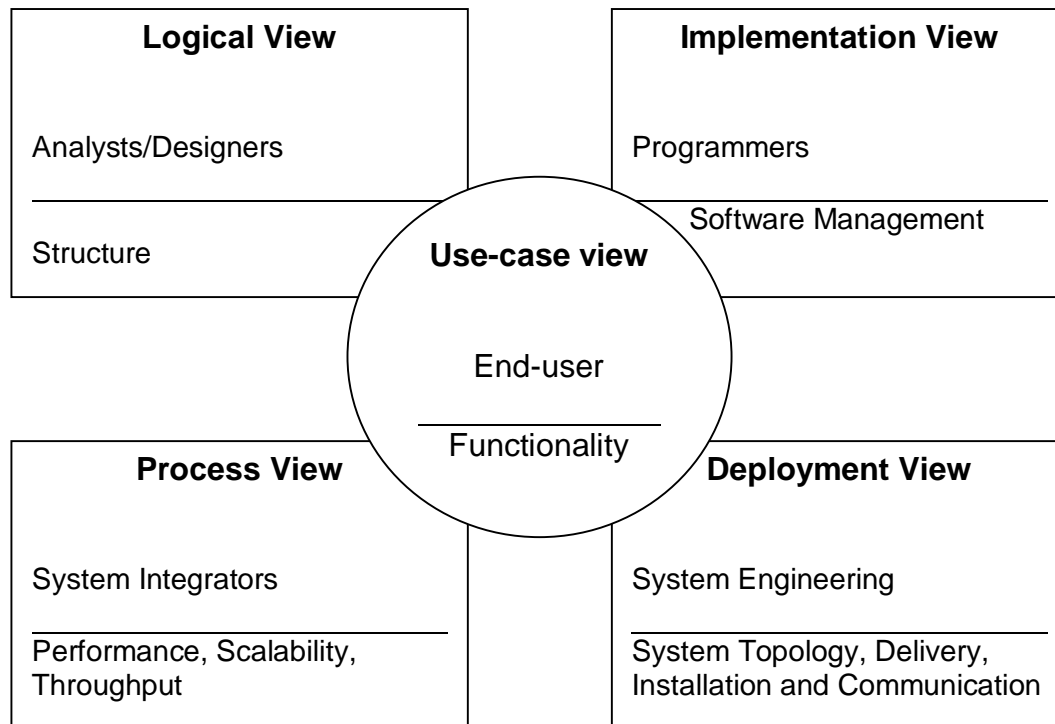


Figure 2.2-6 The architecture view model

Logical View

The logical view handles the functional requirements of the system (what the system should do for its users). This view is an abstraction of the design model and identifies major design packages, subsystems and classes.

Implementation View

The implementation view describes how static software modules (source code, data files, components, executables etc.) are organized in the development environment. It describes packaging, layering and configuration management of these modules.

Process View

The process view describes the system run time. This view looks at tasks, threads, and process as well as how they interact. It addresses issues such as concurrency, deadlocks, parallelism, fault tolerance etc

Deployment View

2.2.3.4 The deployment view looks at how executables and other runtime components are mapped to the underlying platforms. It addresses deployment, installation, performance etc.

Use-Case View

This view describe scenarios and use-cases for the system, and help the task of discovering and designing the architecture as well as help in validation of the other views.

Discussion

The different views are necessary in order for workers in each domain to model and communicate their information. Applying as much as five different views may be slightly extensive in small projects, but as the scope and size of most software projects tend to be large, having a software development process with several views may be necessary. Although the use of different views gives a clear advantage in the ability to express and communicate information, it does raise some issues. For instance, keeping information, requirements and models consistent between views may be difficult. Extra care should therefore be used in order to avoid such pitfalls.

2.2.3.5 Architecture related artifacts

RUP defines two primary artifacts

- SAD (The Software Architecture Description), describes the architectural views related to the project.
- The architectural prototype, for validation of the architecture. Also serves as a baseline for the rest of the project.

2.2.3.6 Architecture and reuse

If we want to gain large-scale reuse, architecture is of major importance. The architecture will clearly articulate the major components as well as the interfaces between them. This is a necessary feature for obtaining reuse.

Architecture assist both internal reuse – the identification of common parts – and the incorporation of COTS (Commercial Off-the-Shelf) components. Architecture also facilitates reuse on a larger scale; the reuse of the architecture itself.

2.2.3.7 Component-based development

RUP support component-based development (CBD), which is the creation of systems by assembling components, as well as the development and acquirement of such components.

“A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces”.

2.2.4 The Project Management Workflow

Software project management consists of:

- Balancing of competing objectives
- Risk management
- Overcome constraints
- Ensure a product that meets customer needs

Projects are seldom 100% successful, and project management is a difficult task. RUP offers a project management workflow to help improve the process. The project management workflow has three purposes:

- Give a framework for managing software intensive projects
- Provide guidelines for planning, staffing, executing and monitoring projects
- Give a framework for managing risk

2.2.4.1 Planning an iterative project

For an iterative process, the development should be based on two types of plans:

- A coarse-grained plan: The Phase Plan

For each development project we must have a phase plan. This is a one or two page document created early in the inception phase, and contains:

- § Dates and objective of major milestones planned (end of each phase)
- § Dates and objective of minor milestones planned (end of each iteration)
- § Staffing profile: which resources are required over time

- A series of fine-grained plans: the Iteration Plans.

There is an iteration plan for each iteration. The iteration plans are built using traditional planning techniques and tools (Gantt charts and so on)

2.2.4.2 The Project Manager

The major worker in the project management workflow is the project manager. Figure 2.2-7 shows how Worker: Project Manager is related to artifacts, activities and workflows in RUP.

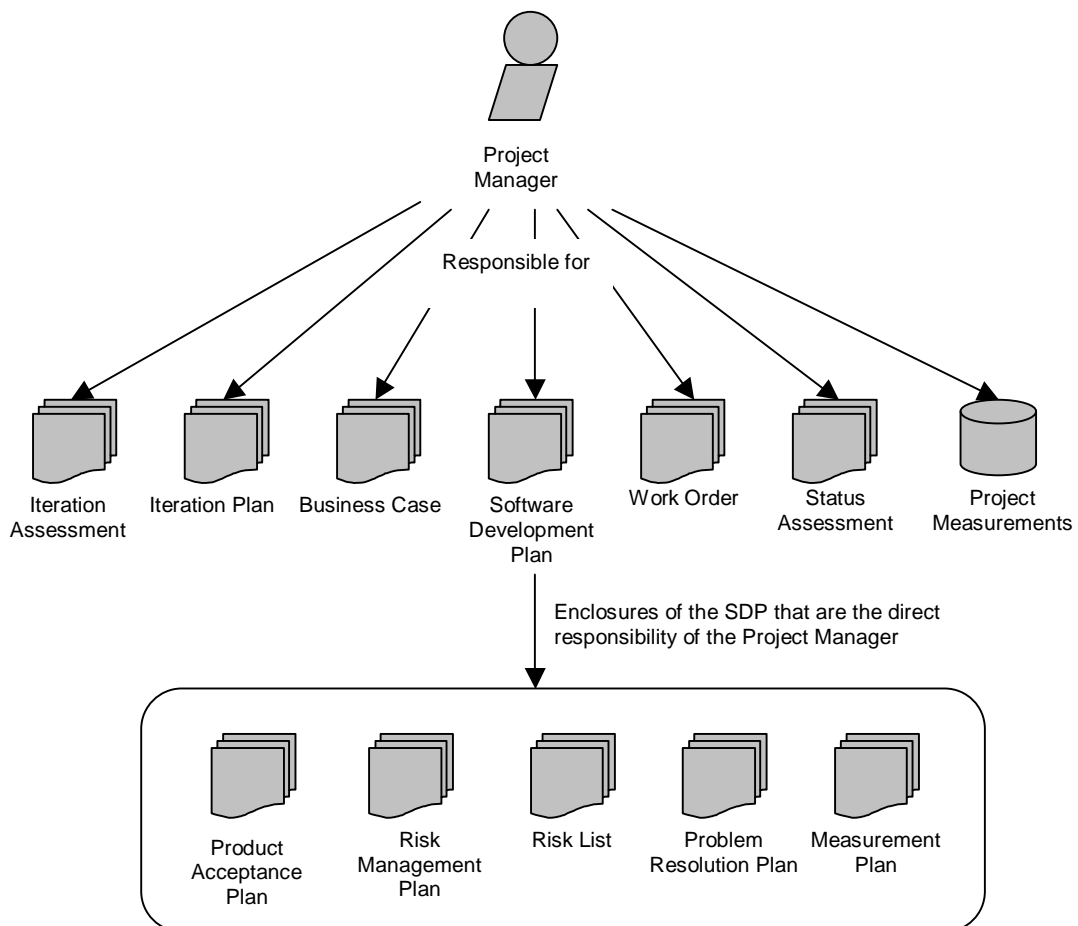


Figure 2.2-7 Worker and artifacts in the project management workflow

Figure 2.2-8 uses an UML activity diagram to illustrate one iteration in project management.

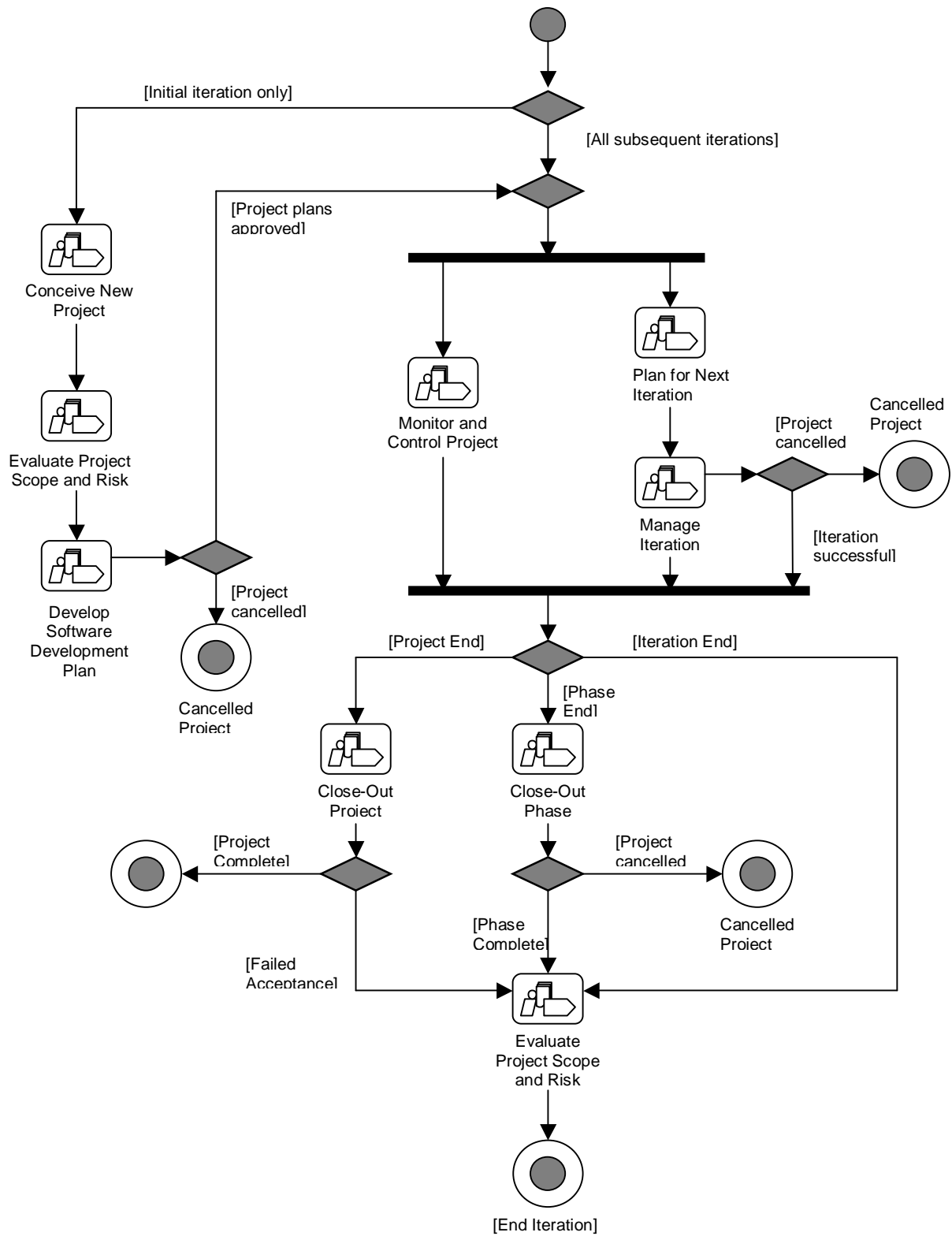


Figure 2.2-8 Workflow in project management

2.2.5 The Business Model Workflow

Business modeling is used to:

- Understand the structure and the dynamics of the customer organization
- Understand the current problems in the target organization and discover potential for improvement
- Ensure that common understanding of the target organization among the stakeholders, customers, end users and developers
- Derive system requirements

Figure 2.2-9 shows the workflow in business modeling. RUP offers the necessary tools to perform business modeling.

