Meta-Method for Method Configuration
- A Rational Unified Process Case

by

Fredrik Karlsson

Submitted to the Faculty of Arts and Sciences at Linköping University in partial fulfilment of the requirements for the degree of Licentiate of Philosophy

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2002
Meta-Method for Method Configuration  
- A Rational Unified Process Case

by

Fredrik Karlsson

Oct 2002  
ISBN 91-7373-474-8  
Faculty of Arts and Sciences 61  
ISSN 1401-4637

ABSTRACT

The world of systems engineering methods is changing as rigorous ‘off-the-shelf’ systems engineering methods become more popular. One example of such a systems engineering method is Rational Unified Process. In order to cover all phases in a software development process, and a wide range of project-types, such methods need to be of an impressive size. Thus, the need for configuring such methods in a structured way is increasing accordingly. In this thesis, method configuration is considered as a particular kind of method engineering that focuses on tailoring a standard systems engineering method. We propose a meta-method for method configuration based on two fundamental values: standard systems engineering method’s rationality and reuse. A conceptual framework is designed, introducing the concepts Configuration Package and Configuration Template. A Configuration Package is a pre-made ideal method configuration suitable for a delimited characteristic of a (type of) software artifact, or a (type of) software development project, or a combination thereof. Configuration Templates with different characteristics are built combining a selection of Configuration Packages and used as a base for a situational method. The aim of the proposed meta-method is to ease the burden of configuring the standard systems engineering method in order to reach an appropriate situational method.

This work has been supported by The Swedish Knowledge Foundation and Volvo IT.
Information systems development is a discipline within the philosophical faculty at Linköping University. Information systems development is a discipline studying human work with developing and changing computer-based information systems in organisational settings. It includes theories, strategies, models, methods, co-working principles and tools concerning information systems development. Different development/change situations can be studied as planning, analysis, specification, design, implementation, deployment, evaluation, maintenance and redesign of information systems and its interplay with other forms of business development. The discipline also includes the study of prerequisites for and results from information systems development, as e.g. studies of usage and consequences of information systems.

This work, Meta-method for Method Configuration – A Rational Unified Process Case, is written by Fredrik Karlsson at Örebro University. He is also a member of research group VITS. He presents this work as his licentiate thesis in information systems development, Department of Computer and Information Science, Linköping University.

Linköping October 2002

Göran Goldkuhl
Professor
DOCTORAL DISSERTATIONS IN INFORMATION SYSTEMS DEVELOPMENT


LICENTIATE THESES IN INFORMATION SYSTEMS DEVELOPMENT


5. dan Fristedt (1995) Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys


15. Bengt EW Andersson (1999) Samverkande informationssystem mellan aktörer i offentliga åtaganden - en teori om aktörsarenor i samverkan om utbyte av information


23. Stefan Holgersson (2001) IT-system och filtrering av verksamhetskunskap – kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system
24. Per Oscarson (2001) Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering i verksamheter


ACKNOWLEDGEMENTS

Hope, joy, anger, frustration and long hours are some words that capture the essence of thesis writing. Thus it is with both relief and emptiness I conclude that finally all pieces have found their places. As always you do not accomplish such a puzzle on your own and I am indebted to several people. The writing of this thesis would not have been possible without your help and support.

My gratitude goes to those of you who were engaged participants during the meta-method project; my colleagues Pär Ågerfalk, Anders Hjalmarsson, and Kai Wistrand. To the people at Volvo IT in Gothenburg, especially Boris Karlsson, Gregor Börjesson and Lars-Åke Hedbom, for facilitating the empirical study and for your valuable contributions about configuration work. At Rational Software Nordic AB I want to acknowledge Thomas Jacobson and Per Löwnertz. To the people at Web Program Center and Volvo IT in Skövde who contributed to the discussion about system engineering projects during the early phases of the meta-method project.

A special gratitude goes to my colleague and fellow PhD-student Emma Eliason who has shared an endless number of trips to Linköping and PhD-courses with me. Today I am glad that we managed our first PhD-course and without your help during the first term it is doubtful whether or not I ever would have written this thesis at all.

Then I want to mention colleagues, former colleagues and friends how have contributed with many important details during the past years: Andreas Sjöwitz, AnnaCarin Gustavsson, Annika Andersson, Annika Kvick, Benny Ottosson, Ella Kolkowska, Jesper Dyberg, Johan Peterson, Jonas Arvidson, Karin Hedström, Kenneth Åhlgren, Lars Taxén, Lina Börjeskog, Magnus Lövgren, Malin Hägmark, Mathias Borg, Per Oscarsson, Per Persson, Pär Fahlén, Ulf Seigerroth. My supervisors: Prof. Göran Goldkuhl and Dr. Anders Avdic for constructive comments and for your ability to give me the freedom I needed. Thank you.

Finally, I want to extend my gratitude to the Karlssons and the Pudas. I would like to thank my mother and father for always believe in my crazy ideas and me. These ideas have been many through the years. You have made it possible for me to combine two impossible worlds, to write this thesis and to run and develop our two
companies at the same time. Anna and Martin, I am glad that your kids do not understand method configuration – at least not yet!

This work has been supported by The Swedish Knowledge Foundation and Volvo IT.

Fredrik Karlsson, Örebro October 2002
1 INTRODUCTION ...............................................................................................................1
  1.1 SYSTEMS ENGINEERING METHODS.................................................................1
    1.1.1 Gathering Oranges.......................................................................................1
    1.1.2 Back to Reality..............................................................................................3
  1.2 STANDARD SYSTEMS ENGINEERING METHODS ...........................................6
  1.3 SITUATIONAL SYSTEMS ENGINEERING METHODS .......................................10
  1.4 METHOD ENGINEERING.................................................................................11
  1.5 METHOD CONFIGURATION BASED ON RUP ...............................................12
  1.6 EXPERIENCES OF RUP CONFIGURATION ...................................................14
  1.7 THE NEED FOR METHOD SUPPORT ...........................................................16
  1.8 RESEARCH QUESTION......................................................................................18
  1.9 CHARACTERIZATION OF THE RESEARCH QUESTION ...............................19
  1.10 AIM OF THESIS..............................................................................................19
  1.11 STAKEHOLDERS.............................................................................................20
  1.12 DEMARCATIONS.............................................................................................21
  1.13 THESIS OUTLINE...........................................................................................22
  1.14 READING INSTRUCTIONS..............................................................................22

2 RESEARCH DESIGN .................................................................................................25
  2.1 FOUNDATIONS FOR RESEARCH APPROACH.................................................25
    2.1.1 Domain Levels............................................................................................25
    2.1.2 Methods as Knowledge..............................................................................27
    2.1.3 Grounding The Meta-method .................................................................29
  2.2 RESEARCH APPROACH ..................................................................................29
    2.2.1 The Research Environment: RUP ............................................................30
    2.2.2 The Research Environment: Business Contexts.......................................30
      2.2.2.1 Volvo IT ...............................................................................................33
      2.2.2.2 Örebro University - ESI .......................................................................34
    2.2.3 Research Process.......................................................................................34
      2.2.3.1 Elaboration .........................................................................................38
      2.2.3.2 Design ...............................................................................................39
# 6 A DETAILED DESCRIPTION OF THE METHOD FOR METHOD CONFIGURATION

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>MMC SECTION 1...</td>
<td>112</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Manage Change Request for Configuration Package</td>
<td>112</td>
</tr>
<tr>
<td>6.1.1.1</td>
<td>Activity: Identify Potential Changes</td>
<td>112</td>
</tr>
<tr>
<td>6.1.1.2</td>
<td>Activity: Evaluate Potential Changes</td>
<td>114</td>
</tr>
<tr>
<td>6.1.1.3</td>
<td>Activity: Identify Potential Values</td>
<td>116</td>
</tr>
<tr>
<td>6.1.1.4</td>
<td>Activity: Evaluate Values</td>
<td>118</td>
</tr>
<tr>
<td>6.1.1.5</td>
<td>Activity: Check with Existing Characteristics</td>
<td>120</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Define Configuration Package</td>
<td>121</td>
</tr>
<tr>
<td>6.1.2.1</td>
<td>Activity: Identify Relevant Prescribed Actions</td>
<td>122</td>
</tr>
<tr>
<td>6.1.2.2</td>
<td>Activity: Consider Importance and Complexity</td>
<td>123</td>
</tr>
<tr>
<td>6.1.2.3</td>
<td>Activity: Configuration Package Validation</td>
<td>128</td>
</tr>
<tr>
<td>6.2</td>
<td>SECTION 2...</td>
<td>129</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Manage Change Request for Configuration Template</td>
<td>129</td>
</tr>
<tr>
<td>6.2.1.1</td>
<td>Activity: Identify Potential Configuration Template</td>
<td>130</td>
</tr>
<tr>
<td>6.2.1.2</td>
<td>Activity: Evaluate Potential Configuration Template</td>
<td>132</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Define Configuration Template</td>
<td>134</td>
</tr>
<tr>
<td>6.2.2.1</td>
<td>Activity: Combine Configuration Packages</td>
<td>134</td>
</tr>
<tr>
<td>6.2.2.2</td>
<td>Activity: Validate The Configuration Template</td>
<td>138</td>
</tr>
<tr>
<td>6.3</td>
<td>SECTION 3...</td>
<td>139</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Select Configuration Template</td>
<td>139</td>
</tr>
<tr>
<td>6.3.1.1</td>
<td>Activity: Characterize Project</td>
<td>139</td>
</tr>
<tr>
<td>6.3.1.2</td>
<td>Activity: Match With Configuration Template</td>
<td>141</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Fine-tune Configuration Template</td>
<td>142</td>
</tr>
<tr>
<td>6.3.2.1</td>
<td>Activity: Fine-Tune The Configuration Template</td>
<td>143</td>
</tr>
<tr>
<td>6.3.2.2</td>
<td>Activity: Validate The Situational Systems Engineering Method</td>
<td>145</td>
</tr>
</tbody>
</table>

# 7 EPISODES OF USE

<table>
<thead>
<tr>
<th>Episode</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>THE EARLY WORKSHOP</td>
<td>148</td>
</tr>
<tr>
<td>7.1.1</td>
<td>The Process</td>
<td>149</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Design Implications</td>
<td>153</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Experiences Gleaned</td>
<td>154</td>
</tr>
<tr>
<td>7.2</td>
<td>THE LARGE WORKSHOP</td>
<td>156</td>
</tr>
<tr>
<td>7.2.1</td>
<td>The Process</td>
<td>157</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Analysis Based on RPW and MMC</td>
<td>158</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Mapped Concepts</td>
<td>160</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Experiences Gleaned</td>
<td>161</td>
</tr>
<tr>
<td>7.3</td>
<td>THE PERSONNEL SYSTEM</td>
<td>162</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Known Business Objectives</td>
<td>163</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Modeling Web Application</td>
<td>164</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Quality Assurance</td>
<td>166</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Existing Environment</td>
<td>168</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Thin Client Deployment</td>
<td>170</td>
</tr>
<tr>
<td>7.3.6</td>
<td>Experiences Gleaned</td>
<td>172</td>
</tr>
<tr>
<td>7.4</td>
<td>SUMMARY</td>
<td>173</td>
</tr>
</tbody>
</table>
8 CONCLUSIONS AND FURTHER RESEARCH..............................................175
  8.1 ACHIEVING THE AIM.................................................................175
  8.2 PRESCRIBED ACTIONS FOR A FOCAL AREA.........................176
  8.3 EMPIRICAL EXPERIENCES.............................................................179
  8.4 LIMITATIONS OF METHOD FOR METHOD CONFIGURATION........181
  8.5 FUTURE RESEARCH.................................................................182
  8.6 RATIONAL UNIFIED PROCESS FOR GATHERING ORANGES?.......183

REFERENCES ..............................................................................................185
FIGURES

FIGURE 1. DEFINITION 1-3 REPRESENTED AS A UML-DIAGRAM............................... 5
FIGURE 2. RUP OVERVIEW (RUP 2001 DOCUMENTATION)........................................ 12
FIGURE 3. PROBLEM DIAGRAM FOR METHOD CONFIGURATION............................ 17
FIGURE 4. THE METHOD ENGINEER’S ROLE.......................................................... 20
FIGURE 5. LAYERS OF METHOD DEVELOPMENT.................................................. 26
FIGURE 6. STRUCTURE OF THE RESEARCH ENVIRONMENT.................................. 31
FIGURE 7. TIME SPAN FOR THE ACTION CASES .................................................. 33
FIGURE 8. RESEARCH PROCESS........................................................................... 37
FIGURE 9. THE USAGE-SITUATION OF METHOD FOR METHOD CONFIGURATION..... 42
FIGURE 10. THE CONSTITUENTS OF A METHOD (GOLDKUHL, 1991)...................... 49
FIGURE 11. METHOD COMPONENT RELATED TO BRINKKEMPER’S THREE
ORTHOGONAL DIMENSIONS, AN ELABORATION OF WISTRAND AND FROHM
(2001)................................................................................................................. 51
FIGURE 12. METHOD IN RELATION TOPERSPECTIVE (GOLDKUHL, 1994).......... 53
FIGURE 13. THE RELATION BETWEEN PERSPECTIVE, FRAMEWORK, PHASES AND
METHOD, INTERPRETATION OF GOLDKUHL (1991)......................................... 54
FIGURE 14. DEGREES OF FLEXIBILITY IN METHOD ENGINEERING, INTERPRETATION,
ODELL (1996). ..................................................................................................... 57
FIGURE 15. THE PROCESS OF SITUATIONAL METHOD ENGINEERING (HARMSEN, 1997)
.................................................................................................................................. 61
FIGURE 16. THE RELATIONSHIP BETWEEN SITUATION FACTORS, PERFORMANCE
INDICATORS AND SCENARIO ASPECTS (HARMSEN, 1997)................................. 62
FIGURE 17. THREE FOCAL AREAS FOR META-MODELING.................................. 64
FIGURE 18. SIMPLIFIED CONCEPTUAL MODEL OF MMC.................................... 75
FIGURE 19. THE CONCEPTS AND RELATIONSHIP BETWEEN OUR FOCAL AREA, SYSTEMS
ENGINEERING METHOD’S RATIONALITY, AND THE ABSTRACTION OF PROJECTS. 80
FIGURE 20. CONFIGURATION PACKAGE AND CLASSIFICATION OF PRESCRIBED
ACTIONS............................................................................................................... 83
FIGURE 21. SUGGESTED WAYS TO PERFORM THE DIFFERENT PRESCRIBED ACTIONS
OF THE BASE METHOD DERIVED FROM THE IMPORTANCE AND COMPLEXITY OF THE
AREAS WITH WHICH THEY ARE CONCERNED..................................................... 87