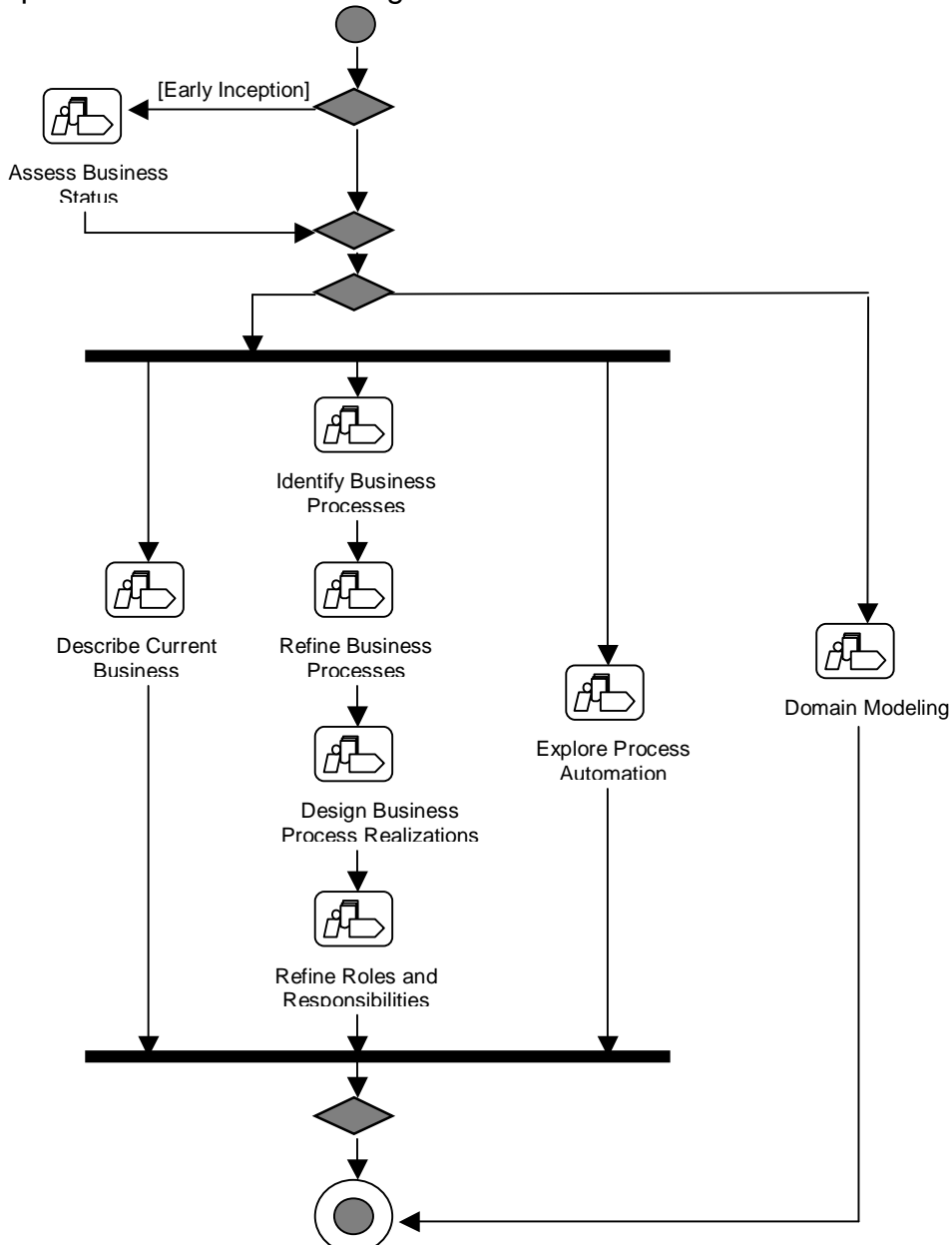


Figure 2.2-9 A workflow in business modeling

2.2.6 The Requirements Workflow

When developing a new system, it is important that developers, customers and other stakeholders establish and maintain agreement on what the system should do, why and how it should be done. These requirements will provide the basis for planning iteration contents, system boundary, cost estimates and so on. RUP provides a requirements workflow for handling requirements.

In software development we separate between two types of requirements; functional requirements, which are used to express the behavior of the system (what a system should do), and non-functional requirements, which specify a variety of attributes that are related to non-functional issues such as usability, reliability, performance and supportability.

2.2.6.1 Capturing and managing requirements

The first step in capturing requirements is to collect the stakeholders' requests. This activity consists of collecting the requests and wish lists from customers, end users, marketing and other stakeholders. The set of all these requirements are used to create a vision document.

The Vision Document consists of:

- Set of stakeholders
- Set of user needs
- System high-level features

The high-level features of this document have to be translated into detailed software requirements that enable you to design and build the actual system. These detailed requirements are captured in the use-case model and other supplementary specifications. Detailed requirements are then realized in a design model and end-user documentation, and are verified by a test model.

Throughout the system development you will undoubtedly find flows and inconsistencies in your requirements. This forces you to repeat the system requirements workflow in an iterative manner.

Rational support tools that capture, models and manage requirements, attributes and traceability.

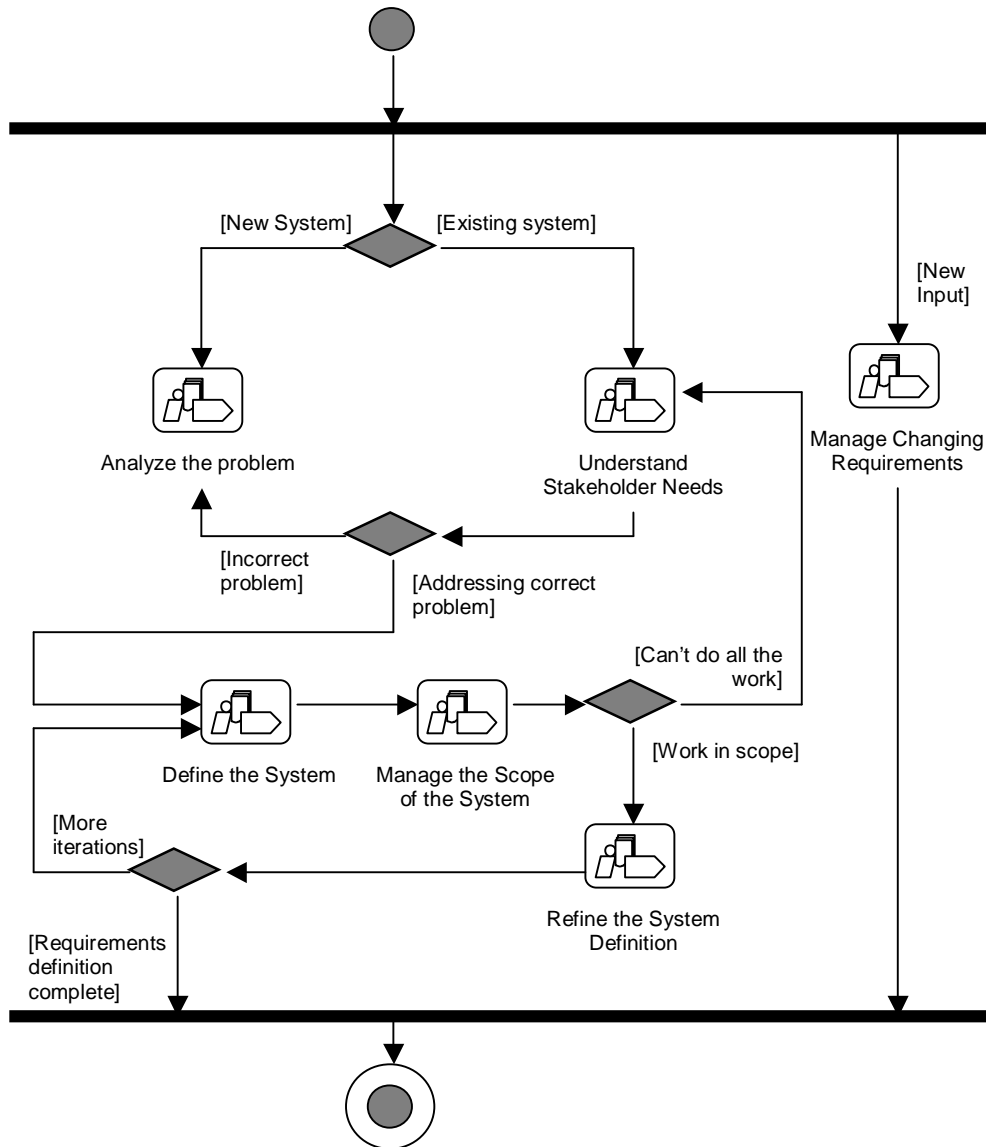


Figure 2.2-10 Workflow in requirements

2.2.7 The Analysis and Design Workflow

In order to translate requirements into a specification that describes how to implement the system, RUP provides the analysis and design workflow.

The main purpose of analysis is to map requirements into a form that is easier to handle when designing how to implement the system. This process is driven by use cases. Analysis focuses on ensuring that the system's functional requirements are handled. It is common to ignore many of the non-functional requirements until the design phase. During the design phase the main goal is to transform the result from the analysis work into a template on how to implement the system.

2.2.7.1 The design model

The design model is the primary artifact of the analysis and design workflow. It consists of the work of model elements providing the behavior of the system. The design model can be considered using architectural views. Using the logical view, the system is broken down in sub-pieces and presented as a set of logical elements (classes, subsystems, packages and collaborations). The process view maps these elements to processes and threads in the system. The deployment view maps the processes to a set of nodes on which they execute.

Figure 2.2-11 illustrates an iteration in the RUP workflow. Note that some activities are dependent on in which phase the iteration is performed.

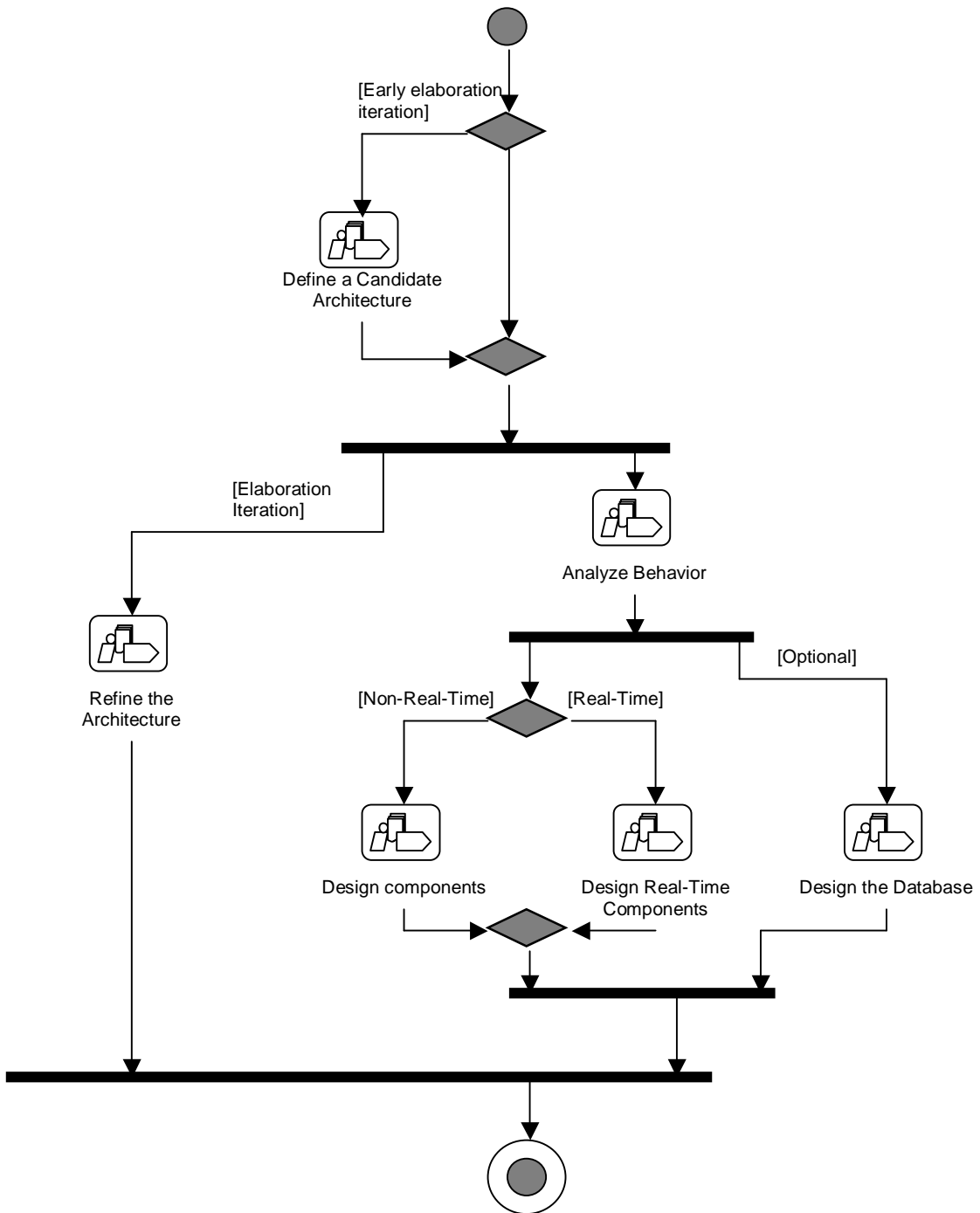


Figure 2.2-11 Workflow in analysis and design

2.2.8 The Implementation Workflow

During the implementation workflow we organize code in terms of subsystems and layers, implement classes and objects in terms of components and implement the executable system.

To understand the implementation workflow in RUP, we introduce three key concepts:

- Builds

At the end of each iteration we create one or more builds. This means implementing and executing a subset of the complete system and verifying that it works.

- Integration

Integration involves combining separate software components into a whole, as well as subsystems into a complete system.

- Prototype

A prototype may be exploratory (throwaway) which is thrown away after it is finished and you have learned what you wanted from it, or it may be evolutionary, in which case it evolves to become the final system.

There are two types of prototypes:

- § A behavioral prototype, which focuses on exploring specific behavior of the system. For example, showing a possible user interface to the customer. Behavioral prototypes are often exploratory.
- § A structural prototype, which explore architectural or technological concerns. These prototypes are often evolutionary, and are typically developed using the same tools as the main system.

Figure 2.2-12 shows an implementation workflow. During the elaboration phase, a lot of work is put into creating an implementation model. This model is organized in such a way, as to make component-based development as conflict-free as possible, help management and so on.

Implementation is closely related to design and there are clear tracing links from design to implementation elements.

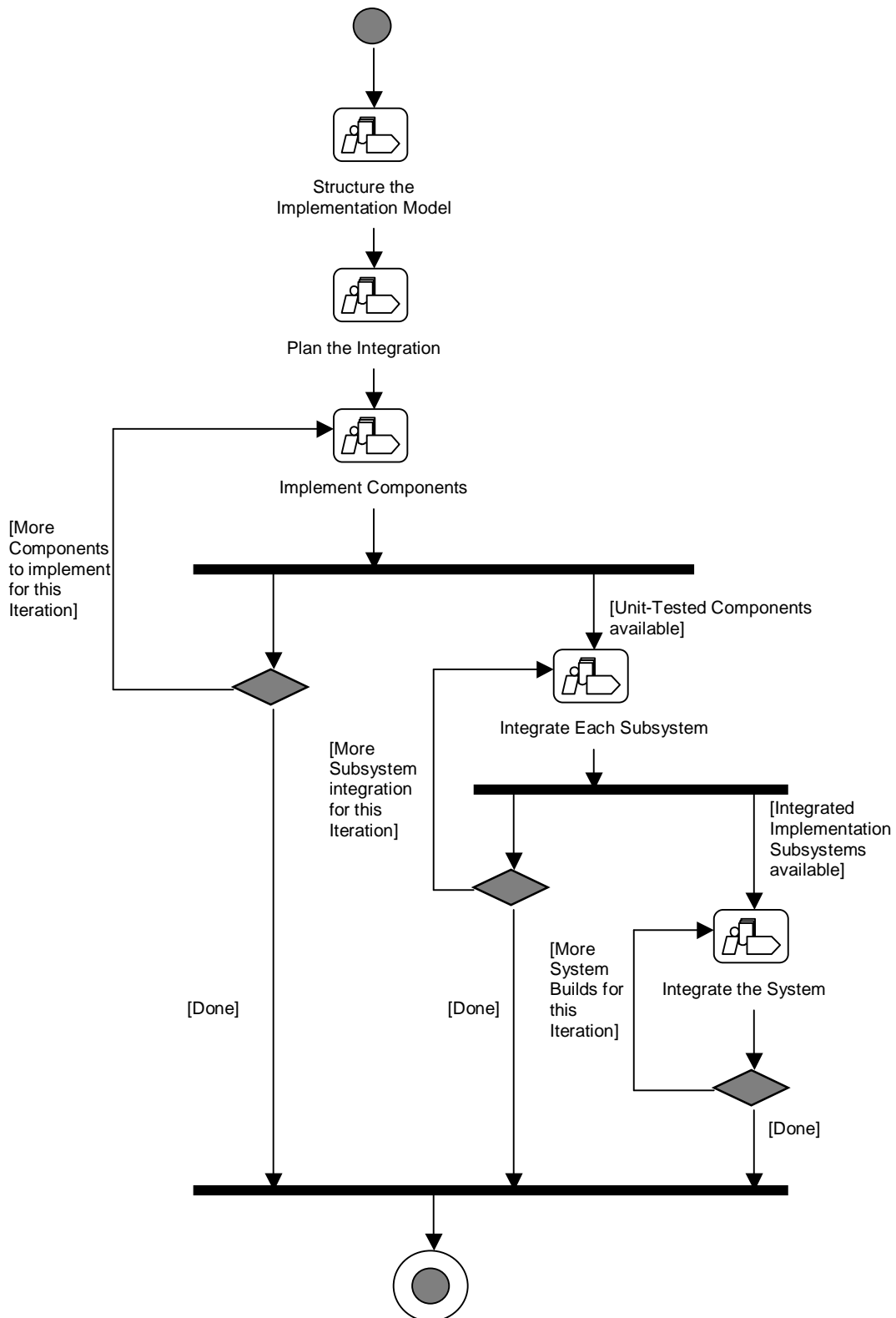


Figure 2.2-12 An implementation workflow

2.2.9 The Test Workflow

For software development to be successful, structured and thorough, testing should be applied to assess product quality. We need to verify that components interact correctly, that requirements have been met and that defects are addressed. Testing should occur in all iterations of a development to provide early feedback on product quality. RUP provides a testing workflow that gives feedback for the project.

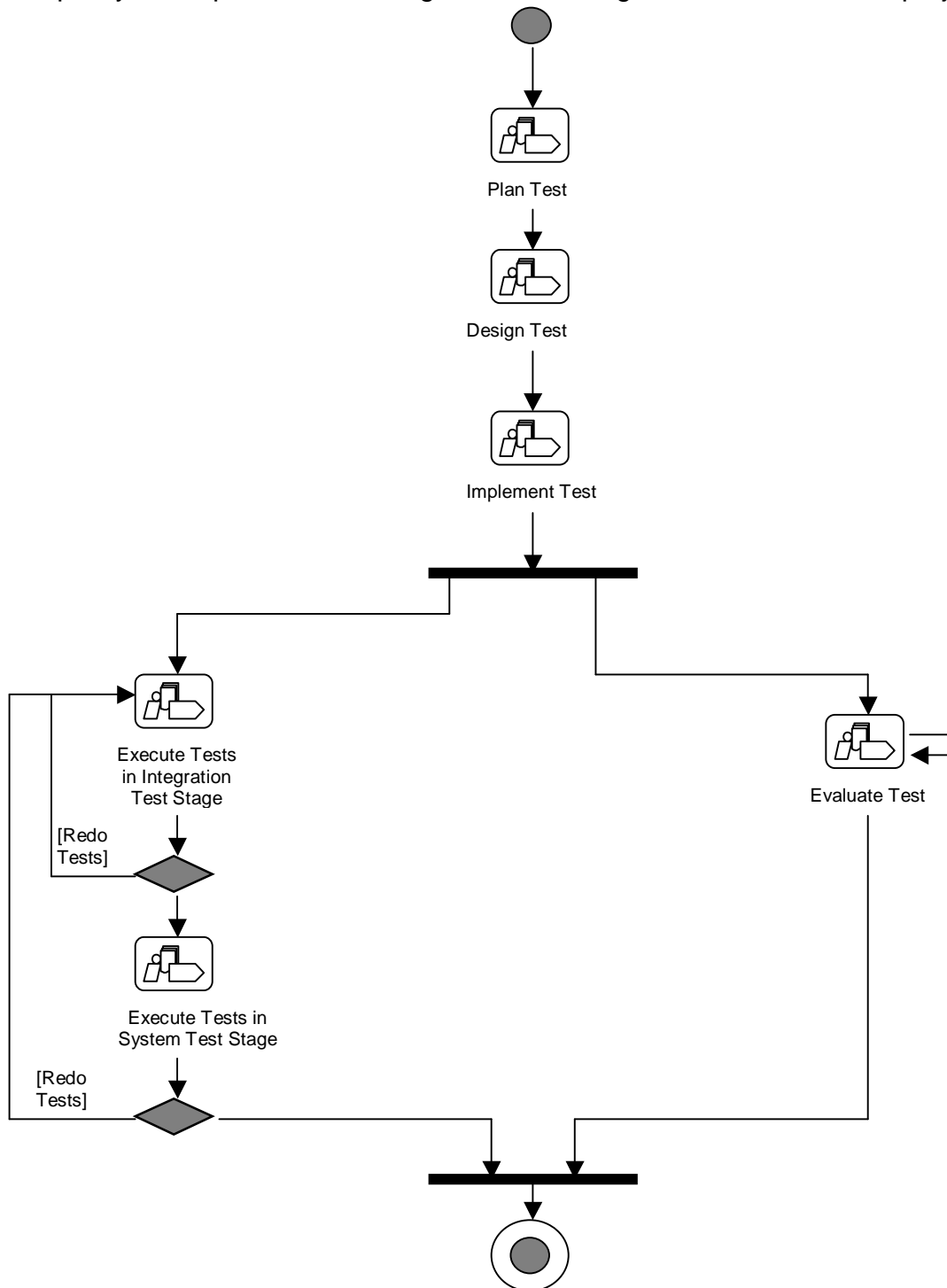


Figure 2.2-13 The test workflow

2.2.10 The Configuration and Change Management Workflow

During software development, several artifacts are created. These artifacts represent a major investment for a company and are labor intensive. When these artifacts evolve or are updated, it is important that this is handled in a consistent and structured manner. The configuration and change management workflow aims at maintaining the integrity of the project artifacts as they evolve.

- Configuration management (CM)

Handles artifact identification, versions and dependencies. The configuration management workflow also handles workspaces and validations of element configuration.

- Change Request Management (CRM)

Handles requested changes generated by internal and external stakeholders. It also handles analysis of the impact the requested changes will have on the development.

Figure 2.2-14 shows the configuration and change management workflow.

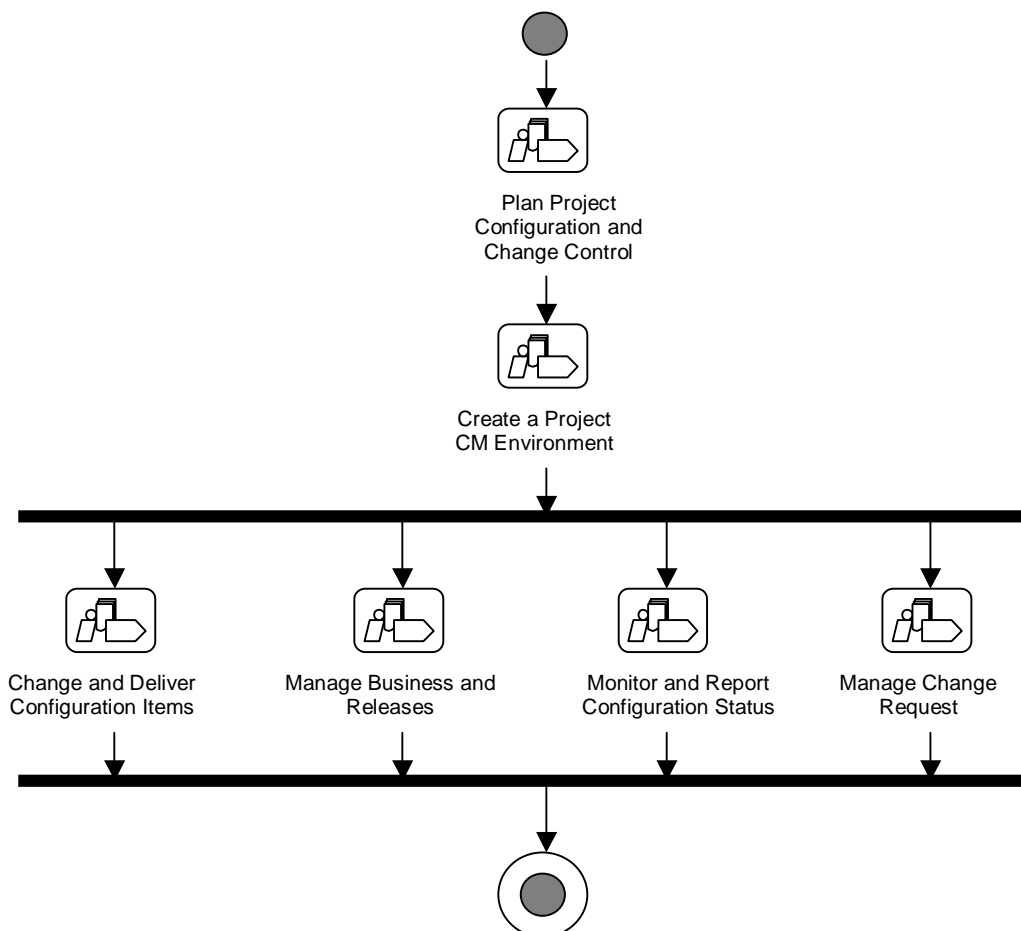


Figure 2.2-14 The configuration and change management workflow

2.2.11 The Environment Workflow

The environment workflow aims at providing processes and tools for the development organization.

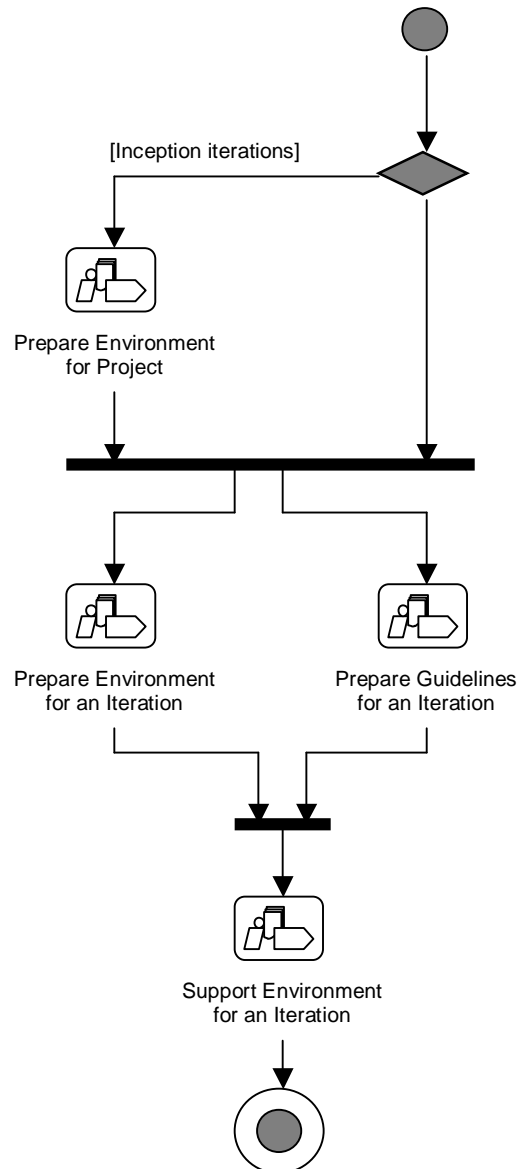


Figure 2.2-15 An environment workflow

The following support is provided through the workflow:

- Tool selection and acquisition
- Tool setup and configuration
- Processes configuration
- Process environment
- Process support and service

2.2.12 The Deployment Workflow

For a software development project to be successful it is not enough to have a working piece of software that fulfills the stakeholder's requirements. For such a project to be successful, we need for the customers to be willing to use the finished product.

RUP provides a software deployment workflow to handle this. The deployment workflow handles:

- Beta testing
- Packaging for delivery
- Installing the software
- Training end users
- Preparing sales force (product roll-out)
- Migrating existing software on converting databases

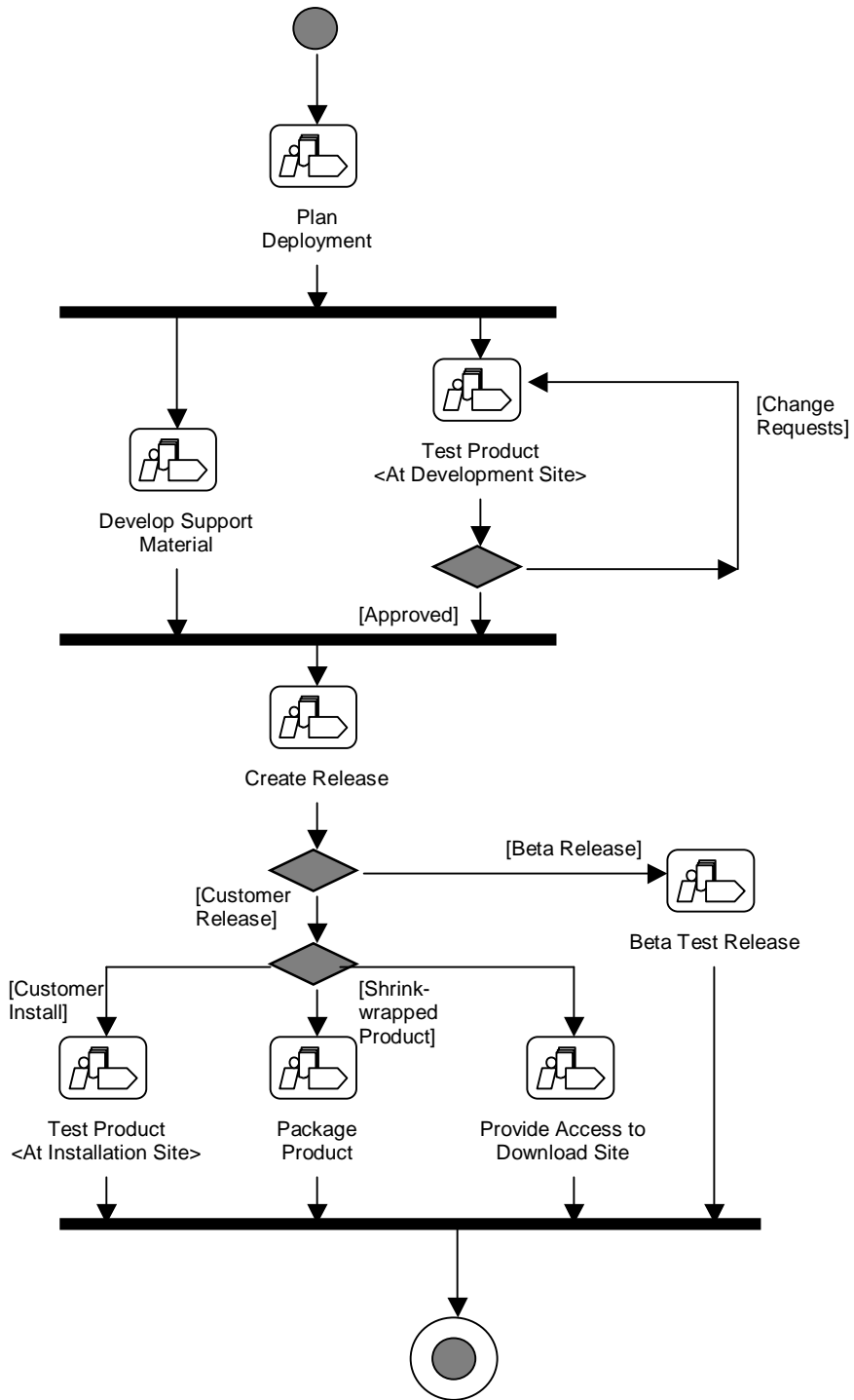


Figure 2.2-16 The deployment workflow

2.2.13 Iterations and Workflows

During each iteration several of the workflows may run, but which activities are emphasized on to what degree they are performed, depends on where in the software development the iteration is done. For example, an iteration in the inception phase will have a strong focus on the business modeling and requirement workflows, while the deployment workflow may not be run at all.