VII
FIGURE 23. THE CONFIGURATION TEMPLATE INTENDED FOR A DEVELOPMENT SITUATION...................................................................................................................... 92
FIGURE 24. SITUATIONAL SYSTEMS ENGINEERING METHOD AS A SELECTION FROM CONFIGURATION TEMPLATES................................................................. 95
FIGURE 25. THE METHOD FOR METHOD CONFIGURATION FRAMEWORK................ 97
FIGURE 26. MAIN CONCEPTS OF MMC................................................................. 100
FIGURE 27. BASIC CONCEPTS IN RUP ................................................................ 102
FIGURE 28. METHOD FOR METHOD CONFIGURATION....................................... 105
FIGURE 29. EXAMPLE OF WORKFLOW DETAIL DIAGRAM................................. 107
FIGURE 30. WORKFLOW DETAIL: MANAGE CHANGE REQUEST FOR CONFIGURATION PACKAGE................................................................................................. 112
FIGURE 31. WORKFLOW DETAIL: DEFINE CONFIGURATION PACKAGE.............. 121
FIGURE 32. ANALYSIS OF PRESCRIBED ACTIONS............................................. 124
FIGURE 33. WORKFLOW DETAIL: MANAGE CHANGE REQUEST FOR CONFIGURATION TEMPLATE.............................................................................................. 129
FIGURE 34. WORKFLOW DETAIL: DEFINE CONFIGURATION TEMPLATE......... 134
FIGURE 35. EXAMPLE OF SELECTION PATTERNS WHEN COMBINING CONFIGURATION PACKAGES TO A CONFIGURATION TEMPLATE................................. 136
FIGURE 36. WORKFLOW DETAIL: SELECT CONFIGURATION TEMPLATE.......... 139
FIGURE 37. WORKFLOW DETAIL FINE-TUNE CONFIGURATION TEMPLATE........ 142
FIGURE 38. ANALYZE OF PRESCRIBED ACTIONS................................................ 144
FIGURE 39: THE PROCESS DURING THE SECOND WORKSHOP............................ 158
FIGURE 40. SUGGESTED WAYS FOR PERFORMING DIFFERENT PRESCRIBED ACTIONS OF THE BASE METHOD DERIVED FROM THE IMPORTANCE AND COMPLEXITY OF THE AREAS WITH WHICH THE PRESCRIBED ACTIONS ARE CONCERNED............................... 177
TABLE 1. LAYERS OF GRANULARITY IN RUP ................................................................. 15
TABLE 2. UNITS OF ANALYSIS AND FOCUS................................................................. 43
TABLE 3. PERFORMANCE OF PRESCRIBED ACTIONS IN THE BASE METHOD DEDUCED FROM THE CATEGORIES OF ATTENTION AND ENHANCEMENT ................................................................. 86
TABLE 4. LAYERS OF GRANULARITY IN RUP ............................................................... 103
TABLE 5. ACTIVITY: CAPTURE A COMMON BUSINESS VOCABULARY (RUP 2001). 108
TABLE 6. PERFORMANCE OF PRESCRIBED ACTIONS ................................................. 126
TABLE 7. PERFORMANCE OF PRESCRIBED ACTIONS .................................................. 145
TABLE 8. UNITS OF ANALYSIS AND FOCUS ................................................................. 148
TABLE 9. TENTATIVE CONFIGURATION ........................................................................ 151
TABLE 10. CONFIGURATION OF MODELING: WEB APPLICATION ............................... 165
TABLE 11. CONFIGURATION OF QUALITY VALIDATION: MANUAL TESTING ............ 167
TABLE 12. CONFIGURATION OF PREPARE ENVIRONMENT: EXISTING ENVIRONMENT ................................................................. 169
TABLE 13. CONFIGURATION OF DEPLOYMENT STRATEGY: THIN CLIENT DEPLOYMENT ................................................................................................................. 171
TABLE 14. PERFORMANCE OF PRESCRIBED ACTIONS IN THE BASE METHOD DEDUCED FROM THE CATEGORIES OF ATTENTION AND ENHANCEMENT ................................................................. 178
CHAPTER ONE

1 INTRODUCTION

In this introduction we will present the background and motivation for the thesis. We will legitimate our research question and aim through the use of examples and a first glance at theoretical references. The examples below are used in order to illustrate this author’s personal experience. Finally, we will present demarcations, reading instructions and a thesis outline.

1.1 Systems Engineering Methods

1.1.1 Gathering Oranges

Once in a while you have to admit that you have failed. So did a fictitious project team which the author was part of during a course in project management. The course consisted of both experienced people and novices in the role as project managers. We had different backgrounds ranging from heavy industry to IT-consulting. During one of the exercises we were requested to gather oranges as a project team. The oranges were spread out in a room, measuring roughly six by six meters. We had used this room during one day for other exercises, which made it familiar to us. It was furnished with small tables and chairs grouped together into three larger tables. Two of the walls had windows and below these windows there were tables. We knew, before the exercise started, that there were fourteen oranges to be found. Thus, the project goal was easily defined. However, we did not know where the oranges were placed. Our first thought was that this would be a fairly simple exercise.

Put blindfolds on each participant. Then there we had no possibility to see where each team member was or what he or she was doing. This limited our way of communicating with each other. In fact, most of our usual way of working was not applicable in this situation. When we entered the room for the exercise we had an initial discussion. It was a rather pointless discussion since some of the team members were restless and too focused on the goal – finding those fourteen oranges.
The team started out with enormous energy. Despite all efforts, it took us five minutes before we found our first orange. It was found under a table, attached with tape under the tabletop. Soon, we found more oranges in different places all mounted with tape, but we also encountered problems. We did not know, as a team, which areas and furniture we had searched. At one point we did not even know how many oranges we had gathered. Therefore we, in an ad-hoc manner, decided to stack the searched chairs. This meant that we had to start all over again searching each chair. We also realized that there was more than one level of the room to search. Initially, we had been searching the floor, chairs and tables. But we had missed the ceiling. Carefully, team members started searching this part of the room. It was a slow process since we were doing it in complete darkness. Partly, because the blindfold made it dangerous, partly because it was hard to know exactly where in the room you were situated. Still we did not find all the oranges.

We resigned! One after the other, team members became inactive. Fifty minutes into the exercise the team decided to stop searching. We had gathered thirteen of the fourteen oranges that were to be found. This was considered a failure. We could have argued that the result was close to fourteen, but in fact it was not the amount of oranges that was the failure. The big failure was our way of working as a team. We did not create the right prerequisites within the limitations that we had.

Of course, we had blindfolds and they limited our abilities. Still, there were measures we could have taken. If we examine our fifty minutes it is obvious that we had no procedures for how to organize the search. A room of about thirty-six square meters is a large area to search wearing blindfolds. Nevertheless, we could have improved our situation. For example, we could have divided the room into four sections, making the demarcation with tables. In each section, we could have placed two team members. That would have given each person a specific and limited area to search, instead of roaming around the room. The last team member could have been placed in the middle of these four sections, acting as a communication central and team leader. This way we would have improved communication. Team members would have known where to give and get information about our progress. In our case, the lack of information limited our performance. As an individual team member, you did not know where or how other team members had found oranges. Thus, you could not adapt your own procedures. Furthermore, we had performed our search in an ad-hoc manner. Instead, we could have decided to first sweep the floor, then the furniture and finish with the walls and ceiling. Searching in this way we would only have had four activities to perform within an area limited to nine square meters.

Thus, we can conclude that besides lacking leadership we had performed poorly in creating procedures, establishing communication and using proper tools for the task. That is why this exercise was considered a failure. We lacked a proper method. But of course this was an exercise and had nothing to do with reality, or did it?
1.1.2 Back to Reality

Gathering oranges is a simple activity, while developing software artifacts is a highly complex one. It has always been, and a glance at today’s software artifacts reveals that they are becoming even more complex. Information systems today involve a wide range of techniques and legacy systems that need to be integrated. This becomes legible when we, for example, study the development of Internet-based Software Artifacts 1 (IBSA). Karlsson et. al (2001) argue that the newness of these artifacts might not be as extreme as one first may think, but nonetheless tend to become complex since there are differences in emphasis between IBSAs and traditional software artifacts.

Looking back at system engineering projects, from the view of a consultant, certain aspects of the development process can be addressed that need structural support. During these projects we have had a rather straightforward interpretation of the structural concept, concerning the creation of an inter-subjective understanding of tasks, how the tasks should be divided, performed and communicated among and by the project members. Hence, when projects become more complex and development teams grow larger it is necessary to have a common vocabulary for effective communication. It is important both in working with the stakeholders as well as within the team. We must be certain that central concepts are discussed with the same meaning. In the initial example with oranges, there is a significant difference in the location concept if we know whether or not tape has been used. The location concept is extended tremendously if it is used; it is, for example, possible to attach an orange to the ceiling or the walls, which in our temporary project group were not considered as locations from the beginning. When projects grow we need a way to stay focused. This is another observation that held true in our exercise. We had poor communication and were unfocused. During the exercise we tried to remedy this problem with sub-groups that were formed in an ad-hoc manner. Within these smaller groups communication increased, nevertheless overall communication did not improve.

All projects have a limited amount of resources and a deadline. Within these limitations a certain degree of functionality and quality are to be obtained. Thus, we need support to focus on the most important issues during the phase of current interest. For example, during the start-up phase of a project it is important to learn about the stakeholders’ requirements. Then, we need a structural way to gather those requirements. As projects evolve further, the requirements often change. In our role as system engineers, we learn more about the stakeholders and their needs, and the stakeholders themselves learn more about their own needs and the technological possibilities. Therefore, we have to handle these changes, systematically, during the development process.

These are examples of experiences obtained during development projects but they are not unique. They have been addressed for a long time in software

---

1 IBSA is often referred to as Web Development, but it means an unnecessary demarcation since this type of artifacts is deployed not only through the Web. For example, mobile Internet is the latest trend in making information systems available to users.
Introduction

engineering literature (see for example Brooks, 1995) and are expressed as management problems and as a need for systems engineering method support in the software development process. Technical solutions have to bridge the user requirements in order to generate value. An elaborated discussion about what we consider to be a systems engineering method will be dealt with in Sections 3.1 and 3.2; however here we define the concept method as:

**Definition 1-1:** A method is normative prescribed actions performed by human actors in order to reach the actors’, or a subset of the actors’, goals.

Extending this definition to the field of systems engineering means that we are more precise about the focal area of methods. Hence we are discussing prescribed actions for systems engineering actors. Furthermore we treat systems engineering methods as a subset to the concept of method. We align with Ågerfalk (1999) and deliberately use the term engineering instead of development. Through this distinction we want to emphasize design and construction of an artifact as a structure and planned effort, not only that it is a design and construction effort.

**Definition 1-2:** Systems Engineering is the structured and planned practice of analyzing, constructing, and integrating an information system with an organization’s way of working.

**Definition 1-3:** A systems engineering method is a method supporting the conduction of system engineering.

Definition 1-1 to Definition 1-3 is elaborated in the UML-diagram in Figure 1. It presents a more extended view, which illustrates a systems engineering method’s complexity. The center illustrates normative prescribed actions that should guide the systems engineering actors during their work. We define a prescribed action as below:

**Definition 1-4:** A prescribed action is a description of how to achieve a purpose.

The prescribed actions are not unrelated to each other as illustrated in Figure 1. Two types of associations can be found; prescribed actions can be part of one another and they can be linked to each other on the same level of granularity. Moreover, prescriptive principles do not exist by mere accident; they are designed. Hence, behind each prescribed action we find a purpose, a concept defined as:

**Definition 1-5:** A purpose states why a prescribed action is to be performed and what is to be achieved.
Actors in specific roles carry out prescribed actions in order to achieve these purposes. Thus, we have an elaboration of the actor concept illustrating that actors behave in a specific business context. For example, an actor could have the role of both software architect and programmer, each of which defines different responsibilities. Furthermore, each actor has a different background, and internalizes the prescribed actions differently. We acknowledge that systems engineering methods and their prescribed actions exist (are part of) and are conducted in a business context. Hence, there is a distinction between a systems engineering method as prescribed actions and how it is conducted. We define role, actor and business context as follows:

**Definition 1-6:** A role is a function played by an actor in a project.

**Definition 1-7:** An actor is an individual participating in a project.

---

Figure 1. Definition 1-3 represented as a UML-diagram
Definition 1-8: A business context is a demarcated environment where the system engineering method or its resulting artifacts are used.

When conducting a method’s prescribed action, concepts and notation are used to produce artifacts. Rules are found for concepts that are possible to use during a prescribed action, the way in which these concepts can be related to each other and finally presented. UML could, for example, be the set of rules used during object-oriented modeling of requirements. Hence, concepts are used to facilitate focal areas during systems engineering while notation is used to standardize the content in artifacts. We choose to define concept, and notation as below, where we borrow the definition of the notation concept from Goldkuhl (1991):

Definition 1-9: A concept is an abstraction of a phenomenon.

Definition 1-10: Notation is the rules for producing and interpreting a description.

The definition of the artifact concept is an extension of how the concept is used in RUP 2001. Two extensions have been made. First, we have added the fact that somebody, who is the actor, produces and uses the artifact. Second, the type of projects in question have been qualified to system engineering projects, since we focus on systems engineering methods. With these modifications, we derive the following definition:

Definition 1-11: An artifact is either a final or intermediate work product that is produced and used by actors during a system engineering project.

1.2 Standard Systems Engineering Methods

The need of systems engineering methods should be obvious by now. However, it is not obvious where we, for example, can find a method for gathering oranges. Finding oranges does not seem to be a frequently performed task, which limits the knowledge about its solutions. In reality, we have two choices: in-house development or standard systems engineering methods.

As our first alternative, we could examine our own way of working and elicit a method. This alternative would mean turning implicit ways of working in the organization into an explicit systems engineering method. However, this is not a trivial task (Goldkuhl, 1994). In order to develop a method, you need specific competence often referred to as method engineering (see Sections 1.4 and 3.3). A person with this knowledge and the responsibility for maintaining a systems engineering method is often referred to as a method engineer. Furthermore, we have
a related role in the method constructor who has originally developed the method. We have chosen to define the concept method engineer as below.

**Definition 1-12:** The method engineer is a role responsible for the maintenance of the systems engineering method.

Such competence might already exist in-house, it can be recruited (will exist in-house) or can be hired. Considering these options, we can also conclude that there is a difference between organization specific and general knowledge about systems engineering. This differentiation in knowledge affects the possibility to externalize the existing way of working into a method description. An in-house method engineer or method constructor is more familiar with the organization, the context in which the systems engineering method exists or is going to exist in. Hence, he or she possesses general knowledge about methods, as well as organizational knowledge. The later is significant because systems engineering methods can exist in different realms (Goldkuhl, 2002), which is discussed further in Section 2.1.2. Method engineering knowledge obtained from outside the organization lack the situational part, at least in the beginning. The purpose of recruiting a method engineer is to create in-house knowledge, which is not the case with hiring (in its pure meaning; of course these strategies can be combined).

An alternative approach to in-house development of a systems engineering method is obtaining a method from outside the organization and with the aim of implementing it in the organization. Returning to our oranges, we could have asked other groups participating in the same exercise about the task and copy their procedure for solving it². Thus, we would be focusing on the method as an artifact instead of the knowledge associated with a method engineer.

These systems engineering methods cover the software development process to different extents. Some of them have a narrow scope. VIBA³ is an instance of such a systems engineering method. It focuses on requirements engineering and aims at emphasizing features that make a system actable (Ågerfalk 2001). However, good a systems engineering method is at requirements engineering, it only supports a delimited part of the development process (Andersen, 1994) and we might need complements. Otherwise, we are not going to be able to give proper support to the complete development process⁴. VIBA, for example, lacks support for test procedures and how to deploy the software to the market. As a consequence, projects relying only on a systems engineering method that covers a limited part of the development process could have shortcomings in quality control. Hence, this

---

² In this case we have to make the assumption that there actually had been at least one group who had solved the problem successfully before us.

³ Versatile Information and Business requirements Analysis

⁴ The software development process needs at least implementation in order to deliver an executable information system. Of course, we could consider a standard system but it still needs to be implemented in the organization.
could mean that work has to be performed in an ad-hoc manner in the unsupported parts, which would add to the project’s risk list\(^5\).

Currently, there is popularity in rigorous ‘off-the-shelf’ methods, which are standard systems engineering methods. These methods often constitute the other extreme of systems engineering methods, covering all phases in the development process. We have chosen to define a standard systems engineering method as below:

**Definition 1-13:** A standard systems engineering method is a systems engineering method that has its origin outside the target organization and is used by more than one organization, and the development of which the target organization does not control\(^6\).

Hence, implementing a standard systems engineering method means adapting to a way of working that does not originate in your organization. Therefore, your organization has to adapt to this method in a way that is similar to implementing a standard system (Brandt et al., 1998).

One prominent example of such a systems engineering method is Rational Unified Process (RUP) (Kruchten, 1999). In order to cover all phases in the software development process, and a wide range of project-types, these methods need to be of an impressive size. For example, RUP gives support for performing business modeling, requirement engineering, and testing. Of course this is impressive. However, both the amount of prescribed actions to be performed and the artifacts to be produced during a project, in order to deliver software, are also impressive.

One argument for choosing a standard systems engineering method instead of in-house development is the impact that methods have on organizations. They always involve changes. Externalizing a method means dealing with variations found in existing but implicit procedures. Choices have to be made and these decisions could be colored by the organization’s internal balance of power. Standard systems engineering methods might, in such cases, be considered to be a neutral option.

Recapturing Definition 1-12 the area of responsibility for the method engineer focuses on maintenance. A systems engineering method needs support and maintenance during its life cycle in the organization. Since methods exist in different realms (Goldkuhl, 2002), there is an important distinction to make between method descriptions and internalized systems engineering methods. It is the latter that is used by project members during their work. This means that the method engineer needs to support these actors with the internalization process. One central part in internalization is education. Implementing methods in large organizations naturally means increasing needs for education in the systems engineering method and its tools, education that should be independent of where in the world different

---

\(^5\) This is based on the assumption that systems engineering methods add value to the development process.

\(^6\) Unless the target organization is not the development organization.
parts of the organization are located. Thus, it can be difficult to cover a scattered organization with in-house competence. Furthermore, a systems engineering method is not static. Improvements have to be elicited and incorporated into the different realms where methods can exist. Hence, we have a continuous need for method engineering knowledge.

Systems engineering methods are hard to use without proper tools. A method’s prescribed actions aim, in one way or another, to create or modify artifacts. This tends to be hard and time-consuming work since it is difficult to maintain consistency between artifacts. With proper tool support such things can partly be automated, or at least made easier. Therefore, we have a need to either develop tools that support the in-house method or try to use techniques that are supported by existing tools bought on the market.

In contrast, a standard systems engineering method opens the door to a world of ‘ready-to-use’ method frameworks. These frameworks are based on experience gathered in other organizations that has been tested in a wide range of cases. Hence, the activities in these methods ought to have been proven useful. For example, RUP is based on what Rational calls the ‘six best practices’ (Kruchten, 1999). The method is continuously being improved with newer versions, which reduces maintenance. Often there are tools developed together with these standard systems engineering methods and they are updated to be compatible with the current version of the method. This reduces the need for in-house competence but does not eliminate it. Nevertheless, we have to be able to support projects using the chosen method and, therefore, we need specific method competence.

However, standard systems engineering methods also have their downsides. We find parallels when reading Brandt et al. (1998) about implementation of standard systems. For example, one risk is supplier dependency. Such a situation could arise with standard systems engineering methods. You do not own the development of the systems engineering method; you have traded away that part. Of course, you do have the freedom to adapt the systems engineering method, however, we refer to the owner concept as the organization controlling the release of official base versions of the standard systems engineering method. Hence, it is hard to influence the content of the next version of the method and its tools. When choosing local adjustments of the standard systems engineering method there is a need to consider how these parts are to be maintained in further versions of the method. Every adaptation has to be anchored in the standard systems engineering method. It is not possible to control the pace of new releases of a standard systems engineering method. As the client company, you are exposed to new releases and the ‘pressure’ to implement them. Furthermore, the advantage of tools proper for the current version of the standard systems engineering method is not free. Investments in

---

7 These methods can be anything from a framework according to Goldkuhl’s definition (1991) to complete method descriptions. Hence there is somewhat a diverse world we are approaching in this discussion.

8 We are aware that one can find license agreements restricting possible tailoring of the standard system engineering method. For example, in many cases it is cogent to explicitly show which method the adapted version is based on.
Introduction

licensed tools induce cost complying with some pattern, for example, yearly fees or for new releases.

1.3 Situational Systems Engineering Methods

The need for method differs depending on the project’s context. In general, there is no one way mapping between needs and how they can be satisfied. Even if we had had an ‘orange-gathering-method’ it is not obvious that it would have been suitable to use during our exercise. We would, for example, have to take into account whether or not we were wearing blindfolds, the size and shape of the environment (which in our opening discussion was constituted by a room), and the project team as such. Facing a larger environment than our initial six by six meters could call for a way of working where the environment is divided into larger sections, which are searched one after another by the complete team. Hence, the way of working would diverge slightly from the one sketched in the end of Section 1.1. In contrast, a project might involve gathering pineapples instead of oranges, while everything else is similar to our initial case. In such a case, the goal differs slightly but the environment is similar and the method could be well suited for solving the problems at hand.

As with the oranges, the need for systems engineering methods differs depending on the type of system engineering project we are facing (see van Slooten and Hodes, 1996; Rolland and Prakash, 1996), and in Brooks’ (1995) words there is ‘no silver bullet.’ Consequently, systems engineering methods have to be made situational and we define a situational systems engineering method as below:

**Definition 1-14:** A Situational Systems Engineering Method is a systems engineering method tailored for a specific project.

Whether or not we discuss standard systems engineering methods does not make a difference. Ideal methods become general methods for general problem situations that do not exist. There are few projects that need all the prescribed actions, artifacts and roles defined in rigorous standard engineering methods. Something that holds true for RUP, which is ‘a generic business process for object-oriented software engineering’ (RUP 2001 documentation). In some situation rigorous standard systems engineering methods could perform even worse than smaller methods since they are extreme in their coverage of the development process. If these methods are applied ‘as is’ on specific projects there is a risk that the method will become a burden instead of the intended support. Team members will become more focused on how to use the systems engineering method in the correct way and the problem they are trying to solve will be given second priority. This is the wrong way to work (Röstlinger and Goldkuhl, 1996), as we would end up with a situation opposite to the one we had in our introductory exercise. In the orange example, we were so focused on the goal that we ignored the need for a method.
1.4 Method Engineering

The procedures for finding, analyzing and combining complements into a general or situational method are referred to as method engineering. According to Harmsen (1997), method engineering is defined as ‘the systematic analysis, comparison, and construction of Information Systems Engineering Methods.’ Situational systems engineering methods are built based on method fragments, which are suitable for the project at hand. Selection, and more specifically, combination, of method fragments are not that straightforward. Method, and hence method fragments, differ in perspective, notation, structure, and concepts (Nilsson, 1994, Goldkuhl, 1991). Thus, combinations of method fragments have to be done with care and through structural analysis. For example, Harmsen (1997), has provided a framework for performing such an analysis and selecting method fragments that fit together.

An important decision when implementing a standard systems engineering method is the possibility of creating a shared cognitive base within the organization. Building a systems engineering method on a modular method construction approach (Odell, 1996) could result in fundamentally different methods for each project. This means that it is difficult to build a core competence around some stable parts of a method. Of course, it could be argued that a business has a main stream of projects and that the systems engineering method should not differ to a large extent for these projects. Nevertheless, why start on the bottom layer in the method fragment hierarchy in these situations? It would be more appropriate to take the constructed systems engineering method as the point of departure and adjust it for the specific project. One could also argue that method competence is not only the team members’ ability to master techniques in specific method fragments. That holds true, but still all team members are not meant to become method engineers (Kruchten, 1999). Diversification tends to be confusing and does not facilitate method as support.

Furthermore, recapturing the definition of systems engineering methods we can conclude that there are several possible choices of focal areas when performing method engineering and turning an ideal method into a situational one. Harmsen’s (1997), as well as Brinkkemper’s (1996), focal areas are concepts, prescribed actions and artifacts with regard to the classes presented in Figure 1. They also deal with tools, however these are not included in our definition of a systems engineering method. Since prescribed actions are part of their focal areas they also deal with the ‘part of’ complexity. In Rolland and Prakash. (1996), we find an extension of Harmsen’s thoughts for using modular construction; adding the context to these constructs. Reading Hares (1992), we find a focus on different types of contexts and prescribed actions in order to construct paths in a systems engineering method. Opposite to modular construction of situational systems engineering methods, is the choice of the complete method without any adjustments (Odell, 1996). Hence, a systems engineering method and its attributes, in such a case, become the focal area.

---

9 Referring to Brinkkemper et al’s (1999) layer of granularity we use the method layer.
10 Kruchten (1999) actually uses the term process engineer. However, we have chosen the concept method engineer on the basis of our discussion in Section 3.1.2.
Introduction

However, there is often an absence of role (actor), goals and purpose (rationale) as focal areas.

1.5 Method Configuration Based on RUP

RUP is structured in two dimensions. On the horizontal axis, in Figure 2, the structure follows a time pattern as a project evolves. According to this pattern, there is a transition through four different phases: Inception, Elaboration, Construction, and Transition. This dynamic aspect is intertwined with the vertical dimension based on the method’s content (Kruchten, 1999). The content is organized in disciplines where the first six disciplines are process disciplines and the last three are supporting disciplines. Together these give an iterative pattern and each discipline is performed a number of times for each phase, depending on how many structured iterations that are needed.