During an iteration in the constructing phase, all the workflows may run, but with a particular strong focus on the implementation, test and configuration/change request management workflows. Whether a particular activity in a workflow is performed may also depend on the needs/scope of the current project. For example not all projects will include the “provide access to download site” activity as prescribed in the deployment workflow.

2.2.14 Adapting the Rational Unified Process into a Software Development Organization

Adapting RUP into a software development organization will typically include a combination of changing RUP to better fit the organization needs (Configuration), and changing the organization to fit RUP (implementation). This process will result in a development case that constitute an enumeration of the different changes to be made to the complete RUP

The following areas must be addressed when implementing RUP in an organization:

- The workers and their competence, skills, motivation and attitude
- The support tools
- The software development process
- The organization

Special care must be taken, such that all these issues are handled correctly, if the process is to be successful.

Discussion

Given that an organization has decided to apply the rational unified process as their software development strategy. What special preparations must be made for a successful transfer to the new process, and is changing to RUP really worthwhile? The extra effort needed in order to motivate and prepare the workers for the new process may be extensive. There also have to be made investment in new tools and resources as well as organizational changes. There have to be done an extensive research work as to how and to what degree RUP should be adapted/ changed to suit the company needs. All these effort and investments combined may prove to be very large, but compared to the possible extra cost of having an unsuccessful project; it could often be economically intelligent for companies to make this investment, and change to the RUP development process.

3 Context

3.1 Reuse and Component-Based Development

Software development has several characteristics. Most importantly, creating the initial product is costly while creating and distributing copies is almost costless. This in turn results in a market where the importance of finishing a product before your competitors is extremely important (if you are not the first to finish, your competitors will steal your market).

Since software development products are so easily and quickly distributed, the importance of marketing your product, as well as ensuring excellent quality and support for the product is of minute importance. The purpose of this thesis includes finding ways to ensure high quality, stability and shorter time to market for software products developed at Ericsson AS. This we hope to ensure by improving one very important part of their software development process; 'reuse'.

Ericsson AS uses a tailored, or adapted version of RUP, called GSN RUP. Detailed description of GSN RUP is found in Appendix **Error! Reference source not found.**

The co-relevance between reuse and achieving high quality, stability, cost reductions and shorter time to market is well documented. Any software developer that is able to handle reuse will have a great advantage.

It is a fact that the developers at Ericsson AS are good at reusing components. Throughout the next chapters we hope to show that Ericsson AS can improve their software reuse even further, and suggest guidelines for achieving this.

3.2 Criteria for evaluating components in the GSN RUP adaptation

In this section we explore component selection and evaluation. It will start by giving a general discussion on requirements and component selection/evaluation, and will continue by giving some discussion on how this could be adapted in GSN RUP.

Component selection involves finding the components that best fit our requirements. The components selected must fit the overall architecture, be modifiable if changes are necessary, and be able to interact and perform satisfactory with the other components of the project.

3.2.1 General guidelines for software requirements

[3] Capturing a complete set of requirements for a software development project is getting increasingly difficult. Stakeholders may be uncertain about what they require the system to do, often maintaining an “I know it when I see it” attitude (IKIWISI). To make things worse, stakeholders may change their requirements mid project, even if they have been shown how the system will work through demos and prototypes. These problems force us to abandon the old way of system development (finding all requirements and then building a complete system that fulfill them). In modern software development we need a more dynamic way of handling requirements.

In order for our requirement strategy to fit any project situation it is unfeasible to use a “one-size-fits-all” approach. In [3], Barry Boehm suggest four guiding principles to help us handle IKIWISI, rapid change and internal/external- components. In short, Boehm reminds us to ensure that the requirements are value-, shared vision-, change-, and risk-driven.

Value-driven requirements: To determine the value-driven requirements it is necessary to perform a business-case analysis. The goal of this analysis is to see how different combinations of requirements add value to the product, and compares this with how much investment is needed to achieve the different combinations. It is also important to consider how the time aspects affect these values. It may be of major importance to get the product delivered to the market as early as possible. Arriving on the market with a product that only fulfill some of the requirements early (and the upgrading or provide patches) may be preferable to arriving later with a product satisfying the complete set of requirements. This is particularly true for the software development business.

“When working with requirements the main focus should usually be on the requirements that add value to the product (“value driven”).”

Shared-vision-driven requirements: In software development the developer has to deal with continuously changing market competition, constantly changing technology, and changing price and organizational realignments. Because of this changing/dynamic situation, it is not feasible for the stakeholders to discover and

agree upon all requirements early in the development process.

“It is, however, important that the stakeholders have a shared vision of the system’s goals and values”.

It must be agreed upon what value/goals the system will provide.

Change-driven-requirements: It is a known fact that the requirements of a project will change throughout the development process. Often, selecting the cheapest solution that satisfies a given set of requirement will provide a product that is very hard to modify if the requirements change. Ideally we should build a product that is easy to modify if requirements change. The best way of solving this problem is to invest some work in *“identifying the most likely directions of change in the requirement, and make the product dynamic to these changes”.*

Risk-driven requirements: The forth guideline provided by Boehm gives a general ‘best practice’ for deciding how much requirement specification detail to include early in development (how fine grained should the requirements specification be?). The answer to this is applying a risk-driven approach: *“If it’s risky to leave it out, put it in. If it’s risky to put it in, leave it out”.*

3.2.2 Requirements and component selection

In any component based development we should make our search for suitable components as broad as possible, and the process for discriminating and finding the best of these components should be as efficient as possible. The previous chapter we discussed some general guidelines for requirements in software development. In this chapter we will continue this discussion but shift our focus to how requirements are used and can be modified to suit the component selection process.

If we want to reuse previously created components in our software project, this involves discovering some requirements, and then trying to find the component that is best suited to satisfy these requirements. In this process we should take advantage of any option available to us. For instance, if our development team includes experts with extensive knowledge about components available in the problem domain, there is no reason why we should not apply this knowledge to generate requirements that are more likely to fit with these components.

[2] One topic in this area is deciding how much detail one should elaborate for the various requirements. One factor that helps shape the answer to this question is the observation that some requirements are more important for discriminating between components than others. Taking this into account, we can derive the conclusion that we should acquire more detail on these requirements in the early stages of the component selection process. This will enable us to discriminate, and reduce the number of components early, and this will in turn help us save time and resources.

For the requirements to be as useful as possible in component selection, they should have certain properties. They should be measurable in a way that helps discriminate between components and they should be structured so that test case formulation can

be performed without problems. Often requirements tend to be rather fuzzy, and it may be difficult to decide whether one component fulfill the requirements better than another component. Also, the hierarchical structure of requirements is often incompatible with the sequential structure of test cases (50 test cases for evaluating 50 requirement configurations etc, may be unfeasible). These problems should be kept in mind during requirement exploration and defining, and if possible, the requirements should be shaped in a fashion that avoid these problems.

When evaluating a component, a lot of different issues must be taken into consideration. Will the component be used for a long period of time? Does the component meet the requirements? Does the component need to be modified? Does the component integrate correctly with other components? And so on. Often the requirements or scope of the project may change and may force us to reevaluate the components selection process, as well as possibly force us to make new assumptions about the problem domain.

To make sure that we can obtain more detailed information about the problem domain in such cases, it may be useful to have a stakeholder representative with good understanding of the problem domain available throughout each evaluation.

The component selection process is all about evaluation components and comparing them. This evaluation process should be as objective as possible, and there should consequently exist standardized techniques for recording information during component evaluation.

If the evaluation process becomes subjective, we run a greater risk at not selecting the best component, and it is easier to select a specific component by being manipulated supplier sales techniques etc.

Figure 3.2-1 shows a simplified model for the component selection process as outlined in [2], and show the relationship between requirements and component selection.

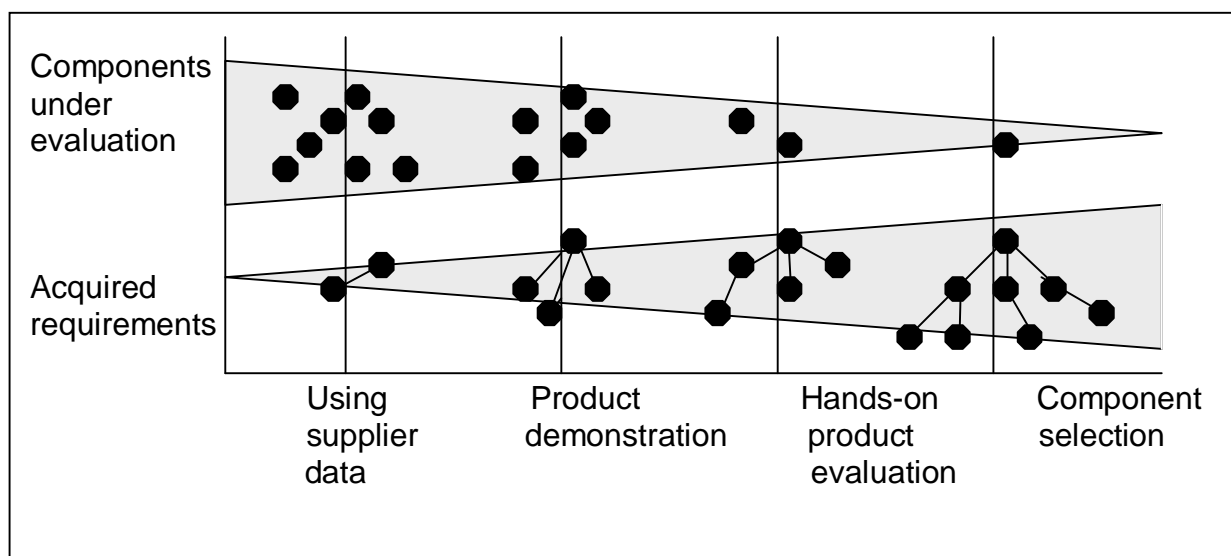


Figure 3.2-1 The relationship between requirements and component selection

3.3 Reuse Related Adaptations to GSN RUP

Following the discussion on requirements and component selection, this section will contain a proposal for changes that should be made to the GSN project's adaptation of RUP, GSN RUP (a detailed description of GSN RUP is found in appendix **Error! Reference source not found.**).

The current version of GSN RUP has some major shortcomings when concerning reuse of components. In brief it can be said that it is missing guidelines/activities for identification, evaluation, comparison and of components. It also has some shortcomings concerning setting criteria and general reuse strategy. To address these shortcomings, we propose some extra activities.

The below table show the Ericsson RUP phases, iterations and decision points in the left column, and the a brief summary of these phases as well as proposed reuse activities in the right column. The table is followed by a more thorough explanation of these activities, and a more precise description on where, and by whom they should be performed in GSN RUP. Tollgates (or TG in short) are Ericsson's defined milestones. A description of tollgates and milestones is found in appendix **Error! Reference source not found.**

<p>TG0</p>	<p><i>Purpose of tollgate:</i> This tollgate is performed prior to the first iteration. It serves the purpose of deciding whether or not to initiate the project.</p> <p><i>Proposed Reuse Activities:</i> none.</p>
<p>1.</p> <p>Pre-Study/ Inception Iteration</p> <p>+</p> <p>Milestone</p> <p>+</p> <p>TG1</p>	<p><i>Purpose of the first phase:</i> Establish the software scope and boundary of the project. Discover the initial use-cases (primary scenarios of behavior). Establish overall cost & schedule for the entire project, and a detailed estimate of the elaboration phase. Estimate risks.</p> <p><i>Proposed Reuse Activities:</i></p> <p>1.1. Plan reuse strategy and criteria for the evaluation strategy. <i>Decision Point 1:</i> Make vs. buy decision: (are we willing to depend on an outside vendor? can we renegotiate the requirements? If no the components have to be made from scratch, and we need no reuse strategy).</p> <p>1.2. Domain analysis (analyze who may reuse the components we make in the future (<i>for reuse</i>))</p>

<p>2.</p> <p>Feasibility Elaboration</p> <p>Iteration + Milestone + TG2</p>	<p><i>Purpose of the second phase:</i> Analyze the problem domain, Define, validate and baseline the architecture. Develop project plan. Eliminate high risk elements.</p> <p><i>Proposed Reuse Activities:</i></p> <p>2.1. Add the activities leading to the second buy vs. make decision</p> <p>2.1.1. Component identification and selection</p> <p>2.1.2. Component familiarization</p> <p>2.1.3. Feasibility study and Make vs. Reuse or Buy decision. <i>Decision Point 2:</i> Make vs. Reuse or Buy decision</p> <p>2.1.4. Renegotiation of requirements</p>
<p>3.</p> <p>Execution / Construction Iterations + Milestone(s) + TG3,4</p>	<p><i>Purpose of the third phase:</i> Building the product and evolving the vision, the architecture and the plans until the project is completed. Achieve adequate quality within time limits</p> <p><i>Proposed Reuse Activities:</i> (In each iteration):</p> <p>3.1. Possibly run second make vs. reuse or buy process in each iteration.</p> <p>Use reuse check list in milestones/TG.</p>
<p>4.</p> <p>Execution/ Transition Iteration + Milestone + TG5</p>	<p><i>Purpose of the fourth phase:</i> Provide user support. Train user community. Market, distribute/sell product. Achieve user self support.</p> <p><i>Proposed Reuse Activities:</i></p> <p>4.1. Update reuse related documentation</p> <p>4.2. Update repository</p> <p>4.3. (Check list)</p>
<p>5.</p> <p>Conclusion Iteration</p>	<p><i>Purpose of the final phase:</i> define and store experience from the current software development project.</p> <p><i>Proposed Reuse Activities:</i></p> <p>5.1. Conclude documentation</p> <p>5.2. Record reuse experiences</p>

3.3.1 Phase 1: Pre-study / Inception

Purpose: Establish the software scope and boundary of the project. Discover the initial use-cases (primary scenarios of behavior)
Establish overall cost & schedule for the entire project, and a detailed estimate of the elaboration phase. Estimate risks.

- **Activity 1.1: Plan evaluation criteria and reuse strategy**

Explanation: The organization should have a standard reuse and evaluation strategy that is applied on all software projects. Since each project is unique, an activity for tailoring this strategy to the particular project should exist.