The Environment discipline is partly devoted to method configuration\textsuperscript{11}. The purpose of this discipline is to support the specific project with a tailored method. The adaptation of the method can, according to RUP documentation, be performed on two different levels. First, the organization-wide process is an adaptation of the method to the specific organization and its application domains. Prescribed actions

\textsuperscript{11} Rational uses the concept process configuration instead of method configuration. However, based on our discussion in Chapter 3.1.2 we have chosen the method concept instead.

Figure 2. RUP Overview (RUP 2001 documentation)
and artifacts are deprecated and new organization specifics might be added. Hence, the final artifact from this adaptation is a cross between RUP ‘as-is’ and a project version of the systems engineering method. Second, the organization-wide process is refined into a project-specific method where the size of the project and the available resources are taken into consideration. Both types of configurations are documented in Development Cases. (ibid.) Thus, the Development Case is a context specific description of RUP. Rational Software defines a Development Case as follows:

**Definition 1-15:** ‘The development-case description describes the development process that you have chosen to follow in your project.’(RUP 2001, online-documentation)

The actual configuration of RUP into a situational method is performed in the ‘Develop Development Case’ activity found in the Environment workflow. The activity is described in a number of steps. Briefly, it is about settling the scope of the Development Case, deciding how each discipline should be performed, mapping workers to job positions and, finally, documenting the Development Case. The Process Engineer, a worker defined in RUP, is responsible for tailoring the method and delivering a Development Case. To support this activity, guidelines for the Development Case and Process Discriminates are found in RUP.

The adaptation of RUP involves considering the tentative prescribed actions to be performed, resulting artifacts to be produced and tools to be used (Strand, 2001). Each part of RUP can be modified, added or suppressed. The reasons for doing so differ. The resulting artifacts depend highly on the project’s needs. A project without a database has no need for a data model. Of course, this would affect activities in which this artifact is a prerequisite or an output. Steps involving an artifact that is no longer necessary should be suppressed. However, this is only a description of what can be configured and not how the configuration should be carried out.

Returning to the different focal areas we discussed earlier (see Section 1.4) the guidelines found in RUP can be analyzed. In general, one can conclude that the guideline focuses on prescribed actions, artifacts, roles, tools and the context. Attributes identified in the context should affect RUP as it is presented in the Development Case. This is done by examining the current organization (a specific part of the context) in order to discover weak, or strong parts in the existing way of working and introducing prescribed actions to support the weaker parts. Decisions should be made on which artifacts to use and to what extent (which could mean tailoring templates for the artifacts). Furthermore, stakeholders are identified which correspond to identifying roles used in the method. Finally, as in Harmsen (1997), there is a focus on tools to use, which in our case, is something found outside the demarcation of a systems engineering method. However, we find a weak and inconsistent use of the systems engineering method’s rationality when considering the different aspect of the context: Why should a prescribed action be performed in a certain way? Of course, it would be advantageous to have several focal areas (for example, artifacts and prescribed actions) to be used as a screen for the context, but then the switch between these focal areas has to be explicitly described. However,
Introduction

this would also increase the complexity when performing the configuration task. For this reason, a vague description could lead to unsystematic procedures.

1.6 Experiences of RUP Configuration

Configuring RUP tends to be difficult and time-consuming. After being part of a project team (having the role as method engineer) tailoring RUP for development of IBSAs some experiences can be summarized. Method configuration involves analysis of the method at hand, and as with every type of analysis it presents possibilities to focus on different aspects of the phenomenon; in this case of the systems engineering method.

Our experiences are discussed in the light of our method definition (see Definition 1-1 and Figure 1), which gives us a point of departure for possible focal areas together with the focal areas proposed in RUP. This episode gave the project team the possibility to apply the focal areas found in RUP, and evaluate if the criticism forwarded in the previous section affected the configuration work. Consequently, from that starting point we can sort out focal areas, which have not worked well and focal areas we find important to go on with in this thesis. Both problems and the selection of focal areas are based on the values we find important during method configuration (see Chapter 4 for detailed discussion).

The adaptation of RUP\textsuperscript{12} was done on both an organizational wide as well as project specific level, as described above. The intention from the start of this project was to generate a project specific Development Case, based on Volvo IT’s organizational wide Development Case. Despite the descriptions in RUP, activity descriptions and guidelines, it was difficult to get focused on relevant prescribed actions to perform. Subsequently, there was a problem in using prescribed actions as the focal area during the analysis.

The arguments behind the decisions for modifying or suppressing a prescribed action on the organization-wide level became important. Thus, during the configuration we tried to use rationality as a focal area, despite that it is not proposed in RUP. However, these arguments were not documented in an organized way. One reason for this problem could be the lack of an artifact facilitating this focal area. Consequently, in the aim to make the organization-wide Development Case reusable, the lack of rationale became an obstacle since rationality could not be used as a focal area. When an organization-wide Development Case is taken as the point of departure for creating a project specific Development Case it is important to know why the process has been modified. Otherwise, the same procedure tends to repeat all over again where each and every prescribed action must be discussed and this is not reuse.

The degree of granularity in the analysis is important, and focuses on the ‘part of’ association between different prescribed actions in Figure 1. The resulting artifact from a RUP configuration, a Development Case, was documented using

\textsuperscript{12} Since RUP is the chosen standard system engineering method it is not useful to use system engineering method as a focal area. We were more interested in the method’s content.
Microsoft Word. In RUP, we have the potential to work with the process on five different layers (see Table 1). These levels range from the method as such and the question whether or not we should use RUP, to the granularity of steps. Between these two extremes we find the levels Discipline, Workflow Detail, and Activity.

**Table 1. Layers of Granularity in RUP**

<table>
<thead>
<tr>
<th>Layers of Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUP</td>
</tr>
<tr>
<td>Discipline</td>
</tr>
<tr>
<td>Workflow Detail</td>
</tr>
<tr>
<td>Activity</td>
</tr>
<tr>
<td>Step</td>
</tr>
</tbody>
</table>

The choice of appropriate granularity is a combination of cost and precision. Choosing the step as the unit of analysis gives high precision but also an intractable amount of data. Moving upwards through the layers we trade precision for lower cost. With lower precision, the result is less useful as support either as a starting-point for further adaptation or as guidance for a specific project. In the configurations discussed, we performed the analysis on the activity layer. The documentation were represented using the layers; disciplines, workflow details and activities. However, it was not possible to see from an aggregate level, discipline, for example, that an activity within that discipline had been changed.

Recalling the structure in Figure 1 artifacts are one possible focal area during method configuration. This has also been pointed out in RUP. Each prescribed action has at least one resulting artifact. Since we had difficulties with prescribed actions as a focal area these artifacts received a lot of attention. If an artifact was considered useful its corresponding activities had to be performed. Relevant resulting artifacts from these analyses were listed as supplementary to the activity list. However, one drawback is that artifacts are only a means to an end, and it is difficult to analyze the intention behind a produced or modified artifact. By ‘intention’ we mean the method engineer’s rationale for prescribing such an action.

The potential to reuse a Development Case was problematic, due to the layers of granularity and the lack of explicit rationality. Choosing the activity layer of granularity means modifying or suppressing activities. Nevertheless, the development project often required some part of the activity or its resulting artifacts. Therefore, modifications had to be made within the activity but such modifications were not possible to show. This resulted in very modest modifications of the process. From the perspective of reuse, the adaptation into a project specific Development Case means reconsidering each activity in the organizational-wide...
Introduction

Development Case. Since no significant changes were visible, the procedure to a large extent had standard RUP as the point of departure.

The roles described in RUP were mapped to roles and actors in the project. The task of mapping general roles in RUP to roles used at Volvo IT had been started earlier and hence made the translation between the method description and the context easier. Still problems could be found. Only a selection of roles should be used during the project and that selection is associated with the selection of prescribed actions, which in many cases was uncertain.

The configuration, either of an organizational-wide or a project specific Development Case, were highly dependent on the individuals performing the configuration. The vague description in RUP of what to focus on during method configuration, in order to give the proper support to a project, tends to make configurations ad-hoc based. Since one can conclude that the organizational-wide Development Case was not effective the final configuration greatly depended on personal experience instead of reused organizational-wide project experience. Consequently, the question arises, if we do not perform method configuration based on clearly defined focal area(s), how will we know in advance if the situational method is becoming a support or an obstacle during use? The answer is: We do not!

1.7 The Need for Method Support

It is widely understood that no generic method, for example RUP, is suitable for every project. Therefore, we must configure a generic method to suite a specific project. However, from the theoretical discussion and the experience described above we can argue for an improved understanding and support for method configuration. Despite the effort put into configuring RUP we were uncertain if the situational systems engineering method matched the project’s need. This is the initial problem statement illustrated in Figure 1 (referred to as problem number 1). We had a need for method support during the configuration activity discussed in Section 1.6. The fundamental problem target in this thesis is the design of a method that supports the configuration of rigorous standard systems engineering methods.

Based on earlier discussions, we can conclude that standard systems engineering methods should not, and are not intended, to be applied ‘as-is’ to projects (2). Such action could lead to methods being obstacles rather than support during the development process. Therefore, situational systems engineering methods are needed. Behind this statement we find a belief in methods and their potential to support, which is discussed in Section 4.1.1.

Method configuration has, in our case, been a time-consuming activity to carry out (3), especially when considering the uncertainty that still existed about the situational method’s value. Therefore, we need to elaborate a method that reduces this uncertainty. In Section 1.6 two major problems, the reusability aspects of method configuration and the lack of consistent and explicit procedures to use, have contributed to the uncertainty and the time consumed. The process description in Section 1.6 illustrates a rather ad-hoc based configuration (4), despite the proposed guidelines in RUP. The context is important to consider, both the organizational
context and the project specific one. However, it seems difficult to bridge the gap between contexts that are made explicit and how they should affect the situational method. Prescribed actions, artifacts, and roles are proposed focal areas, according to the guidelines. Nevertheless, it is not explicit how they should be used together with the context to sort out prescribed actions to perform, artifacts to produce and actors to involve. In Figure 3 we illustrate this as ‘(5) Lack of explicit focal area(s)’.

(5) Lack of explicit focal area(s) → (4) Method Configuration is ad-hoc based → (3) Method Configuration is time-consuming

(10) Lack of a practicable conceptual framework to facilitate reusability → (8) Method Configuration is not experience-based → (6) Method Configuration is not made reusable

(9) Reuse is based on the complete standard system engineering method (RUP) → (7) The rigidity in the Development Case complicates reuse

(2) Standard system engineering methods are not intended to be applied ‘as-is’

The second major flaw is the reusability aspect of our method configuration process (6). We obviously had difficulties in reusing the organizational version of the standard systems engineering method. Every project was ‘inventing’ their specific practice for configuration since they lacked a unified way to perform method configurations. This problem increased the time needed for configuration work since more time had to be spent on discussions. Furthermore, discussions also affect uncertainty. Discussions per se should not be considered as something negative, rather they can in many cases be valuable. Nevertheless, in this case a discussion arose concerning the lack of documentation of the rationale of earlier
configuration decisions. The problem with a transparent documentation of adjustments to prescribed actions is much the same. We have chosen to illustrate this as one problem in Figure 3, ‘(7) The rigidity in the Development Case complicates reuse’.

We identified additional problems for not being able to reuse earlier configurations, besides the problem with the Development Case artifact. These were ‘(8) Method Configuration is not experience-based’ and ‘(9) Reuse is based on the complete standard systems engineering method (RUP)’. In order to facilitate reuse, knowledge about different contexts has to be incorporated into the configuration process in a systematic way. Reusing experience of the context means the opportunity to improve future configurations, and consequently make a quality improvement. However, with a lack of focal area(s) incorporation of the context into the configuration was inconsistent. Furthermore, it was only shared outside the project to a lesser extent, depending on the actor(s) participating as method engineer(s). The fundamental problem behind these problems is ‘(10) Lack of a practicable conceptual framework to facilitate reusability.’

1.8 Research Question

The research question originates from the problem diagram in Figure 3 and the possible focal areas for method configuration that have been discussed above. We have concluded that a need for method support exists in the field of method configuration. Since related research (see for instance Brinkkemper, 1996; Harmsen, 1997; Röstlinger and Goldkuhl, 1996; Rolland and Prakash, 1996), in the field of method engineering focuses on situational methods built on modular construction and not adaptation of rigorous standard systems engineering methods, this method support cannot be found within that field. Therefore, the fundamental problem statement in this thesis is

How should a meta-method for the configuration of standard systems engineering methods be designed in order to support reusable configurations based on the standard systems engineering method’s rationality and context?

Behind this research question we have a design idea founded on two fundamental values: a method’s rationality and reuse (see Sections 4.1.3 and 4.1.5). Of course, it is possible to uphold other values (see for example, Röstlinger and Goldkuhl, 1996) during method engineering and an elaborated discussion can be found in Section 4.1. Since values can be conflicting some of them have to be placed in the background, it is necessary to prioritize.

Emphasizing the rationality value means that method configuration should be based on the systems engineering method’s rationality. In Figure 1 we show that prescribed actions have purposes describing what to achieve and why, something that we find highly important. Since the standard systems engineering method shares context with development projects or projects that are sub-parts of a context, we
believe that the purpose concept could be used as a systematic raster when tailoring standard systems engineering methods to different parts of the context. Thus, with this design idea we propose putting the rationality-aspect of systems engineering methods in the foreground during method configuration. This does not mean that we find other focal areas unimportant; however we do intend to use rationality as the primary focal area.

We find reuse important from several different points of view. The concept cannot be derived from Definition 1-3 as rationality can. However, the concept has an empirical foundation in the problem diagram illustrated above (see Figure 3). Facilitating reusable configurations could reduce resources spent on configuration issues, as well as improve the configurations. At the same time, we introduce somewhat a paradox: methods cannot be applied ‘as-is’, they have to be made individual, and introducing reuse means trying to use the same configuration at least twice. The latter could mean a restriction on the individuality of the situational systems engineering method.

1.9 Characterization of the Research Question

Knowledge is a concept widely discussed in literature (e.g. Molander, 1996; Rolf, 1995; Goldkuhl and Braf, 2002). We can have different aims with the development of knowledge and it can therefore be characterized in different ways. Goldkuhl (1998) presents eleven aspects, or characteristics of, knowledge. Consequently, these can be used to characterize the need for knowledge in the thesis.

The aim of the thesis is to design a meta-method based on theoretical and empirical grounds. Methods are principles for practices; methods are about ‘ought to.’ Therefore, and not surprisingly, at the bottom line we are interested in normative knowledge. Behind the need for meta-method support for this specific practice, method configuration, we have an idea how it should work. We have quality goals we want to obtain. This aspect of knowledge we refer to as ‘value’ and it is found in the research question.

We can conclude that reuse is not a clear-cut concept, thus we need to stipulate what is a reusable configuration. More specifically, we need to address the issue of how reusable configurations should be delimited. In this aspect, we are interested in ‘categorical’ knowledge, what and what is not a reusable configuration. We need to have a deeper understanding of the concept reusable configuration in order to make such a categorization. Hence, in some aspect, this research question is also about characterization of the concept. Since configuration means making decisions on which method parts to modify, add or suppress, and the configuration should be done in a unified way, we need classifications of method parts. Hence, we are interested in ‘classification’-knowledge.

1.10 Aim of Thesis

The aim of this thesis is to present and elaborate on the design of a meta-method emphasizing reusable configurations based on standard systems engineering methods’ rationality and context. By ‘present’ we refer to presenting the principles
Introduction

and structure of the method for configuration. Behind the principles and the method’s structure we have an argumentative dimension on which the method is theoretically and practically grounded, this ‘grounding’ is referred to as ‘elaborated.’ Theoretical grounding is a matter of relating the method’s concept, structure, and principles to research in the field of Method Engineering and ensuring that they are consistent within the method. Practically grounding is reflections based on experience of using the method for configurations.

1.11 Stakeholders

Every project has its stakeholders. As argued above, the roles in a development project are supported to different degrees by RUP, and for that matter by other methods as well. We want to improve support for the role method engineer (or process engineer in RUP) during projects in their work to generate and maintain the situational systems engineering method (see Figure 4 below).

Method engineer is the generic role of the process engineer in RUP, who is a stakeholder in this specific project. We use the word generic because different parts of the meta-method differ in applicability. Organizations that have implemented RUP as their software development process have the largest applicability. Still, the method for configuration applies to a wide range of methods and is of interest for those who have a role equivalent to process engineer in RUP, but are using another rigorous standard systems engineering method.

Figure 4. The Method Engineer’s Role
The contribution of the thesis is interesting for organizations searching for a way to improve their business value through the use of “off-the-shelf” methods. Volvo IT has a special interest in this project since the method for configuration is being developed together with them and more specifically in their project for implementing RUP. Rational Software, the company behind RUP, has an interest in improving the understanding of how to configure their method, and thus make RUP better fit a wide range of projects.

The audience above can be referred to as practitioners. A second target group for this thesis can be found; researchers within the method engineering field. Method configuration based on the systems engineering method’s rationality is interesting to relate to both existing and forthcoming studies on situational systems engineering methods, in order to move the knowledge forward. Moreover, we have a different point of departure than, for example, method engineering based on modular method construction, which is interesting to evaluate as well. In our case, we start with a rigorous method with the main intention of reducing its size instead of using several small methods that need to be integrated with each other.

The Swedish Knowledge Foundation and Volvo IT have financed the thesis, which makes them stakeholders. Volvo IT’s interest has been discussed above, and it differs from The Swedish Knowledge Foundation. Their interest is first and foremost to strengthen competence in Swedish institutions of higher education with a focus on the use of IT. This thesis is one step in that process.

1.12 Demarcations

The primarily concern in this thesis is method configuration. The result of this thesis will be a method specification for method configuration, and the argumentation behind it. Of course, method configuration is not an isolated activity in itself. It has its prerequisites and produces results. One important prerequisite is the need of a systems engineering method, which in our case should be qualified as a standard systems engineering method. In this thesis, we will take the existence of such a method for granted and it will not be a focal area for the proposed meta-method. Therefore, we have no intention to study why and how a method is chosen for implementation in an organization. That has to do with the characteristics of possible systems engineering methods organizational change, not method configuration. Method configuration is not limited to standard systems engineering methods only, however, we are not interested in methods other than rigorous standard systems engineering methods.

As stated above, methods often need tools to be effective in use. It is not within the scope of this thesis to elaborate tool support for the meta-method. We have two main reasons for this decision. First, we need a stable meta-method before a tool can be implemented. Limited resources demarcated this possibility. Second, dealing with a standard systems engineering method could open an arsenal of standard tools. Consequently, there may exist possibilities for complementing existing tools with the meta-method. This option is considered more interesting, and is not considered as tool development.
1.13 Thesis Outline

The thesis is split into eight chapters. In the thesis’s opening discussion, Chapter 1, we introduce the background to the research question and the thesis’ aim. Together with a characterization of the research question it is the foundation for our research design, which is presented in Chapter 2. In this chapter we discuss criteria for different grounding processes, resulting in three related research environments. Within regard to these research environments we discuss how they can be integrated through our research process design.

Perspectives on the systems engineering method are central to explaining how the values systems engineering method’s rationality and reuse can be applied. Therefore, we devote Chapters 3 and 4 to these issues. In Chapter 3 we discuss the method concept, associated concepts and different approaches to method engineering. This discussion is then used as input to the following chapter. Our theoretical framework is elaborated in Chapter 4, starting with a discussion of fundamental values for our meta-method, followed by a conceptual framework supporting systems engineering method’s rationality as the focal area, as well as the concept of reusability. This chapter rounds off with a principal abstract of our meta-method. In this chapter, we utilize both UML and XML to formalize our discussion; hence the reader is assumed to have basic knowledge of both.

Chapters 5 and 6 are devoted to the operationalization of the theoretical framework elaborated in Chapter 4. In Chapter 5 we present RUP-specific concepts and the structure of RUP in more detail. The concepts and structure are necessary to facilitate the detailed meta-method description found in Chapter 6. Chapter 6 is an externalized version of the meta-method in a RUP-shell.

The design of the meta-method has been cutout both in an academic, as well as an empirical research environment. The empirical research has, in several cases, been conducted as workshops facilitating an integrated engineering effort between researchers and practitioners. A selection of these episodes is found in Chapter 7. The aim is to present the significant design decisions, to illustrate usage and to present delimited evaluation of the meta-method. All in all, the discussion in Chapters 4 to 7 is intertwined with references to each other. This is a presentation problem, since the development process has been iterative, but we have chosen a sequential presentation. The reason for this has its origins in the aim of this thesis. One important part of the thesis is the presentation of the meta-method, and with the chosen structure Chapter 6 could be refined to contain a detail meta-method description.

We conclude the thesis with Chapter 8. It contains concluding remarks about the meta-method. Every method has its focal area, and we discuss the strengths and weaknesses of our meta-method, as well as its structure, which in turn points out future research issues.

1.14 Reading Instructions

The outline presented in the previous section can be divided into three categories depending on the intended reader. In Section 1.11 several stakeholders are
presented. Coarsely outlined, we have the categories; researchers, practitioners using a rigorous standard systems engineering method other than RUP and practitioners using RUP. Based on these categories, we can give brief reading instructions.

**Researchers in the method engineering field**
Chapters 2 to 4 are of special interest for researchers in the method engineering field. They cover both research design as well as theoretical foundations. Design decisions, meta-method testing and concluding remarks can be of interest and can be found in Chapters 7 and 8. However, Section 7.2 is closely related to RUP.

**Practitioners not using RUP**
Practitioners using a standard systems engineering method other than RUP can find relevant material in Chapter 4. The chapter is devoted to a theoretical presentation of the meta-method, and the presentation does not align to any specific standard systems engineering method. However, Chapters 5 and 6 provide ideas about how a specific method-shell could be applied to the meta-method. Section 7.3 provides a valuable example of using the meta-method in a context.

**Practitioners using RUP**
Practitioners using RUP will find the most valuable material in Chapter 6, a chapter completely devoted to a detailed presentation of the meta-method in a RUP-shell. Chapter 5 will, in many cases, be unnecessary for the competent RUP user, since RUP-specific concepts and the structure of RUP should be familiar to them. As for the former reader category, Section 7.3 provides a valuable example of using the meta-method in an RUP context.
CHAPTER TWO

2 RESEARCH DESIGN

The purpose of this chapter is to elaborate on the research design used in this thesis. The field of study and the research question have shaped this design. Consequently, we start with a brief elaboration of how the field of study affects our chosen research strategy. We have to consider what a grounded method is in order to give relevant emphasis to the different aspects a research and design process could have. Furthermore, we operationalize these thoughts into the iterative research process that has been used to develop the new meta-method. In following sections, we discuss gathering data from the research project, data analysis and our adopted attitude during these phases.

2.1 Foundations for Research Approach

2.1.1 Domain Levels

The bottom line in the thesis is prescriptive principles for method configurations, that is a method. The characterization of knowledge and the nature of our problem should be the guiding lights for the implementation of the study (Merriam, 1994). In this section, we will elaborate further on this topic.

The development of the new method, hereinafter, Method for Method Configuration or MMC for short, is method engineering. The only difference is that our new method is a meta-method, a method for application to other methods. We have chosen to define the concept as below:

**Definition 2-1**: A meta-method is a method of the second degree.

If we divide the world of system engineering into different layers depending on their main focus, then we have the first layer of the system engineering activity as such. This layer is illustrated in Figure 5 as layer 1. System engineering focuses, for example, on investigation into the need for business changes, whether or not they
Research Design

should be implemented as an information system, and furthermore how such implementation should be conducted. Consequently, the aim is often to deliver an artifact in the form of an information system. During this work, systems engineering methods are used\(^{15}\), however, they should not be in focus. If they are in focus during the development process, we risk obtaining a tool-driven, instead of a problem-driven, development process. Hence, we have chosen to place the development of systems engineering methods on the layer above the system engineering activity. Subsequently, we obtain a layer 2 added to Figure 5. This division of the domain into layers is not unique and can be found, for example, in Tolvanen and Rossi (1996). Of course, there is no strict division of these activities despite the risk of mixing them, since changes in focus between the layers during projects could add value both to the present project, as well as coming projects. However, an awareness of which layer and what task is in focus is required. One can conclude that both tasks are complex individually, and thus a combination inherits complexity as well.

\(^{15}\) Of course this is a statement that can be discussed. It is not unusual with arguments that system engineering methods are not used as frequently as one thinks or wishes (see for example Fitzgerald, 2000). However, we will postpone that discussion until Section 4.1.1.

Figure 5. Layers of Method Development

In this thesis, our main focus is on tailoring systems engineering methods, and more specifically large methods shared between organizations as standard methods. This should place our interest on layer 2. However, extending the reasoning behind
different focuses we can conclude that another layer can be added to Figure 5. Since one part of our aim is to elaborate a meta-method for adapting systems engineering methods, we could apply a recursion of the reasoning above. The development of the tailoring process is a different focus than the tailoring itself. Therefore, layer 3 focuses on method engineering of method engineering, or as we will address it in this thesis, method engineering of method configuration.

The layers are not independent; rather they are highly dependent on each other. As illustrated in Figure 5 lower layers have artifacts from higher levels as prerequisites. For example, the situational standard systems engineering method is a result of method configuration and a prerequisite for system engineering. Hence, these artifacts are placed between the layers in Figure 5 in order to illustrate this condition.

2.1.2 Methods as Knowledge

Still the principles of how methods can exist, as knowledge, apply, despite the focus on meta-method. However, when reading method engineering literature we have to distinguish between how methods can be decomposed and how methods can exist. For example, Tolvanen (1998) discusses the former but we postpone that issue to the next chapter, putting more emphasis on how and where a method can exist. Goldkuhl (1999) claims that methods exist as action knowledge on five different levels (1) the subjective level, (2) the inter-subjective level, (3) the linguistic level, (4) the action level, and (5) the consequence level. Methods at the subjective level are an individual’s pragmatic knowledge. The second level is shared knowledge between several individuals. At the third level, the method is externalized into guidelines. Following this pattern the use of the guidelines from the linguistic level should constitute the base for the action level, and at the consequence level, the use of methods results in the materialized effects of those actions. However, this seem strange, since actions are taken based on knowledge, which leads us back to the first and second levels. Subsequently, we interpret the linguistic level as being more useful during the internalization of the method; it is one way of externalizing the method for communication.

Ågerfalk (2001) has criticized the use of levels, finding the distinction between them difficult. As above, we acknowledge this as a problem, since the levels seem to be based on each other. For example, during the development process of our meta-method it is hard to state that the meta-method has to be made into inter-subjective knowledge before we can make it linguistic. Expressing knowledge in linguistic form could be one step towards making the knowledge inter-subjective and it is when we have made the method inter-subjective that we can use it as a team. Another example could be picked from the thesis’s opening discussion about oranges. The fictive project team would have had insignificant value from the knowledge of the use of tape, if it had not been made inter-subjective. However, in order to do make it inter-subjective, the knowledge would have had to pass the linguistic level since the linguistic level was the only communication channel.
Research Design

Subsequently, there is a problem with specifying a given order in which methods evolve.