Purpose: Tailor the existing evaluation strategy to fit the current project.

Steps: Evaluate project reuse needs, update standard reuse and evaluation strategy documentation to fit the current project, define evaluation criteria, make first make vs. buy decision

Input Artifact: Standard reuse strategy document, standard evaluation strategy document.

Resulting Artifacts: project specific reuse strategy document, project specific evaluation strategy document.

Worker: Project Manager

Workflow: Project Management (PM) workflow

First make vs. buy decision:

Answer the following questions:

Are we willing to depend on an outside vendor?

Can we renegotiate the requirements?

If 'no' to both questions, all components have to be made from scratch and we do not need a reuse strategy.

- **Activity 1.2: Perform domain analysis**

Explanation: If we want to create components that are feasible for reuse in the future, we need to make an analysis of whom possible reusers may be.

Purpose: Analyze future reuse domains to find possible reusers.

Steps: Study previously developed projects. Gather information from domain experts. Examine emerging technologies. Study possible reusers and reuse domains for the components.

Input Artifacts: Final reports from previous projects. Documentation on emerging technologies. etc. etc.

Resulting Artifacts: Domain analysis report

Worker: Domain Analyst (new)

Workflow: A&D

If there exist no possible future reusers, the components created does not need be reusable.

- **Additions to the Inception Phase Milestone:**
Checklist: The Inception checklist should be extended to include the following (see appendix **Error! Reference source not found.** for full checklist):

	Project Status	Comment
14	Have a decision been made to whether the project will reuse components (based on the first Make vs. Buy decision)?	Project Manager, PM workflow
15	Have the standard evaluation strategy and reuse strategy been tailored to suit this specific project?	Project Manager
16	Is the domain analysis report sufficiently developed?	System Analyst
17	Have a decision been made to whether new components created should be created to be reusable (based on the domain analysis)?	System Analyst and Architect

3.3.2 Phase 2: Feasibility/ Elaboration

Purpose: Analyze the problem domain, Define, validate and baseline the architecture. Develop project plan. Eliminate high-risk elements

- **Activity 2.1: Second Make versus Reuse or Buy decision**

Explanation: The software organization should as much as possible try to reuse components rather than creating new components in order to minimize costs and component maintenance.

Purpose: Find components that can be reused in the system under development. This is a top-level activity consisting of the low-level activities “component identification and selection”, “component familiarization”, feasibility study”, “renegotiation of requirements”, and “decide on component”.

Steps: Identify and select suitable components, get familiar with components, test components, select component.

Input Artifacts: reusable components, component documentation, analysis model, design model.

Resulting Artifacts: selected components, updated design and analysis model, updated documentation.

Worker: architect

Workflow: analysis & design

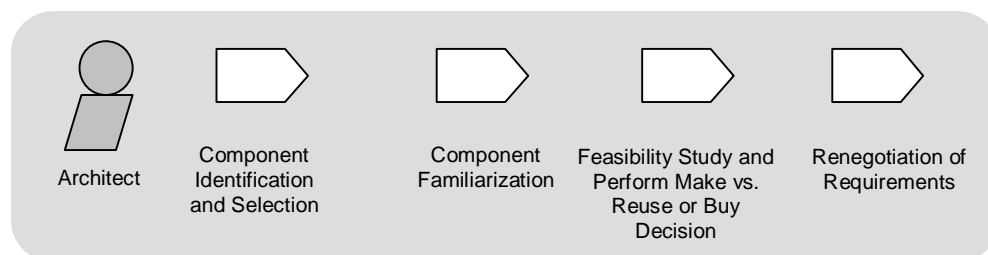


Figure 3.3-1 The second make vs. reuse or buy decision activity

- **Activity 2.1.1: Component Identification and Selection**

Explanation: Discover components suitable for the project. At this stage in development, requirements are general, and the components under evaluation are high-level components (typically large components with major impact on the software architecture (OS, ... etc.)).

Purpose: Find suitable components

Steps: Gather requirements, search internal repository for components, search external repositories for components, select some candidates for further evaluation (5-30).

Input Artifacts: ARS, Change Requests

Resulting Artifacts: candidate reusable components

Worker: architect

Workflow: analysis and design

- **Activity 2.1.2: Component familiarization**

Explanation: Get to know components selected

Purpose: To get familiar with and to narrow down number of candidate components

Steps: Study component documentation, gather information from experts familiar with the component, test component functionality, test non-functional attributes, study component impact on architecture, study component integration cost. Create a FIS for each candidate component; select the 1-3 best suitable components for further evaluation.

Input Artifacts: ARS, candidate reusable components, component documentation.

Resulting Artifacts: candidate reusable components, FIS for each candidate.

Worker: designer

Workflow: analysis and design

- **Activity 2.1.3: Feasibility study and perform make versus reuse or buy decision**

Explanation: The candidate components resulting from the component familiarization activity (1-3 components) as well as the option of creating the component from scratch should be scrutinized, tested and compared in order to find the best suited component.

Purpose: Find the best-suited component for a specific task.

Steps: For each component: create a high level architecture, do an effort estimation, do a cost estimation, create a risk assessment model, test dependencies between components, compare components, select the best suited component.

Input Artifacts: ARS, candidate reusable components, component documentation.

Resulting Artifacts: One suitable component, Change Request

Worker: designer

Workflow: analysis and design

- **Activity 2.1.4: Renegotiation of Requirements**

Explanation: In every of the previous activities one may discover that the components under evaluation do not completely fit the requirements from the ARS. It may then be useful to renegotiate and change the ARS.

Purpose: Renegotiate requirements to better suit available components.

Steps: Renegotiate requirements with stakeholders

Input Artifacts: ARS, Change Request, candidate reusable components.

Resulting Artifacts: Updated ARS.

Worker: project manager

Workflow: requirement

As shown in Figure 3.3-2, a high level architecture should exist prior to executing the 'Make versus reuse decision 2' activity. The high level architecture is needed in order to understand which components are suitable, how they affect the overall system under development and so on. The architecture may also be updated/changed during the 'Feasibility Study and Perform Make vs. Reuse or Buy Decision' sub activity.

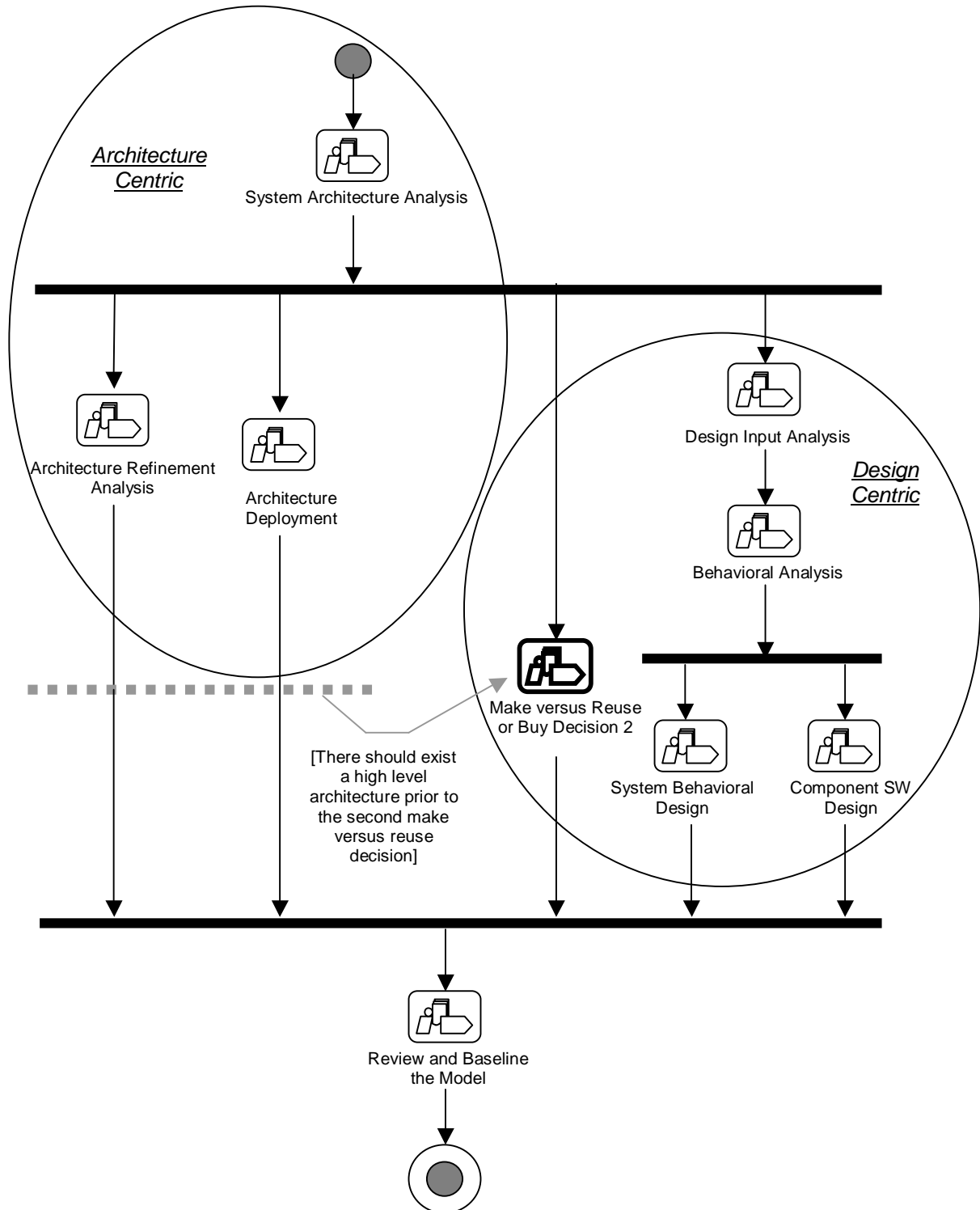


Figure 3.3-2 The proposed Analysis and Design Workflow

The 'Second Make versus reuse or buy decision' activity is closely linked to the other activities in the Analysis and Design workflow.

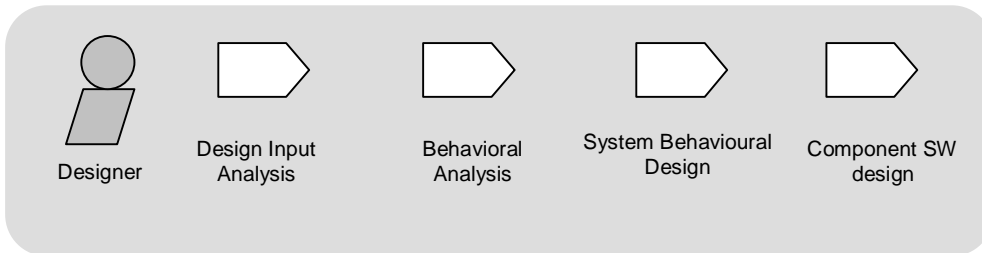


Figure 3.3-3 Some other activities in the Analysis and Design Workflow

For instance, the activity 'System Behavioral Design' (Figure 3.3-4) contains many of the same sub activities needed for the feasibility study for reusable components ('Design Component Interaction' and 'Define Component interfaces'). The sub activity 'Define Component Interfaces' in the 'System Behavioral Design' activity is also a sub activity to the 'Component Identification and Selection' activity.

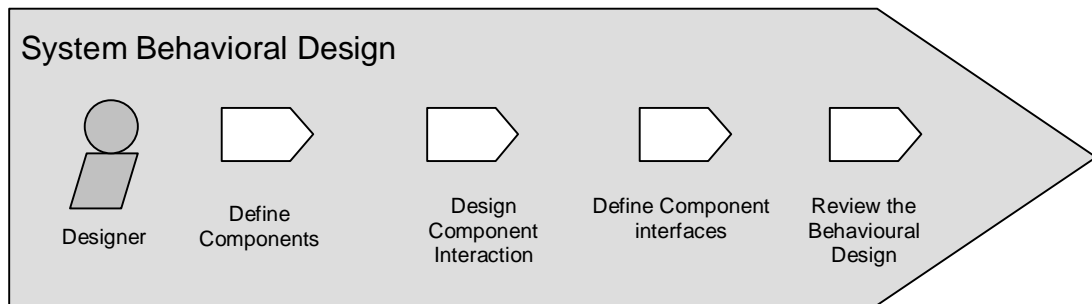


Figure 3.3-4 The System Behavioral Design Workflow

In the proposed Analysis and design workflow Figure 3.3-2, the 'Second Make versus Reuse or Buy' activity is drawn as a parallel sub flow to the architecture and design centric flows, but the flow will affect and interact with most of the other activities in these flows. This new sub flow is design centric.

- **Additions to the Elaboration Phase Milestone:**

Checklist: The Elaboration checklist should be extended to include the following (see appendix **Error! Reference source not found.** for full checklist):

	Project Status	Comment
18	Has adequate work been performed with respect to finding and incorporating reusable components?	Designer, Architect
19	Have changes made to the ARS, or Change Requests written during component selection been introduced, and coordinated with the rest of the software organization?	PM

3.3.3 Phase 3: Execution / Construction Iterations 1,2,3

Purpose: Building the product and evolving the vision, the architecture and the plans until the project is completed. Achieve adequate quality within time limits.

- **Activity 3.1: Second Make versus Reuse or Buy decision (Revisited)**

Explanation: The make versus reuse process have to be repeated for each iteration. In the early iterations, when requirements are general and the system architecture flexible, the main focus will be on finding major/large reusable components (high level). In later iterations the reusable components will be smaller, and typically not have a great impact on the architecture (low-level). Low-level components may for instance be software classes, structures etc. It may not be necessary to perform each activity (component familiarization and so on) for such components.

Purpose: Find components that can be reused in the system under development. This is a top-level activity consisting of the low-level activities “component identification and selection”, “component familiarization”, feasibility study”, “renegotiation of requirements”, and “decide on component”.

Steps: Identify and select suitable components, get familiar with components, test components, select component.

Input Artifacts: reusable components, component documentation, analysis model, design model.

Resulting Artifacts: selected components, updated design and analysis model, updated documentation.

Worker: architect

Workflow: analysis & design

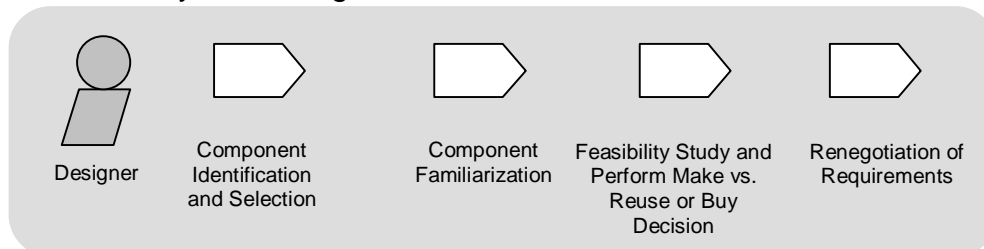


Figure 3.3-5 The second make vs. reuse or buy decision (revisited)

- **Additions to the Execution / Construction Phase Milestones:**

Checklist: The Execution / Construction phase milestone checklist should be extended to include the following (see appendix **Error! Reference source not found.** for full checklist):

	Project Status	Comment
	Has adequate work been performed with respect to finding and incorporating reusable components?	Designer, Architect
	Have changes made to the ARS and possibly Change Requests written during component selection been introduced and coordinated with the rest of the software organization?	PM

3.3.4 Phase 4: Execution / Transition Iteration

Purpose: Provide user support. Train user community,. Market, distribute/sell product. Achieve user self support (make users self-reliant).