The concept of level and the importance of choosing a level can be too strictly interpreted. When reading Goldkuhl (2002), we find the problem reduced by means of an elaboration on the existence of phenomena. The distinct levels are exchanged for a more balanced picture with the ability to explain the existence of, what Goldkuhl (ibid.) refers to as, multi-existing phenomena better. This makes it easier to understand that a method can exist in different realms at the same time, and that each realm can improve the others. For example, a method could exist as a linguistic expression, as well as through action. Reflections on consequences from actions could improve intra-subjective, inter-subjective knowledge, and subsequently future action.

The distinction of different realms for the existence of phenomena, and hence in a more generic consideration, knowledge about phenomena, shows that the aim is to make methods inter-subjective knowledge. It is then, and only then, we can consider whether the meta-method is valuable for a project team as a tool. Therefore, irrespective of the concepts level, aspects or realms, our main interest is how to improve the possibility of making a method inter-subjective knowledge. We find, both in Goldkuhl (2002) and Ågerfalk (2001), that there are three options to make the knowledge explicit outside the human inner world, that is making it observable. First, we could use the possibility to externalize the method linguistically or build artifacts containing the method\(^{16}\). Second, we could perform actions and argue for meaningfulness of these actions. Finally, the consequences of method usage could be used to improve inter-subjective knowledge. Through consequences it is possible to reconstruct ways of working to a certain extent, but more important, we can compare consequences with intended results and results from different ways of working. All in all, this adds to the possibility of improved inter-subjective knowledge.

Summarizing this discussion we can conclude that the initially used linguistic, action and consequence levels are tools for improving inter-subjective knowledge about our meta-method. Furthermore, these tools have affected the thesis in two different ways. First, it affected the research process since we had to reach a critical mass of inter-subjective knowledge in the project team. Consequently, we used descriptions, actions and consequences of actions as to improve the meta-method. Second, this has affected the presentation of both the research process, as well as the meta-method itself. With the thesis’s aim to present a meta-method for configuration of standard systems engineering methods, the method can be externalized through the three tools discussed. Thus, the meta-method description should be complemented with episodes of action and consequences.

\(^{16}\) However, it risks becoming implicit since a tool can hide many considerations and assumptions made during constructions. For example, as a user of a word processor you often do not pay attention to the assumptions made about the concept document, and the constraints it imposes on possible actions.
2.1.3 Grounding The Meta-method

The linguistic, action and consequential aspects of a method reveal the importance of a grounded method since grounding imposes a validation of the developed meta-method, and thus of knowledge. Grounding a method can be performed through three processes: internal, external theoretical, and empirical grounding (Goldkuhl, 1999). Each of these differs in focus. Internal grounding has its focus on the concepts’ completeness and how consistent they are. Consequently, the linguistic aspect of method becomes important, making it possible to focus on explicitly used concepts. External theoretical grounding is about relating to knowledge within the same domain, and subsequently imposes a focus on method engineering literature. Yet again, through the linguistic aspects of methods, it is possible to ground in terms of the method’s theoretical foundations. We can relate the content and assumptions of different meta-methods to each other, and thus making them more precise in their similarities, as well as differences. The third type of grounding, the empirical, emphasizes the importance of the application of the meta-method in its intended business context. Application is used in a broad sense, involving analysis, design and implementation, as well as test and evaluation in that context. The argument is that users’ participation during the development process of the meta-method is crucial.

None of these processes can be performed in total isolation. Therefore, we are applying a hermeneutic research process, letting the results from one grounding process affect work done in the other processes. Continuously improving MMC means switching between these sub-processes, and hence is one reason behind our overall research strategy that is elaborated more in detail below.

2.2 Research Approach

The previous sections are aimed at setting the scene for the research approach. It was towards these basic foundations we balanced needs with practical demarcations. In this section, we will be more precise about these decisions and how our research scene was set. Basically, we could turn to either a qualitative or a quantitative approach. Selecting a qualitative approach in favor of the quantitative one means focusing on concepts and their structure rather than occurrence instances. If we recapture our research question and aim we inherit restrictions on what is to be considered as an appropriate research approach. We have a need for ‘categorical’- as well as ‘classification’ knowledge in order to operationalize our focal area, systems engineering methods’ rationality, and facilitate reusability. Consequently, the research approach has to involve an empirical design process in order to reach prescriptive principles useful in a business context. This is rather natural, since methods are social constructs. Furthermore, methods have consequences when used, which are better captured through a qualitative than a quantitative approach. In other words, to properly ground MMC empirically, the vicinity to the social reality of the research participants was an important aspect during this study.
2.2.1 The Research Environment: RUP

The standard system engineering used as the unit of analysis during the design of MMC was RUP. When considering the setup of our research environment, the choice was important but not a self-evident one. Considering our research question and its origin, we needed a rigorous standard systems engineering method. RUP is an extensive method if we consider the number of prescribed actions that are possible to perform. Often its size is referred to as one of the method’s drawbacks, it is sometimes considered as ungainly large. RUP is a standard systems engineering method that receives much attention today. It has had significant penetration on the Swedish market for standard systems engineering methods. Many system engineering businesses use some part of Rational Software’s large range of tools that are built to complement RUP. Hence, it is in many cases a natural choice to implement RUP as the systems engineering method to use, and it has become close to a de facto standard in large businesses. Thus, RUP is a standard systems engineering method that is used in a large range of software projects in Sweden.

Certainly, we can find other systems engineering methods, which are as large as RUP. There are many systems engineering methods with significant sizes. However, several are developed in-house and consequently exist in a unique context. With regard to our research question, these methods fall short of the standard issue. Examining the market for rigorous standard systems engineering method reveals some choices, for example, Dynamic System Development Method (Stapleton, 1997). Compared to RUP it is a standard systems engineering method with a rather small market share in Sweden. Hence, it would restrict our possible choices of business contexts.

2.2.2 The Research Environment: Business Contexts

A qualitative approach is only a broad demarcation of possible research approaches, which is obvious when considering the research framework presented by Braa and Vidgen (1999). Their framework for the organizational laboratory consists of three points represented in a triangle. Each point is an ideal type of research outcomes and they are derived from positivism, interpretivism and intervention. Positivism, on the one hand, focuses on the reduction of problems in order to gain a prediction. An interpretive approach, on the other hand, aims at understanding the phenomenon when it occurs in the context. Hence, reduction is not appropriate. Finally, research approaches could involve intervention with a desire to improve the problem situation. Since these are ideal types, hybrid approaches are found which align more or less with these three points. However, the qualitative approach only states a priority towards the axis change-understanding rather than prediction.

---

17 The company, Rational Software is conducting work in order to facilitate a more agile version of RUP through the tool Process Workbench. When our research was conducted, this tool could not be found on the market.
Empirical grounding of the meta-method is essential for the validation of the knowledge developed. Thus, we need to cover both the action and consequential aspects of methods to fulfill this quality assurance. Action in this case will involve changes in the way of working with method configuration, hence it involves intervention. Furthermore, methods as such, aim at change when they are developed and introduced, a change in the way of working. Thus, in this sense an action research approach (Benbasat, 1987; Mumford, 2001; McKay and Marshall, 2001) could preferably be considered. However, we are not only interested in the elaboration of the design rather we want an understanding of its value for the configuration work. Hence, this leads us in the direction of interpretation and case study (Braa and Vidgen, 1999). According to Merriam (1988), a case study is characterized by its focus on one type of situation, the possibility to make descriptions, and that it stimulates an iterative learning process. Braa and Vidgen propose action cases that balance understanding and change.

Our choice of research strategy fell on interaction research (Koch et al., 2001) as a modification of action research using action cases. We interpret action research as an emphasis on the elaboration of theses in an academic research environment, which are later tested in an empirical environment. Interaction research captures the possibility to involve competent practitioners in the design efforts and the elaboration of our meta-method, thus facilitating an improvement of the empirical grounding. Nevertheless, action research literature has inspired our iterative research process design in Section 2.2.3. The principle structure of our research environment is illustrated in Figure 6 below.

![Figure 6. Structure of The Research Environment](image-url)
Research Design

engineering. Although the author has had the main responsibility for the
development effort other researchers have participated. Altogether, the research
team has consisted of four researchers from the VITS research group. In this team,
we have expounded the draft ideas originating from the other environments,
considering possible inputs and integration with existing knowledge found in the
field of method engineering. One example of the work performed in this
environment is the final meta-method description. It has been carve out of this
environment by the author.

The empirical research environment is divided in two parts. We make a
distinction between the development and testing of MMC. The main reason for this
is to distinguish method-utilization, which normally is not considered as method
engineering. Furthermore, these two main research environments can be divided into
smaller business contexts and mapped to one or more collaboration partner, who
could participate in the meta-method project.

Reading Yin (1994), we have a choice between a single-case and a multiple-
case design. Several action studies, either in sequence, parallel or a combination
thereof could, in an ideal world, be an advantage, thus improving empirical
grounding. As Yin (ibid) states, such a study is often regarded as more robust. In
contrast, we have not taken into account the demarcations that each project has.
Resources are one of the more essential restrictions on a project of any type. It is
unwise to spread limited resources over a wide range of studies. We have to
consider that each business has its own structure and standard systems engineering
method that we would have to master and none of them are simple. This is an
argument for reducing the range of action cases to perform.

The obvious drawback from such a choice is the possibility of analytic
generalisability. Performing a single case means capturing no more than a fragment
of the different business contexts that the meta-method could exist in. Recalling the
thesis’ aim, our main focus should be on the elaboration meta-method’s design. Of
course, as discussed above, that design has to be made reliable, a requirement that is
met by combining the three grounding processes. Nevertheless, a different focus
could be used in the study, affecting the chosen combination. Furthermore, in our
case we can emphasize the empirical research environments ‘Method Engineering
With Qualified Practitioners’ and ‘Method Testing’ differently. However, the testing
efforts are highly dependent on the test object and its maturity. Spending vast
resources on testing an immature design is in many cases a waste, since many basic
flaws can be sorted out through rather small tests.

Initially, much effort had to be put into considering the requirements of the
project before beginning to sketch a new meta-method. This work is allocated to the
‘Method Engineering With Qualified Practitioners’ research environment. Hence,
combining it with extensive testing in different business contexts is not efficient use
of the project’s limited resources. In addition, we have to consider the opportunity to
access each business context and the fact that each case study imposes a complex
context. We believe that an externalized prototype of MMC has its importance,
facilitating for other method engineers to apply it in their specific business contexts.
Of course, the prerequisite is that the prototype has reached a certain degree of
stability. From the perspective of a meta-method prototype, the design elaboration
aligns with an exploratory study. Benbasat et al. (1987), suggest that single case
designs could be useful in explorative studies.

In the choice between balancing depth and breadth in our study, the choice fell
on giving priority to depth. This thesis is empirically grounded in two action studies
conducted at Volvo IT and the Department of Informatics (ESI) at Örebro
University. Hence, it is not a strict single-case design. Furthermore, these studies
have not had equal magnitude on MMC. The ESI case has been used as a test
environment for the meta-method, while Volvo IT has been the collaboration partner
in both empirical research environments. Consequently, the ESI case has a later
position in time than the Volvo IT case (see Figure 7. Pre-case is the initial work
performed at Volvo IT discussed in Section 1.6). The choice of collaboration
partners was an important decision, since we searched for an empirical environment
facilitating interaction research.

Figure 7. Time span for the action cases

2.2.2.1 Volvo IT
The choice of Volvo IT was based on several criteria. First, our choice of standard
systems engineering method was of crucial importance. Since we had chosen RUP
as the unit of analysis with regard to standard systems engineering methods, our
collaboration partners had to use it. Volvo IT is using RUP and is continuously
improving its well-advanced implementation. Furthermore, Volvo IT has an existing
team, MAPS (Method And Process for System development) for implementing and
tailoring RUP. Thus, an existing infrastructure made the research project easier to
perform. An important aspect to consider is the initial work performed at Volvo IT
and MAPS, which resulted in the thesis’ research question. Furthermore, in this
infrastructure, we can find a connection with different system engineering projects,
as well as connections to the company Rational Software Nordic AB. Thus, Volvo
IT gave us the opportunity to collaborate with consultants at Rational Software,
which improved the notion of competent practitioners both regarding systems
engineering methods in general, but more important, regarding RUP.
Volvo IT has a long tradition of using systems engineering methods, and hence has highly competent method users. Before the implementation of RUP, they used an in-house developed systems engineering method, which dated back to the 1980’s. Volvo IT is a large multinational business with a wide range of system engineering projects. Volvo IT is represented in every part of the world, and has a complex pattern of software platforms to support. Hence, the company resembles the target group for rigorous standard systems engineering methods in many senses. Their objectives with introducing RUP as the systems engineering method was an increased project success rate, shorter lead times, improved product fit to actual needs, and better product maintainability. These objectives should be reached through a standardized systems engineering method for application development, incremental deliveries, controlled iterative development, training programs and clearly identified roles (Grahn and Karlsson, 2002). Hence, tailoring the systems engineering method was considered an important issue.

2.2.2.2 Örebro University - ESI

The second action study was performed at the Department of Informatics (ESI) at Örebro University. Compared to the Volvo IT case, this study began in the late part of the research project, see Figure 7. The choice of ESI as a sub-part of the research environment ‘Method Testing’ was based on the following line of reasoning. Empirically grounding a method should involve empirical testing. Depending on the test design, such a research environment tends to become complex (see Section 2.2.3.3 for further discussion). We had limited resources to conduct a system engineering project in order to test MMC. Hence, we needed a project with a demarcated scope that would be conducted within a rather short timeframe. Such a project was not possible to find at Volvo IT at that moment in time. However, a small personnel system with a web user interface was planed at ESI with the author as project manager. Thus, it was an easy-to-access research environment of adequate scope where the author could take the role as method engineer. The project team involved four project members and was performed over three calendar months. The members of the project team had their background in the Department of Informatics, and thus had a good understanding of system engineering. The choice of standard systems engineering method was RUP. It was a natural choice if we consider the unit of analysis at Volvo IT. However, with regard to development work at ESI, RUP was used for the first time. On the one hand, this is a drawback of this specific choice of research environment. On the other hand, we had the advantage of following a complete project and being able to test the meta-method in a new business context.

2.2.3 Research Process

One problem with literature about action and interaction research, as well as action cases and case studies (McKay and Marshall, 2001; Merriam, 1994), is their vague guidance in elaborating a tailored research process, with regard to the research environment at hand. The guidance found in the discussion about grounding is on a
rather general level, as well. The similarities found in these discussions emphasize a cyclical approach. In our discussion of grounding methods, we identified a need for exchanging the focus between the three processes internal, external theoretical and empirical grounding. McKay and Marshall (2001), summarize the patterns in action research concluding that they are iterative. Action studies also impose iterations for the gradual improvement of understanding. Furthermore, we have searched for inspiration in the field of system engineering itself, in how to approach problem situations with a vaguely defined final product. One can argue that we knew from the start that the goal was a meta-method. Nevertheless, its shape was not known and it was not possible to set the requirements once and for all. In summary, we can argue for a research process with an iterative pattern; crafting the design of the meta-method during each iteration.

However, McKay and Marshall (2001) claim that action research involves two types of iterative patterns that are overlaid each other. The first pattern is referred to as the researcher’s problem solving interest and the second as the research interest. Hence, we have two patterns of interest. These patterns diverge on some points and McKay and Marshall (ibid.) propose the use of separate research methods for each pattern. Basically, this means one method covers the research interest and a second one covers the researcher’s problem solving interest. Aligning this affected our research approach. The iterative approach, as such, fit our needs to gradually improve our understanding of the specific context at Volvo IT, both the needs at Volvo IT, as such, and the structure of RUP. On the downside, with such an approach, we find the overwhelming amount of data and the risk of solving problems that are outside the intended demarcation.

The entire research process is a combination of the systems engineering method RUP and activities elicited from grounding methods (discussed in Section 2.1.3). The reason for using elements from RUP is rather straightforward. Volvo IT is one of our empirical sub-research environments and they use RUP as the standard method for development projects. One advantage of RUP is its iterative pattern of working with small increments, which is important with a vaguely defined target. However, RUP is not without criticism. Hesse (2001) discusses critical arguments concerning RUP from the point of view of applying it in system engineering projects. Some of these arguments are relevant to our situation. Hesse criticizes RUP for centering on phase iterations instead of ‘(sub-) product development cycles.’ With modern component-based software artifacts, we acknowledge the need such development cycles, which could involve a specific situational method for each component of magnitude. However, in our case we have to treat MMC as one coherent part, which reduce this need.

Still, RUP is an architecture system driven engineering method and hence focuses on technical solutions, which had to be suppressed\(^\text{18}\). Much focus was spent on the business modeling discipline since the aim was business change, but without implementation of an information system. In the later parts of RUP’s six core

\(^\text{18}\) Actually a situational version of RUP was used were many of the technical implementation and deployment issues were suppressed.
disciplines a technical implementation is implicit, hence these prescribed actions add insignificant value to the research process. For example prescribed actions about testing focus testing of information systems using Rational Software’s automated tools. Quite natural this is a limitation in RUP as a method for the research process, since automated testing of a meta-method is hard to achieve. Hence when elaborating on test procedures for the meta-method we have to turn elsewhere. Furthermore, RUP does not discuss the need to perform theoretical grounding of the final artifact, which we need in order to assure quality as researchers.

Figure 8 illustrates the research process used in this thesis together with the three research environments discussed earlier. This illustration is a further development of Figure 6, now with the research process’ activities incorporated. The activities are grouped into blocks when they have been performed, mostly, at the same time and are hard to distinguish from each other. All in all, the research process is divided into four major blocks of activities: ‘Theoretical Grounding,’ ‘Elaboration,’ ‘Design’ and ‘Testing.’ In Figure 8 these blocks are mapped into the research environments. We use arrows to represent the flow of activities, which were performed through the elaboration of artifacts. However, these artifacts are not included in Figure 8 in order to make the overall picture easier to grasp. These arrows make the relationships between the research environments clearer; in Figure 6 these were only illustrated with lines.

A closer look at Figure 8 reveals that it resembles the two partly overlaid iterative patterns discussed above. An iteration pattern is found in the empirical research environments illustrated by the solid arrows. Additional iterative patterns are found that bring the academic and empirical research environment together. These are illustrated using dotted arrows, as well as the fact the activity blocks ‘Design’ and ‘Testing’ span both environments.

The former iterative pattern was conducted in the action cases at Volvo IT and ESI. The dotted arrow illustrates the mutual affect the author’s research interest and the action cases have had on each other. Furthermore, the dotted arrows marked 1 to 3 represent gathering primary data through log-files written by the researchers participating in the different development activities. Below, we focus each block of activities and the presentation follows the structure ‘Elaboration,’ ‘Design’, ‘Testing’ and finally ‘Theoretical Grounding’.

---

19 We ignore the possibility to implement the meta-method in a tool, that might to some extent be possible to test through the use of automated test tools.
Figure 8. Research Process
2.2.3.1 Elaboration

The activity block ‘Elaboration’ was conducted in the empirical research environment ‘Method Engineering With Qualified Practitioners’. Hence, it involved research work together with qualified practitioners at Volvo IT and their MAPS team. The researchers were all tied to the VITS research group. This included the author who had the main responsibility for engineering the meta-method during this meta-method project. Below, we will refer to this constellation as the meta-method project team.

First, the meta-method project team took three main sources as the point of departure when starting the elaboration of MMC: method engineering literature, requirements at Volvo IT and the standard systems engineering method RUP. The method engineering field has been important for the meta-method project team, since thoughts about reuse and situational methods already existed in that field. Subsequently, it has been a foundation for both the project, as well as for as for the thesis (basically this part is found in Chapter 3).

Second, the empirical research environment has shaped the foundation for the meta-method project team. Especially the sub-environment at Volvo IT and the elicited requirements there were important, since the meta-method had to function and receive acceptance within this business. The sub-environment at ESI had no impact on the meta-method until late in the development process, which is quite natural when considering the time span between these action cases (illustrated in Figure 7).

The third point of departure was RUP, the standard systems engineering method used at Volvo IT. In this case, we do not refer to RUP as part of our research process, rather as the systems engineering method to tailor with the meta-method. This means that RUP’s internal structure came into focus during many parts of the meta-method project.

During the MMC project’s early stages the meta-method project team worked together with practitioners at Volvo IT that were using or were supposed to use RUP in their system engineering projects. This was done in order to gain knowledge about different aspects of the business context at Volvo IT. The two main contributors were Web Program Center (WPC) and a project-team at Volvo IT Skövde. This work is referred to as Pre-case in Figure 7. During this stage, the meta-method project team gathered data for the requirements analysis, which is illustrated as the ‘Analyze the Problem’ activity in Figure 8. This work was mainly performed through workshops and was documented in log files. The results have been described earlier in Sections 1.5 to 1.7. The intention of such an approach was to reach a more durable meta-method prototype.

Considering our research process and its foundation in parts of RUP, the Business Modeling discipline was used during the work at Volvo IT. For example, we used RUP’s vision document, business glossary and business-use case modeling (Kruchten, 1999), in order to structure these requirements. The vision document was

---

20 It was also used to elaborate the research question in this thesis. However, it was performed in the activity ‘Analyze the Thesis Problem’ found in the ‘Theoretical Grounding’ activity block.
used to delimit the purpose and the scope of the meta-method project at Volvo IT. It included refining problem statements, which were turned into business opportunities for the meta-method project. Each problem statement was detailed with descriptions of the problem, its impact and whom it affected. Rough ideas about solutions were also included. These problem statements have also been used as the foundation for this thesis. Through the vision document\textsuperscript{21} the meta-method project team could be more precise about who the intended users of MMC were. Thus, it was a source of important inputs for business-use case modeling. In addition, the glossary was used to establish a common vocabulary during the meta-method project and support the internal grounding.

The descriptions of the standard systems engineering method, as well as the meta-method were made using meta-models. Since the meta-method project team worked together with consultants from Rational Software Nordic AB they have been the guarantors when interpreting the standard systems engineering method. Furthermore, while working with granularity issues, new concepts were defined and later operationalized into an analysis models. The reason for such an elaboration was that the meta-method project team could not find the necessary concepts in RUP or in the business context at Volvo IT. Of course, some concepts could be found outside this context and the chosen standard systems engineering method, however such search was not the primary focus during this stage of the study. We refer to this work as the activities ‘Internally Grounding Concepts’ and ‘Operationalize Concepts’ in Figure 8. This resulted in rules for how certain concepts should be used in the meta-method.

During the work with concepts, creative modeling techniques\textsuperscript{22} were gradually exchanged for more formal ones. Finally, for this thesis, the author chose UML (Booch et al., 1999) in order to represent concepts and their associations.

2.2.3.2 Design

The thesis’ aim and thus the aim of MMC includes enabling reusable configurations based on a chosen standard systems engineering method. This aim yields a basic design principle when considering the significance of reusability (see Section 4.1.3). The meaning is that all prescribed actions should not be performed with the same frequency. Hence, the meta-method’s framework had to support a diversified performance.

The activity ‘Operationalize Concepts’ in the ‘Elaboration’ resulted, for example, in the classification matrix in Section 4.2.4.1 and its basic rules. However, such an operationalization is very incomplete in terms of a method. Descriptive procedures, which include the use of, for example, a classification matrix, how they should be performed in relation to each other, and specifications of prerequisites and deliverables had to be developed. In Figure 8 above we refer to this as the activity ‘Define Meta-Method Parts.’

\textsuperscript{21} The vision document is a deliverable from business modeling according to RUP.

\textsuperscript{22} For example, rich pictures and theory graphs
With RUP as part of the research process, the natural choice fell on further development of the business-use cases from the ‘Elaboration’ block. The foundation in these use cases gave the meta-method project team brief descriptions of the meta-method, or a discipline for method configuration when considering the possibilities for integrating MMC into RUP. Hence, the meta-method project team identified pre-requisites for each part of MMC’s prescribed actions on different levels of granularity, the roles associated with these actions and finally a rough framework to enable a structure for these actions. The final version of this framework is presented in Figure 28 in Section 5.3. Furthermore, the meta-method project team also assessed which risks that had to be eliminated or reduced in order to establish functional procedures during the meta-method project. This was done in order to give priority to the different development tasks needed.

The author, as well as the rest of the meta-method project team, has continuously made a distinction between modeling the content of the meta-method, which is described above, and turning it into a written method description. The former is referred to as the activity, ‘Defining the Meta-Method’ and is regarded as part of the ‘Method Engineering with Qualified Practitioners’ research environment. The author has had the responsibility for turning the modeled meta-method’s content into a detailed meta-method description using the business-use cases as input. This activity is referred to as ‘Externalize the Meta-Method’ in Figure 8. It would not have been an efficient use of resources if the complete meta-method project team had actively participated in this activity. Hence, the ‘Design’-block in Figure 8 spans two research environments, and is part of ‘Reflected Method Engineering’ as well as ‘Method Engineering with Qualified Practitioners’.

The meta-method project team produced several unofficial release candidates (versions) of the MMC. A release was often done after an empirical iteration had made a major impact on the meta-method’s content or framework. The version presented in this thesis is version 1.48. The log files used for documentation during each workshop with the meta-method project team were used to capture the argumentation for realizing each chance, thus forming the input for this thesis.

2.2.3.3 Testing
Testing is about the meta-method, sections of it or rough ideas put into action and evaluating their consequences. Thus, in the activity ‘Test of Meta-Method’ we address the action aspect of methods (Goldkuhl, 1994). Through the action we want to realize the intended consequences in an empirical research context, the ‘Method Testing,’ and evaluate them in the activity ‘Evaluate Episode of Usage.’ The latter is part of the ‘Reflected Method Engineering’ research environment, and hence ‘Testing’ in Figure 8 spans two research environments. The need for testing has changed during the development process. Furthermore, the need for testing has been

---

23 In the discussion, we will sometimes use the term discipline when we refer to granularity discussions in the MMC’s structure with regard to issues of integration into RUP. This is needed since RUP is the action case study object and MMC will be externalized in a RUP shell.

24 This is the fourth major version, and the eighth revision.
motivated by both the meta-method project team’s interests, as well as the author’s research interest. As far as possible, the testing has been a combined effort within the meta-method project at Volvo IT. However, we will elaborate on the principle possibilities for testing before discussing the chosen test scene. From this principle scene for testing, the author has selected episodes of usage for evaluation.

**Principle Scene for Testing**

As sketched in Figure 9, below, testing as well as evaluating the meta-method could entail several things. Recapturing the layers in Figure 5 (see Section 2.1.1), the chain of effects could be long, if an exhaustive test and evaluation are to be done. Consider we have three main activities for testing and evaluating the meta-method. These are illustrated in Figure 9 as ‘Configuration of Reusable Configurations,’ ‘Reuse of Pre-made Configurations’ and ‘Situational Method Usage.’ Furthermore, these can be divided into two categories depending on their direct or indirect concern with the meta-method. The first category includes ‘Configuration of Reusable Configurations,’ and ‘Reuse of Pre-made Configurations.’ During these activities, we would use the meta-method with the potential of addressing the action and consequential aspects of the meta-method, and evaluating if it is appropriate for selecting and tuning a pre-made configuration into a situational method. In ‘Configuration of Reusable Configurations’ we transformed the ideal type of the standard systems engineering method into reusable assets through the use of the meta-method. Thus, as input we then need a standard systems engineering method (in our case RUP), the new meta-method and configuration requirements. The resulting artifacts from this activity are pre-made configurations. From this set of reusable configurations we could then select a pre-made configuration that matches (hopefully) a specific information system development project.