- **Activity 4.1: Update Execution / Transition Iteration Reuse Related Documentation**

Explanation: If our choice to reuse components have any effect on execution / transition iteration related aspects, such as user support, product distribution etc, these experiences should be documented. For instance, if we depend on an outside vendor, we may discover advantages/disadvantages with respect to product maintenance and user support opposed to components created/supported internally. We may also for instance, find that the reusable components give fewer errors when deployed because they are better tested.

Purpose: Document reuse related effects.

Steps: Gather information related to reuse. Document this information. If there is any existing documentation, update this with the new information. Update the reuse repository with the new documentation.

Input Artifacts: old reuse related documentation.

Resulting Artifacts: an updated document related to reuse.

Worker: Project Manager

Workflow: Project Management

The 'Update Execution / Transition Iteration Reuse Related Documentation' activity is a sub activity of the 'manage iteration' activity in the execution / Transition phase of the PM workflow

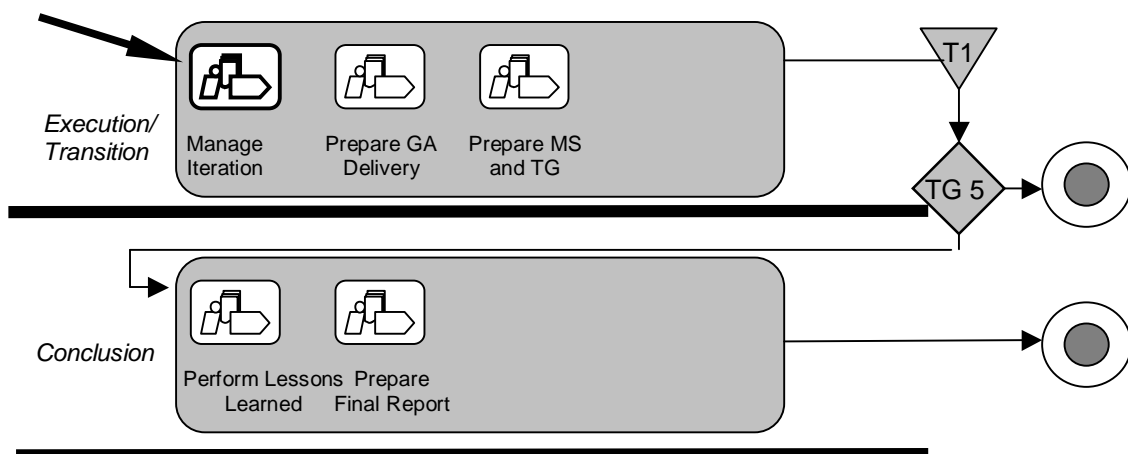


Figure 3.3-6 An extract of the Project Management Workflow

Phase 5 (new phase): Conclusion / Iteration

Purpose: Define and store experience from the current software development project.

- **Activity 5.1: Gather / Document reuse experiences**
Explanation: It may be useful to store experience related to reuse learned throughout the project. This information may be used to improve quality for later similar projects. The experience can be used to avoid pitfalls and to incorporate aspects known to be particularly successful.
- *Purpose:* Document reuse experiences
Steps: Gather reuse related experiences from subprojects, teams and individual project members. Analyze material and suggest improvements.
- *Input Artifacts:* every artifact related to reuse throughout the project.
- *Resulting Artifacts:* final report
- *Worker:* Project Manager
- *Workflow:* Project Management

The 'Gather / Document Reuse Experiences' activity is a sub activity of the 'Perform Lesson Learned' activity in the conclusion phase of the PM workflow

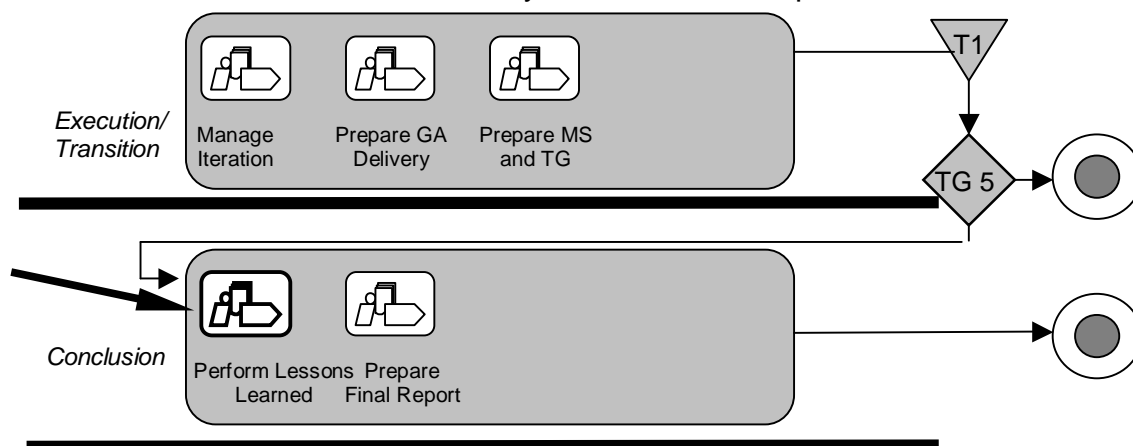


Figure 3.3-7 The Conclusion Phase of the PM Workflow

3.4 Support Changes in Software Development

In addition to the new proposed activities in GSN RUP, some changes have to be made to the existing documentation framework. This includes documentation for requirements and change requests. This is discussed in the Ericsson specific appendix **Error! Reference source not found.**

4 Research Approach

4.1 Work to be done

Phase 1, Familiarizing with RUP and GSN RUP

The first task will be familiarization with the Rational Unified Process. This familiarization should be thorough, since it forms the basis for our contribution to the thesis. Our next task will be an extensive study of the GSN adaptation of RUP, GSN RUP, which is the process we hope to improve. We will use documentation provided by Ericsson AS to get familiar with RUP.

Phase 2, Propose changes to the GSN RUP

Using the experience gathered from the pre-study project on reuse performed during fall 2001, as well as the understanding gained in phase 1, we will make an effort towards improving the reuse aspect of GSN RUP (by adding new activities, processes, workflows etc.).

Phase 3, Explore how the current process is viewed by the employees

Conducting a survey, information will be gathered from employees. This information will be used to discover how employees experience the current process.

A schedule of how we planned our work can be found at appendix B.2.

5 Survey of reuse aspects in software development

5.1 Hypotheses related to the survey

The existing development process at Ericsson does not sufficiently support reuse of components.

The process does not contain structured / organized routines for evaluation of components, comparison of components, and decision making concerning modification of components versus creating new ones.

Hypotheses:

- H₁: Reuse in Software Development gives significant advantages.
- H₂: It is not easy for a given design/code component to choose between: reuse "as-is", reuse "with modification", or make new from scratch. Even when reuse of components occurs, we have no way to ensure that the best candidate component has been selected.
- H₃: Better guidelines and workflows for development are needed. Criteria for compliance with existing architecture are not clearly defined.

5.2 A Survey of the software development process and reuse aspects at Ericsson AS

A small group of software developers at Ericsson were engaged to participate in a survey about their software development process and their reuse practice.

Nine people with different roles and experience in the company were selected. These people were members of the same team at Ericsson AS, and not randomly chosen (so keep in mind the results may be somewhat biased). However, the selection of a team for this survey were somewhat chosen in a random fashion.

The nine participants of the survey filled out a questionnaire prepared by the authors of this diploma.

The questions were associated with the hypotheses from the former chapter (5.1), and also divided into categories. The first category was Personal Info. Even though the survey was done anonymously, some information was still required (current role at Ericsson, experience, number of projects participated in, and so on). This information is used to gain better understanding of what the candidates base their answers on.

The next category was general reuse questions, formed as how important reuse is in relation to different assertions, and how important different technologies and tools are.

Then followed a category dealing with components and reuse, to see what the SW developers thought about the current component reuse situation.

Three questions were added to get some feedback on requirements, and the change of requirements during the course of a project.

A section was added concerning Application Frameworks. The Application Frameworks at Ericsson is something the SW developers should know well. The authors would like to see if that was the case, and what they thought about the framework documentation.

The GSN RUP workflows are helpful project tools. The authors wished to see how much these were used and what the participants thought about GSN RUP as an information guideline. Also, what impact can improving reuse activities have on the development? A section of question was added to get these answers.

The last category of the questionnaire was Architecture Compliance. The questions dealt with criteria for integrating components into an existing architecture, and non-functional requirements and properties.

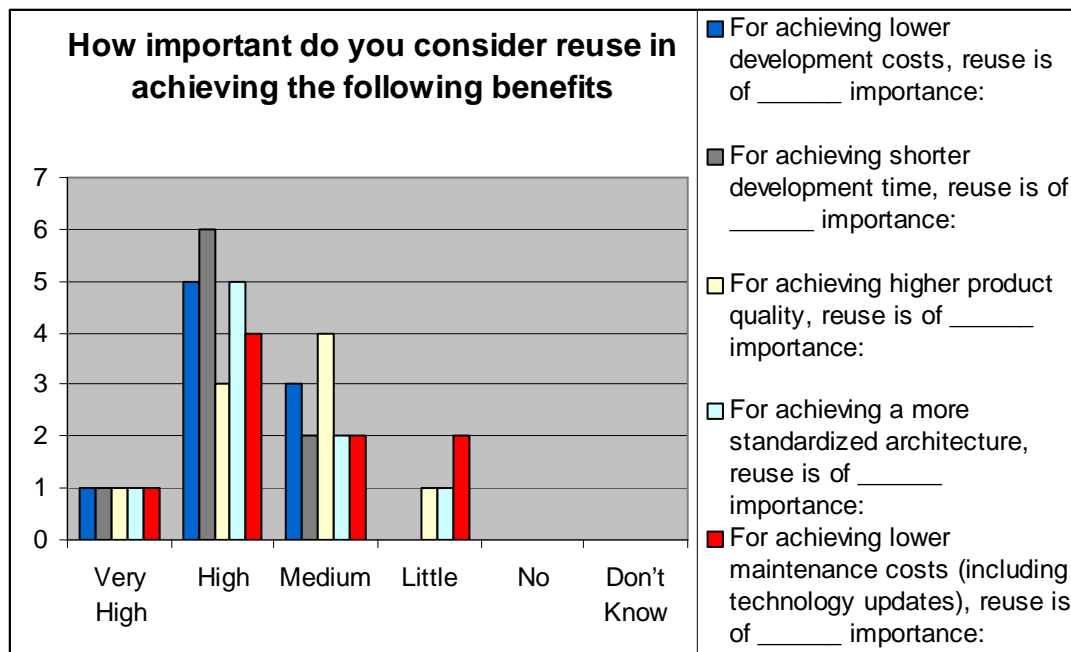
The questionnaire from the survey can be found in appendix B.1, and the results from the survey (questionnaire answers) can be found in appendix **Error! Reference source not found.**

6 Evaluation of results

This chapter contains the evaluations of the results from the survey found in appendix **Error! Reference source not found.**. The relatively small scale of this survey (9 candidates) means that some of the results may be inconclusive. The questionnaire did provide some useful, some interesting, some ambiguous, and some interesting answers. Some of the candidates felt that they were not sufficiently experienced to answer some of the questions; those questions were left blank. This explains why some of the answers do not add up to nine candidates in the following results.

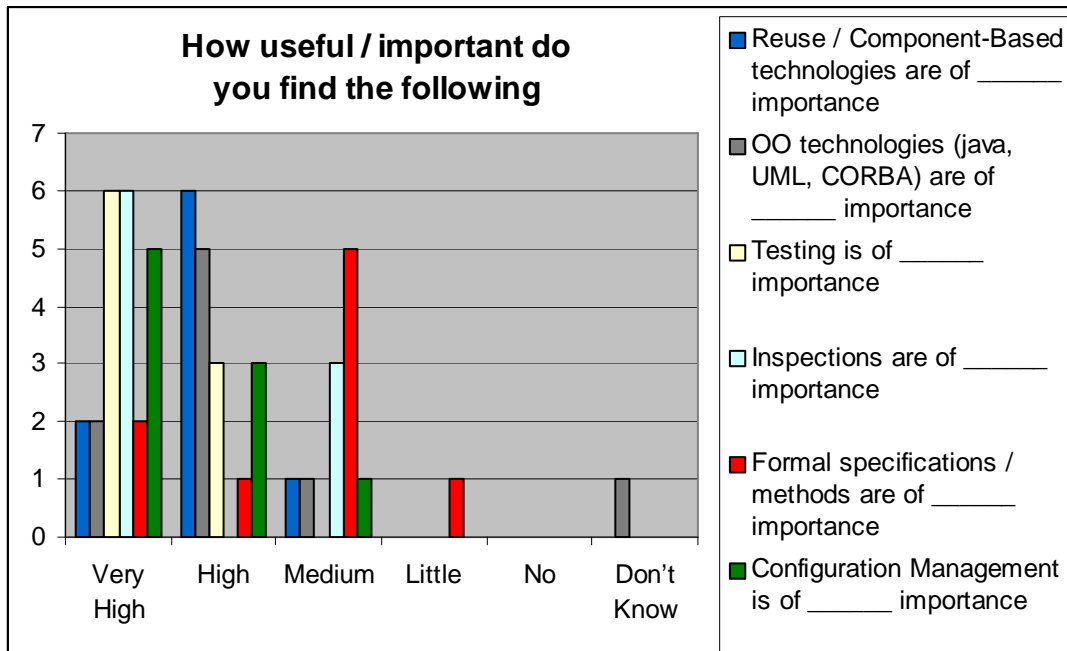
6.1 The General Questions

General Question G1:



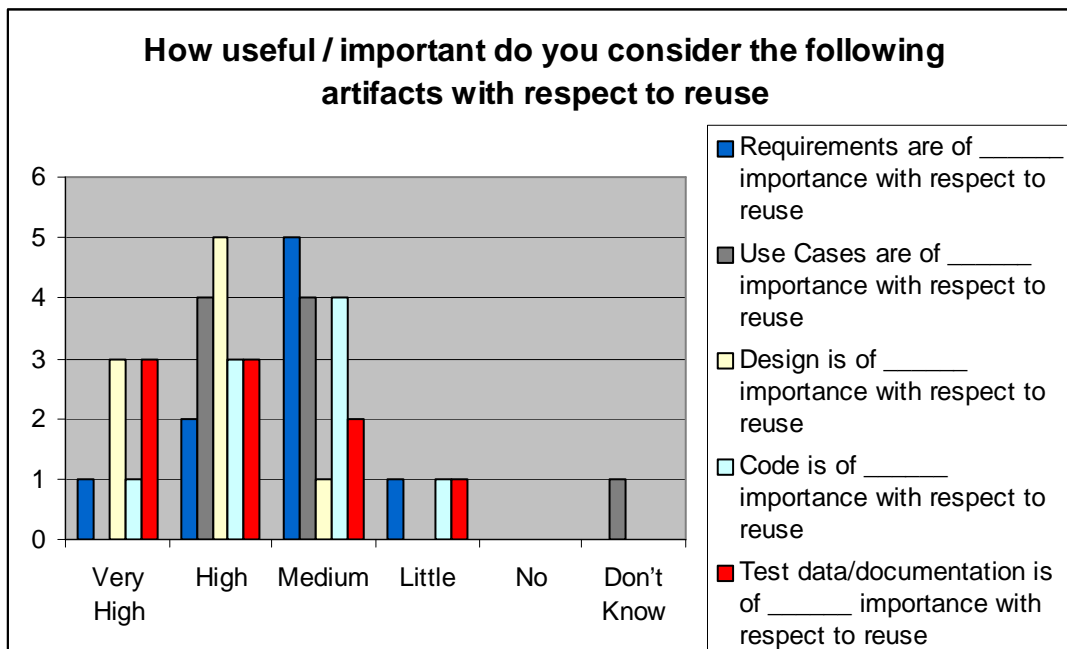
Comments: By the answers that was given here, we can see that the workers at Ericsson that participated in the survey think that the importance of various reuse benefits are typically in the range from “medium” to “very high”. Especially in the question about shorter development time, there seem to be an agreement about a high importance.

General Question G2:



Comments: The candidates seem to be in agreement that the technologies/tools mentioned in question G2 are of some importance. The graph shows that the answers are in the range from “medium” to “very high”, with the exception of one person who thinks that formal specifications / methods are of “little” importance and one person that did not know about the importance of OO technologies.

General Question G3:



Comments: Here, the trend is in the range from “medium” to “high” with respect to the importance of various artifacts. A few answers land on “little” and “very high” importance, and one person don’t know whether Use Cases are any importance with respect to reuse. Artifacts of “high” importance are typically Use Cases and Design, and artifacts of “medium” importance are typically Requirements and Code.

6.2 Component Questions

C1: During development:

Answer Alternatives		
There is too much reuse of code / design components going on.	The percentage of code/design components reused is as high as possible (optimized).	There should be more reuse of code / design components.
1	4	3

Comments: Most of the candidates believe that the percentage of reuse is as high as possible, although almost as many thinks there should be more reuse. One candidate thinks that there is too much reuse of code/design in the project under study.

C2: Do you feel that the process of finding, assessing and reusing existing code / design components is functioning?

Answer Alternatives	
Yes	No
4	5

Comments: Almost a split on this question, but the fact that more than half of the candidates think that the process is not functioning should be a strong pointer that this is something that should be improved.

C3a: Is the existing code / design components sufficiently documented?

Answer Alternatives		
Yes	Sometimes	No
0	3	5

C3b: If 'Sometimes' or 'No': Is this a problem?

Answer Alternatives	
Yes	No
7	1

Comments: None of the candidates think the existing components are sufficiently documented, and 7 out of 8 think this is a problem! This is a very interesting discovery. Component documentation and related problems should be subject for further study.

C4: *Would the construction of a reuse repository, with extra component documentation etc:*

Answer Alternatives	
Not be worthwhile: The current system works sufficiently	Be worthwhile: Make finding / reusing components easier
4	3

Comments: Even though the candidates seem to be discontent with component documentation, only three (out of the seven that answered the question) believe that a repository would be helpful. This answer is slightly surprising given the discontent with component documentation.

C5: *How would you decide whether to reuse a code / design component “as-is”, reuse “with modification”, or make a new component from scratch?*

Candidate #	Answer Alternatives			
	By following the guidelines	By consulting experts	By using GSN RUP	Not clearly defined
	3	6	4	2
1	x	x	x	
2	x	x		
3	x	x	x	x
4		x		
5				x
6			x	
7		x		
8		x		
9			x	

Comments: The answer from this question is possibly rendered useless since three of the candidates tagged more than one answer (the remaining five candidates probably use more than one of the alternatives as well...). Still, most of the candidates have tagged that they consult experts, and four have tagged GSN RUP.

C6: *A code / design component that is reused (and possibly modified) is usually:*

Answer Alternatives		
More stable and cause less problems than a component that is created from scratch	About equal to a component created from scratch	Inferior to a component created from scratch (in performance, stability and so on)
6	2	1

Comments: Most of the candidates believe that components that are reused are more stable and cause fewer problems. This supports the usefulness of reuse in Software Development.

C7: *Integration when reusing a component*

Answer Alternatives		
Usually works well (the components usually fit easily into the architecture)	May cause some problems	Is difficult (hard to fit component into architecture)
1	7	0

Comments: Most of the candidates think that integration may cause some problems, but none of the candidates think integration is difficult/hard. It might be useful to do some further study on component integration and related problems.

C8: *Is any extra effort put into testing/documenting potentially reusable components?*

Answer Alternatives	
Yes	No
4	5

Candidate Comment: "We do not consider reuse well enough when we make a component and thus we do not get design, test and documentation to that end".

Comments: Almost a split. Of further interest is the fact that all the candidates that answer 'no' work as software designers. In question C3, none of the candidates thought the existing components were sufficiently documented (one of the candidates even commented on this). These questions are definitely related. The process for documenting components in the project under study is something that should be considered for revision.

Comments – Components

By evaluating the results from the reuse-component section of the questionnaire we note that there is some discontent among the candidates. Most candidates seem to agree that the level of reuse of the project under study is as high as possible, but there seem to be a general agreement that the process of achieving this level of reuse could be improved. Half of the candidates said that the process for finding, assessing and reusing code/design was not functioning.

One interesting discovery was that none of the candidates seemed to think the existing components were sufficiently documented. A hint to how this could be improved was given by the fact that half of the candidates (all of whom were software designers) said that no extra effort were put into documenting components that were likely to be reused (Ericsson should definitely consider improving their routines for documenting components in the project under study). Furthermore, there seemed to be a general agreement that integration of components may sometimes cause problems; this is another issue that may be related to poor component documentation.

Integration problems aside, most candidates agreed that components that were reused were more stable and caused fewer problems than components that were created from scratch.

6.3 Requirements

R1: Is the requirements renegotiation process at Ericsson working sufficiently?

Answer Alternatives	
Yes	No
4	4

Comments: Half of the candidates that answered the question think that the process is not working sufficiently.

R2: In a typical project:

Answer Alternatives		
Requirements are usually flexible, and may often be adjusted	No particular trend (sometimes rigid, sometimes flexible)	Requirements are usually very rigid, little or no change can be negotiated
3	4	1

Comments: A slight slope towards no trend, only one thinks that requirements can not be negotiated and changed.

R3: Are requirements often changed / renegotiated during a development project

Answer Alternatives		
Often	Sometimes	Seldom
6	2	0

Candidate Comment: "The process and responsibility around requirement renegotiation could be clearer. Too often System Management does not consult SW Design when renegotiating. This leads to requirements being ambiguous, frivolous, or may contain design choices that are undesirable"

Comments: Most candidates say that requirements are subject to frequent changes.

Comments - Requirements

There seem to be some discontent with how requirements negotiation is functioning. The trend as not as clear as the discontent with how reuse of components is functioning, but it should still be taken seriously. Given the fact that the candidates say they experience that requirements change frequently, and the fact that they trend towards thinking requirements are flexible regarding change, the usefulness of optimizing requirement negotiations should not be underestimated. One of the candidates also gave a short comment where he/she complained about poor communication between SW Management and SW Design when renegotiating. The importance of requirements being consistent is very important, so further study regarding this issue should be performed.

6.4 Application Frameworks

AF1: *Do you know the existing framework well?*

AF1a: *Framework subsystems: (Authors' note: The original question referred to the name of the package)*

Answer Alternatives	
Yes	No
4	5

AF1b: *Interfaces*

Answer Alternatives	
Yes	No
4	5

AF1c: *Design rules*

Answer Alternatives	
Yes	No
6	3

Comments: It seems that the few that know the Cappella subsystems are all Software designers; the same goes with knowledge about the Interfaces. 6/9 of the candidates know the design rules, the surprise here being that one of the most experienced people of our assortment and also a software designer does not know the design rules well.

AF2a: *Is the framework sufficiently documented?*

Answer Alternatives	
Yes	No
2	6

AF2b: *If 'No', how would you prefer the documentation?*

Answer Alternatives		
Text	Web pages	Described through models
2	1	3

Comments: A vast majority thinks that the framework of the project under study is not sufficiently documented; only two out of eight thinks it is. The six that think the framework is not sufficiently documented are split in how they prefer the documentation. Half of them think model descriptions are preferable. The documentation should be improved to obtain better efficiency.

AF3: What is your main source of guideline information during Analysis & Design?

Candidate #	Answer Alternatives			
	Other developers	Previous work	GSN RUP	Other
	3	7	4	0
Candidate #	Answer Alternatives			
	Other developers	Previous work	GSN RUP	Other
1	x	x		
2	x	x	x	
3			x	
4	x	x		
5		x		
6		x		
7			x	
8		x		
9		x	x	

Comments: The answer from this question is possibly rendered useless since four of the candidates tagged more than one answer (the remaining five candidates probably use more than one of the alternatives as well...). By the answers in this table, one can see that some of the candidates tagged more than one alternative. We can see by these answers that six out of eight think that "Previous work" is a good source of guideline information. Half of those that tagged "Previous work" seem like they mean that this information should be supplemented with information from other developers.

Comments - Application Frameworks

The fact that the nine candidates that participated in the survey have different roles in software development (designer, architect, UC tester and so on) may be the reason that they have different knowledge of the framework. Experience in the company may also be a significant factor for this, although one instance proved this wrong.

Only two out of eight thinks that the framework is sufficiently documented. This is something that Ericsson should consider improving, most preferably through models description.

It seems that experience in the company leads to the best source of guideline information, both from previous work and from other developers. GSN RUP is a good source, but most probably as an introduction to the system, and a work of reference.

6.5 GSN RUP Workflows

GSN1: Do you refer to GSN RUP workflows during Requirements management?

Answer alternatives				
Always	Quite often	Seldom	Never, as they are useless	Never, as I know what to do
1	5	1	0	0

Comments: The GSN RUP workflows are typically 'quite often' used during Requirement management.

GSN2: Do you refer to GSN RUP workflows during Analysis and Design?

Answer alternatives				
Always	Quite often	Seldom	Never, as they are useless	Never, as I know what to do
1	7	1	0	0

Comments: The GSN RUP workflows are typically 'quite often' used during Analysis and Design.

GSN3: Is the information in the GSN RUP web pages understandable?

Answer Alternatives	
Yes	No
9	0

Comments: All the candidates think that the GSN RUP pages are understandable (The authors of this thesis concur that the pages are well organized and easy to navigate).

The current adaptation of GSN RUP does not include reuse activities such as

- § Comparing candidate components
- § Evaluating existing components
- § Deciding whether to modify an existing component versus creating a new component

GSN4: Will improving such reuse activities in GSN RUP workflow:

Answer Alternatives		
Have little or no impact on the development	Have positive effects / improve development	Have negative consequences (result in more work / give a more complex and less helpful development process / have other negative effects)
1	8	0

Comments: The candidates are positive towards including extra reuse related activities in GSN RUP.

Comments - GSN RUP Workflows

According to the questionnaire, the GSN workflows are important guidelines during both requirements management and Analysis and Design ('quite often' referred to). The candidates agree that the GSN RUP web pages are understandable.

Most of the candidates are positive to including new reuse activities in GSN RUP. (This is a very encouraging result for this thesis, as one of the goals of the thesis is to provide just such activities).

6.6 Architecture Compliance

AC1a: Do you feel that the existing criteria for compliance with architecture are clearly defined concerning component integration?

Answer Alternatives		
Yes, they are clearly defined	To some degree, but it is rather fuzzy	No, they are not clearly defined
1	7	1

AC1b: If not 'Yes', does these shortcomings often lead to problems (trying to fit unsuited components and so on)?

Answer Alternatives		
Yes, problems may arise during integration, testing etc.	Sometimes	No, even though the criteria for compliance is poorly defined, this does not lead to any problems
2	6	0

Comments: The candidates generally believe that the criteria for architecture compliance when integrating components are somewhat fuzzy and that this may cause problems (This was the middle alternative).

AC2: Are criteria for design regarding non-functional requirements (especially capacity and stability) well defined?

Answer Alternatives	
Yes	No
3	5

Comments: A slight majority says that the criteria for design regarding non-functional requirements are not well defined. When reusing components handling non-functional requirements is very important, so it may be useful to perform some further study regarding this result.

AC3: *Do you test a component for non-functional properties before integration with other components (or are only the functional requirements are tested?)*

Answer Alternatives		
Yes	Sometimes	No
2	4	2

Comments: Only two of the candidates answer that they test nonfunctional requirements before integration. This could be a problem, and is something the project should study further.

Comments - Architecture Compliance

The answers in this part of the questionnaire typically hit the middle of the answer range. The candidates think the criteria for architecture compliance when integrating components is somewhat fuzzy and that this may sometimes cause problems. It may be useful to make the criteria slightly less fuzzy, and by doing so reducing some of the problems.

The most interesting part in this section of the questionnaire was the questions regarding non-functional requirements. In a software development organization such as Ericsson that has large-scale reuse of components, there should be a major focus on handling non-functional requirements. Only 3 out of 8 candidates think that the criteria for handling non-functional requirements is clearly defined, and only 2 say they always check non-functional requirements prior to integrating a component (although 4 say they sometimes do so), this is definitely something that should be improved.

6.7 Evaluation of the proposed hypotheses

Prior to creating the questionnaire we proposed 3 hypotheses. The questions in the questionnaire were formed to support/reject these hypotheses. Given the nature of surveys, and the relatively small size of this survey (9 candidates), we will only make some general comments regarding the hypothesis. The data provided is not sufficient to provide any scientifically valid answers. Each of the 3 hypotheses has associated null hypotheses.

H₀₁: Reuse in Software Development gives no significant advantages.

H₁: Reuse in Software Development gives significant advantages.

In support of the new hypothesis H₁:

In question G1, the mode answer was 'high' when the candidates were asked the importance of reuse in achieving lower development costs, reuse achieving shorter development time, reuse in achieving a more standardized architecture and reuse in achieving lower maintenance costs. In question G1-c concerning the importance of reuse in achieving better product quality 'medium' was the mode answer.

In question G2, The usefulness / importance of Reuse / Component-based technologies, all the answers were in the range 'high' – 'very high'.

In question C6, most of the candidates (6 out of 9) said those components that are reused usually are more stable and cause less problems than components created from scratch (2 said they were about equal, and 1 said such components were inferior).

In support of the null hypothesis H₀₁:

In question C7 most of the candidates agreed that integration when reusing components may cause some problems.

Conclusion H₁ / H₀₁:

The results concerning the first associated hypotheses were almost entirely in favor of the suggested hypothesis (*H₁*). The null hypothesis may be discarded.

H₀₂: The process of choosing between: reuse “as-is”, reuse “with modification”, or making a new design/code component from scratch, works fluently using the current process.

H₂: It is not easy for a given design/code component to choose between reuse “as-is”, reuse “with modification”, or make new from scratch. Even when reuse of components occurs, we have no way to ensure that the best candidate component has been selected.

In support of the new hypothesis H₂:

In question C2, 5 out of 9 candidates said that the current process for finding, assessing and reusing existing code / design components was not functioning. One issue the candidates were particularly dissatisfied with was component documentation. In question C3, none of the candidates thought the current components were sufficiently documented and 7 out of 8 thought this was a problem. In question C8, 5 of the candidates said that no extra effort was put into testing / documenting potentially reusable components (This probably indirectly makes the process of finding and assessing components more difficult).

Also some issues in regarding development guidelines, etc. are in favor of *H₂*. In question G3 regarding the importance of requirements with respect to reuse and design regarding reuse, the mode answers were ‘medium’ and ‘high’. Since requirements and design were considered of importance in reuse, any problems the candidates reported concerning these fields may have some relation with *H₂ / H₀₂*; choosing between: reuse “as-is”, reuse “with modification”, or making a new design/code component from scratch. In question R1 4 out of 8 candidates said that the renegotiation in the project under study was not working properly. 6 out of 8 thought the current framework was not sufficiently documented. In question GSN4, 8 of the candidates thought the introduction of reuse activities in GSN RUP would improve development.

Supporting *H₂*, is also question AC1 where most of the candidates said the criteria for compliance with architecture were fuzzy when concerning component integration and that this could sometimes cause problems (this probably makes the process of assessing components more difficult). In question AC2, 5 out of 8 said the criteria for evaluating non-functional requirements was not well defined. In question AC3 only 2 candidates said they always tested non-functional requirements prior to integrating components (4 said sometimes, 2 said they did not check non-functional requirements). (Testing of non-functional requirements is according to most literature an important part of component assessment).

In support of the null hypothesis H₀₂:

In each question mentioned above, some candidates were happy with how things work (but none of the candidates were satisfied with all the issues mentioned).

Conclusion H₂ / H₀₂:

The results concerning the first associated hypotheses were generally in favor of the

suggested hypothesis (H_2). The null hypothesis may be discarded.

H_{03} *The current guidelines and workflows works well, and the criteria for architecture compliance is well defined*

H_3 *Better guidelines and workflows for development are needed. Criteria for compliance with existing architecture are not clearly defined.*

In support of the new hypothesis H_3 :

In question C8, 5 out of 9 candidates say that they do not put any extra effort into documenting components that are potentially reusable; this may be something that could be changed with better guidelines / workflows. The need for better guidelines / workflows when documenting components is further made visible by the fact that 5 of the candidates thought the existing components were not sufficiently documented (3 said sometimes, zero said that they were sufficiently documented). The questionnaire also showed that better guidelines in requirements might be needed. In question R1, 4 out of 8 said that the requirements renegotiation process in the project under study was not sufficiently working.

In question GSN4, 8 out of 9 said the introduction of reuse activities into GSN RUP would be useful.

In question AC1 most of the candidates agreed that the existing criteria for architecture compliance when integrating components was somewhat fuzzy. In AC2 5 out of 8 said that the criteria for design regarding non-functional requirements was not clearly defined.

In support of the null hypothesis H_{03} :

In question AF2 6 out of 8 said the current framework was not sufficiently documented. The problems mentioned above may be related to how the current framework is documented and not the quality of the guidelines / workflows as such. On the other hand, in question GSN3, all 9 candidates agreed that the GSN RUP pages were understandable. In question AF1 the candidates gave answers on which frameworks they were familiar with. 5 out of 9 said they did not know Cappella, 5 out of 9 said they did not know the Interfaces and 3 out of 9 said they did not know the design rules. This result is of course related to what tasks the different candidates perform at Ericsson, but it may still have some importance in the current discussion. It may be that the problems above is related to the candidates having insufficient knowledge concerning the guidelines / workflows, and not that guidelines / workflows are poor the way they are formed.

Conclusion H_3 / H_{03} :

The first part of H_3 , whether better guidelines / workflows are needed is difficult to reject or accept given the problems mentioned concerning the documentation of guidelines / workflows, and the fact that some of the employees may not have sufficient knowledge about them. The second part of the hypothesis, concerning criteria, seems to be correct (according to the limited information provided by this questionnaire).

7 Conclusion and further work

In this thesis we have made a study of RUP, the GSN project's Adaptation of RUP (GSN RUP) as well as the current development process at Ericsson. We have done this with a strong focus on reuse. We have made a survey to get better understanding of how some Ericsson employees view their current situation, and we have used the answers from this survey to test 3 hypotheses. Furthermore we have made some suggestions to how GSN RUP should be further adapted to better incorporate reuse.

Of the below hypotheses, the first two were deemed correct, while there is still some doubt regarding the last (although most of the proof were in favor of it being correct). Also, the results from the evaluation of the survey may not be scientifically valid because of the small scale of the survey (9 persons).

H₁: Reuse in Software Development gives significant advantages.

H₂: It is not easy for a given design/code component to choose between reuse "as-is", reuse "with modification", or make new from scratch. Even when reuse of components occurs, we have no way to ensure that the best candidate component has been selected.

H₃: Better guidelines and workflows for development are needed. Criteria for compliance with existing architecture are not clearly defined.

Regarding our adaptations in GSN RUP, several activities were suggested for improving reuse of components. We believe that by including these activities, Ericsson could better find, assess and compare components, they could better prepare newly created components for future reuse, and so on. In the survey 8 out of 9 candidates were positive to such an adaptation.

As for further work, we urge Ericsson to follow our advice and include reuse activities in their GSN RUP adaptation.

This study (particularly the survey) also discovered some other issues Ericsson should address. These include:

- Make further study regarding how components are documented (most of the candidates agreed that bad documentation of existing components caused problems)
- Make further study regarding how the requirements renegotiation process is performed (there were some discontent on this among the participants in the survey) In particular look at how the different groups (such as SW Management and SW Design) communicate changes in requirements. It is very important that requirements remain consistent throughout the organization. One of the candidates reported problems regarding this issue.
- Make further study regarding the framework documentation at Ericsson (most of the participants in the survey agreed that the current framework was insufficiently documented).

The survey performed in this thesis included only 9 test persons. Ericsson should perform a more extensive survey in order to test the validity of the results obtained.

Appendices

A. References

- [1] Phillippe Kruchten: "The Rational Unified Process – An Introduction, Second Edition", 2000, Addison Wesley, ISBN 0-201-70710-1
- [2] Neil A. Maiden and Cornelius Ncube: "Acquiring COTS Software Selection Requirements", March/April 1998, IEEE Software, p. 46-56, Volume 15 Issue 2
- [3] Barry Boehm: "Requirements the Handle IKIWISI, COTS and Rapid Change", July 2000, Software Management Magazine, p. 99-102, Volume 33 Issue 7
- [4] Ivar Jacobson, Martin Griss and Patrick Jonsson : "Software Reuse – Architecture, Process and Organization for Business Success", 1997, ACM Press, ISBN 0-201-92476-5
- [5] Walseth,Ole Anders and Naalsund, Erlend: "Component-Based Development – Models for COTS / Software Reuse", Student Project at NTNU, Norway, Fall 2001, <http://www.idi.ntnu.no/grupper/su/sif8094-reports/p4.pdf>
- [6] Even-André Karlsson (Ed.): "Software Reuse: A Holistic Approach" (The REBOOT Methodology Handbook), Wiley Series in Software Based Systems. John Wiley. 510 p., 1995. ISBN 0-471-95819-0, REBOOT report no. 8218. A very well-respected book on reuse, with large contributions from SINTEF and NTNU.
- [7] Ochs,M., Pfahl,D., Chrobok-Diening,G., Nothhelfer-Kolb,B: "A Method for Efficient Measurement-based COTS Assessment and Selection – Method Description and Evaluation Results" in Proceedings IEE 7th International Software Metrics Symposium, London, England, 4-6 April 2001. pp. 285-296
- [8] Guttorm Sindre, Reidar Conradi, Even-André Karlsson: "The REBOOT Approach to Software Reuse", Journal of Systems and Software (Special Issue on Software Reuse), Vol. 30, No. 3, (Sept. 1995), p. 201-212.

Internal information from the intranet of Ericsson AS, GSN RUP version R3A.

A.1 Recommended Reading

Maurizio Morisio, Marco Torchiano: "Definition and classification of COTS: a proposal", 2001, Submitted to publication, 10 pages.
<http://www.idi.ntnu.no/emner/sif80at/curriculum/ICCBSS-sumit.pdf>

Ivar Jacobson, Martin Griss, and Patrik Jonsson: "Software Reuse Architectures, Process and Organization for Business Success", ACM Press and Addison Wesley Longman, 495p., 1997, ISBN 0-201-92476-5.

Parastoo Mohagheghi, Reidar Conradi.: "Experiences with certification of reusable components in the GSN project in Ericsson, Norway", In Judith Stafford et al. (Eds.): "Proc. 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction" (before ICSE'2001), Toronto, May 14-15, 2001, p. 27-31. SU-report 6/2001

Maurizio Morisio, Carolyn Seaman, Amy Parra, Victor Basili, Steve Kraft, and Steve Condon: "Investigating and Improving a COTS-Based Software Development Process", Proc. 22nd International Conference on Software Engineering (ICSE'2000), p. 31-40, ACM Order No. 592000.

Craig Larman: "Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design", 1998, Prentice Hall, ISBN 0-13-748880-7

X. Burqués, C. Estay, X. Franch, J.A. Pastor, C. Quer: "Combined Selection of COTS Components", Paper accepted at the international conference on COTS-based software systems, Florida, feb. 2002

Ochs,M., Pfahl,D., Chrobok-Diening,G., Nothhelfer-Kolb,B: "A COTS Acquisition Process: Definition and Application Experience", published in the proceedings of the ESCOM 2000 Conference and on ESCOM website:
<http://www.escom.co.uk>

B. Technical information

B.1 Questionnaire of the software development process and reuse aspects at Ericsson, Grimstad

The purpose of this questionnaire is to map how Ericsson employees view their development process especially concerning reuse, as well as where they feel changes of the development process may be necessary.

The questionnaire consists of multiple-choice questions. Mark the choices you feel is most correct. Be honest; if you feel the current developments process works perfectly and is in need of no changes, say so.

The questions are enumerated by the category they belong to, and a number. The Components category has questions C1, C2, C3 and so on. If a question is depends on a particular answer from the prior question, these two questions are enumerated by an "a" and "b". C4a and C4b are two such questions. Feel free to add comments.

Personal Info:

P1: What is your current role at Ericsson?

P2: How long have you been working at Ericsson?

P3: How many projects have you been involved in?

P4: Which programming and design languages do you master?

P5: Which programming and design languages are you currently using?

P6: What is your education degree?

General

G1: There are several benefits to reuse code / design components. How important do you consider reuse in achieving the following benefits:

G1a: For achieving lower development costs, reuse is of _____ importance:
 very high high medium little no
 don't know

G1b: For achieving shorter development time, reuse is of _____ importance:
 very high high medium little no
 don't know

G1c: For achieving higher product quality, reuse is of _____ importance:
 very high high medium little no
 don't know

G1d: For achieving a more standardized architecture, reuse is of _____ importance:
 very high high medium little no
 don't know

G1e: For achieving lower maintenance costs (including technology updates), reuse is of _____ importance:
 very high high medium little no
 don't know

Comments:

Components

C1: *During development:*

- There is too much reuse of code / design components going on.
- The percentage of code/design components reused is as high as possible (optimized).
- There should be more reuse of code / design components.

C2: *Do you feel that the process of finding, assessing and reusing existing code / design components is functioning?*

- Yes
- No

C3a: *Is the existing code / design components sufficiently documented?*

- Yes
- Sometimes
- No

C3b: *If 'Sometimes' or 'No': Is this a problem?*

- Yes
- No

C4: *Would the construction of a reuse repository, with extra component documentation etc:*

- Not be worthwhile: The current system works sufficiently
- Be worthwhile: Make finding / reusing components easier

C5: *How would you decide whether to reuse a code / design component "as-is", reuse "with modification", or make a new component from scratch?*

- By following the guidelines
- By consulting experts
- By using GSN RUP
- Not clearly defined

C6: *A code / design component that is reused (and possibly modified) is usually:*

- More stable and cause less problems than a component that is created from scratch
- About equal to a component created from scratch
- Inferior to a component created from scratch (in performance, stability and so on)

C7: *Integration when reusing a component*

- Usually works well (the components usually fit easily into the architecture)
- May cause some problems
- Is difficult (hard to fit component into architecture)

C8: *Is any extra effort put into testing/documenting potentially reusable components?*

- Yes
- No

Comments:

Requirements

In most software development projects, the initial set of requirements may change during the course of the projects. The customers may think of new features they want to add, the developers may find some requirements unfeasible (or possibly easier) to fulfil and so on. In such cases it may be necessary for the stakeholders to renegotiate requirements.

R1: *Is the requirements renegotiation process at Ericsson working sufficiently?*
 Yes
 No

R2: *In a typical project:*
 Requirements are usually flexible, and may often be adjusted.
 No particular trend (sometimes rigid, sometimes flexible).
 Requirements are usually very rigid, little or no change can be negotiated

R3: *Are requirements often changed / renegotiated during a development project*
 Often
 Sometimes
 Seldom

Comments:

B.2 Work Schedule

This is the work schedule that was written early in the project.

Week	Topic	Activity
7	RUP/GSN RUP	Deliver summary on RUP. Visit Ericsson, Grimstad. Keywords: Decision Points, Interface Documentation and Domain Engineering.
8	GSN RUP	Continue work on GSN RUP from what we have learned during our visit.
9	GSN RUP: General	Deliver summary on GSN RUP plus probably some propositions to the keywords above.
10	GSN RUP: Criteria	Start working on the criteria for evaluating components where the architecture is already developed.
11	Criteria	Continue criteria work
12	Criteria	Continue criteria work
13	EASTER	HOLIDAY
14	GSN RUP: Various	Decision points, documentation of interfaces/components and domain engineering
15	GSN RUP: Various	Decision points, documentation of interfaces and domain engineering
16	A&D	Reuse aspects in development of components. (Introduce FIS). Visit to Ericsson this or next week.
17	A&D	More on reuse.
18	Checklist: Study	Look at the FIS (Feature Impact Study) template and try to change it for reuse purposes.
19	Checklist: Creation	Continue working on the FIS (Feature Impact Study) template. Make checklist.
20	Checklist: Practice	Try to use our checklist in either a case study or an experiment at Ericsson. Another visit to Grimstad is necessary.
21	Checklist: Evaluation	Document and evaluate results from last week
22	Report	Try to finish report
23	Report	Continue with the report
24	Report	Delivery of the final thesis June 17 th

C. Ericsson Related Information

The subsection with Ericsson sensitive information has been omitted in this version of the document. Please contact

Parastoo Mohagheghi parastoo.mohagheghi@eto.ericsson.se at Ericsson AS, Grimstad or Reidar Conradi reidar.conradi@idi.ntnu.no at NTNU, Trondheim to acquire this information.