The second category, ‘Situational Method Usage,’ has its main focus on the action and consequential aspect of the chosen situational systems engineering method. Thus, we move beyond the meta-method’s scope. Instead, we would use its resulting artifact, the situational systems engineering method. The consequences of this method can be evaluated both during the actual use and the effects it has on a developed IT-system. Evaluation of the actual use should then focus on the situational systems engineering method and its strengths and weaknesses with regard to a specific project. Weaknesses could be due to, for example, incongruence in the situational systems engineering method or those, for the project, important prescribed actions that might be missing.

During evaluation we would have to classify problems whether or not they origin from method configurations, and hence the meta-method, or not. Evaluation of the end product would be even more distant from the use of the meta-method. Usage of the developed IT-system affects the business context that it is part of and there can be both positive and negative effects. Furthermore, these effects can be deliberate or occur by mere accident. Consequently, we would then have to consider tracing these effects back to where in the situational systems engineering method their design was supported or not. The next step would then be to trace the specific configuration back to the features of the meta-method and evaluate if the effect
Research Design

originates there. As one soon understands, the more distance there is between the meta-method and what is evaluated, the more parameters there are to be considered.

**Figure 9.** The Usage-Situation of Method for Method Configuration

*The Test Scene: Selected Episodes of Usage*

An evaluation as exhaustive as that outlined in Figure 9, is time-consuming. For example, effects from IT systems do not occur immediately. Thus, that is another important aspect to consider. For practical reasons, the author had to delimit the test and evaluation. It had to be performed during a reasonable amount of time. Therefore, evaluation of the situational systems engineering method and its resulting artifact is beyond the scope of this thesis. The center of interest is creating pre-made configurations based on MMC. This is a natural choice, having test and evaluation episodes more directly related to the meta-method. In addition, this choice allows the possibility to influence and further refine the meta-method, the analytical framework and the concepts, before the meta-method’s official release.

Returning to the author’s selection of episodes they are delimited to the first activity described above, and illustrated in Figure 9 as ‘Configuration of Reusable Configurations.’ These episodes are referred to as ‘The Early Workshop,’ ‘The Large Workshop’ and ‘The Personnel System.’ A detailed description of both the set-up, the realization and the experiences obtained can be found in Chapter 7. In addition, a more general description of the research environments has been presented in Sections 2.2.2.1 and 2.2.2.2. Hence, we will only present a brief description of the set-ups below.

The first two workshops were both conducted together with Volvo IT. The aim of ’The Early Workshop’ was to test parts of the meta-method at an early stage of the development process. The meta-method project team conducted a two-day
workshop at a conference facility in order to bar ordinary work. Small configurations were made on demarcated sections of the standard systems engineering method in order to test concepts and drafted procedures. Thus, this workshop was one part in illustrating the design decisions made during the meta-method project.

‘The Large Workshop’ was conducted at Rational Software Nordic AB in Gothenburg. During this workshop, the meta-method project team was extended with consultants from Rational Software Nordic AB and the chief developer of Rational Process Workbench (RPW) from Rational Software Corporation in Vancouver. This set-up was useful, since RPW was tested as a configuration tool for RUP, and its implicit way of working and concepts were compared to those of the meta-method, with a focus on integration.

The meta-method project team was not involved in the third episode, ‘The Personnel System.’ A second empirical test environment was used, this time at ESI. The main difference between this episode and the first two was its completeness. None of the episodes are complete configurations if we consider that it should be possible to select one pre-made configuration from a set of pre-made configurations. The test episode ‘The Personnel System’ has the widest scope where one complete configuration has been made by the author in the roles as method engineer and project manager.

<table>
<thead>
<tr>
<th>Focus</th>
<th>The Early Workshop</th>
<th>The Large Workshop</th>
<th>The Personnel System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action-Effect</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Concept Integration</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Method/Tool Integration</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

These episodes were deliberately focused on different areas of improvements, merits and considerations to be made when applying the MMC. The evaluation was divided into three different areas of concern, presented in the vertical dimension of Table 2, above. Methods are, as described earlier, perceived as action knowledge. Reading Goldkuhl (1999), we can see that actions have effects. However, the prescribed statements in a method have intended goals, which do not naturally map cause and effects in empirical observations. Hence, we need to evaluate whether or not the prescribed actions are successful when used for method configuration. Therefore, two of the evaluations, ‘The Large Workshop’ and ‘The Personnel System’ have this specific focus (‘Action-Effect’ in Table 2). Goldkuhl (ibid.), discusses the correspondence between prescribed actions and causes, on one hand, and goals and effects, on the other, which has been the foundation for the analysis.
Research Design

The current version of the meta-method at the time for testing was used as input, the purpose of each activity was to be compared with the effects. The empirical observations have been selected from the log-files written during each workshop and the personnel system project. The log-files have been distributed to the meta-method project team to facilitate for each team member to form an opinion of the empirical data. This way of working has a bias towards anomalies in MMC, but at the same time the log-files highlight the need for improvements in MMC. Therefore the log-files are one part of reconstructing the design process, and explaining the design decisions made.

During the research project, four major externalized versions of the MMC were released. The latest one is presented in Chapter 6. In essence, this means that the areas of improvement have already been fed back to the design process ('Design' in Figure 8) and modifications have been made. Thus, this was a deliberate decision, and Chapter 7 is partly used as argumentation for the method design presented in the previous chapter.

The second evaluation was, as described above, justified by the business context at Volvo IT and their choice of standard systems engineering method and tools. It was considered relevant to evaluate the RPW-tool, since it imposes a way of working as well\(^\text{25}\). This tool and the meta-method were intended to complement each other, and the choice to use a tool based on the standard systems engineering method was natural when considering the reasons for selecting such a method in the first place (see Section 1.2)\(^\text{26}\).

The evaluation had two different focuses; first, the author investigated the meta-method project team’s procedures when trying to use the meta-method (‘Method Integration’ in Table 2), as well as the tool, and second, the author investigated the correspondence between concepts in RPW and the meta-method (‘Concept Integration’ in Table 2). In order to fulfill the first part, the author used a Grounded Theory approach based on the paradigm model prerequisites, action and consequence (Strauss and Corbin, 1990). The choice of paradigm model was based on the correspondence between that pattern and the generic pattern of methods with input, action and deliverables. The aim was a pattern of the meta-method project team’s way of working and the author was not completely open-minded. From this work pattern, the author was able to address weaknesses and strengths of the meta-method project team’s way of working when trying to use the MMC together with tool support.

The second part of the evaluation can largely be characterized as a variety of external theoretical grounding. With the starting-point in MMC’s concepts, the author searched for correspondences in RPW. The concept’s scope and purpose have been compared, as well as the structure between the concepts. In order to obtain a

\(^{25}\) Assuming that each tool has an implicit method.

\(^{26}\) Rational’s development of Process Workbench has not been triggered or directly influenced by the work at Volvo IT. In essence, this illustrates the difficulties with standard system engineering methods and the elaboration of local versions. From Rational’s viewpoint, MMC is a local complement to RUP and is not incorporated into standard RUP.
more balanced picture, the author introduced a comparison with concepts from other method engineering literature, when concepts in focus did not match closely.

The evaluation of the third episode centered on the ‘action-effect,’ and thus shares a focus with the first evaluation. The main difference is, as discussed earlier, the completeness of this episode. The author had the opportunity to use the meta-method on a more extensive configuration task. However, the basic aim was the same: to evaluate whether the prescribed actions could be applied successfully, that is achieve the intended goals. Furthermore, there is a difference in time span between the first and the third episode; hence the latter is a test of a more stable version of MMC. The exchange of business context was also an interesting aspect since the meta-method was moved from Volvo IT for the first time.

2.2.3.4 Theoretical Grounding
The dotted iterative patterns in Figure 8 facilitated the theoretical grounding of the meta-method. We acknowledge external theoretical grounding as an important part of validation during method engineering (see Section 2.1.3). Each concept and the proposed flow of activities can be new, modified or borrowed, which should be made explicit. During the inner iteration, the meta-method project team was concerned about the internal as well as the major part of the empirical grounding. However, the external theoretical grounding was for the most part left out.

Thus, the dotted iterations had a more reflective purpose, however using the solid iterations for empirical grounded primary data. Consequently, the later were input for the activity ‘External Theoretical Grounding of Concepts’ and ‘Analyze the Thesis Problem’ in Figure 8. Furthermore, as illustrated, these iterations overlaid each other, they were not independent of each other and the exchange was not one-way only. Rather, a continuously exchange of experience was conducted in order to improve the results achieved in both environments.

Up to three researchers from the VITS research group participated in each workshop conducted by the meta-method project team, hence making efficient use of the log-file technique. One of the researchers always had the role of log-file writer, while the others could participate more actively in the design or test activities. On those few occasions a researcher participated alone, the workshop discussions were recorded on tape and later transformed into a log-file. Each log-file was distributed within the meta-method project team, thus making the gathering of primary data transparent. Selected log-files have also been distributed to temporary participants when feasible.

The ‘External Theoretical Grounding of Concepts’ in Figure 8 focused on existing knowledge in the method engineering domain. The selection within that domain was founded on values that were considered primary during the meta-method project. One main value has been reusable method configurations. Much attention has been given to existing possibilities for the decomposition of systems engineering methods and the areas that are proposed to be the focus of the analysis of their content. These perspectives are presented in Chapter 3, and are the basis for the author’s theoretical work, in Chapter 4, about MMC’s concepts and basic structure.
During both ‘External Theoretical Grounding of Concepts’ and ‘Analyze the Thesis Problem’ the author used graph techniques for modeling the content of selected meta-methods, approaches and thoughts. More precisely, graphs modeling problems, concepts and ways of working, came into use, though all of them are not used in the presentation. The theoretical grounding has, as the research process in Figure 8 illustrates, been refined through the intersections in the empirical environment. Thus, the refinement has been based on the author’s improved knowledge of applying the meta-method’s concepts and prescribed actions, as well as practitioners’ judgments of the same.

The activity ‘Analyze the Thesis Problem’ actually concerns the elaboration of many of the issues discussed in this chapter. Thus, returning once again to McKay and Marshall (2001), this activity covers the consideration of the initial and completing steps of the research interest in action research. However, their model only presents the exiting considerations as part of the iterative pattern, not the initial eliciting of research theme and questions. We disagree with this part of the model, since we cannot ignore learning more about the problem through working with it. This view gives the researcher the possibility to refine the research question and aim. Therefore, we propose to view at least the eliciting of research questions as part of the iterative pattern.

2.3 Summary

All knowledge has to be validated. In method engineering, this is a matter of quality assurance needed to ensure a successful project. In this chapter, we have discussed such knowledge validation based on a combination of internal, external theoretical, and empirical grounding (Goldkuhl, 1999), and how this grounding affects the research design. These three grounding processes have different aims and directions. Internal grounding emphasizes the knowledge’s congruency and consistency while external theoretical grounding implies that the developed knowledge is evaluated in the light of knowledge from relevant domains, which in our case is the field of method engineering. The third grounding process focuses on the operationalization of the meta-method through development and testing in an empirical environment.

We have presented a research process facilitating this validation through an interactive strategy in which the academic and empirical research environments meet. The development efforts have been conducted together with qualified practitioners at Volvo IT in an iterative pattern. The empirical research environment’s counterweight is an iterative pattern in the academic research environment facilitating reflected method engineering through external theoretical grounding.
The purpose of this chapter is to elaborate on the concept of method and related concepts. Many scholars have tried to define the concept of method as it is used within the field of information systems (e.g., Checkland, 1981; Rumbaugh, 1995; Brinkkemper, 1996; Goldkuhl et al., 1998; Gønholm and Ågerfalk, 1999), and author’s contribution can be found in the opening discussion of this thesis (see Section 1.1.2). Summarizing the discussion of recent decades we can find different uses for the concept. These differences can be seen both in the discussion of the concept of methods, as well as in method manuals (Blaha 1998; Kurchten 1999; Yourdon 1989). Furthermore, other method-related concepts have been proposed, which are central to Method Engineering. Finally, we discuss existing research in the field of method engineering with regard to situational systems engineering methods.

3.1 The Concept in Many Shapes

3.1.1 Method versus Methodology

Not only the meaning of concepts adds to confusion. The usage of different terms for describing a phenomenon adds to it. Unfortunately, this is the situation when it comes the concept of method. At least one term, methodology, can be found in literature that sometimes embodies the same meaning as we do as a method in this thesis. Kumar and Welke (1992), use the term methodology, which has been criticized by Brinkkemper (1996). Avison and Fitzgerald (1995), Nilsson (1995) as well as Checkland (1981), also use the term methodology. However, aligning with Brinkkemper this should not been seen as proper use of the term. Methodology should be reserved for the study of and theory of methods. Brinkkemper (ibid) states that there can only be one methodology, however we can find different schools within this field.
Reading Bennett et al (2002), we find another use of these two concepts. They use the term method when referring to situational methods, and the term methodology as the ideal method. Using their words ‘a methodology is a type while a method is its instantiation on one project.’ We understand the distinction they are making; however, we do not agree with it since such a distinction would leave us with a void when discussing the study and theory of methods. Furthermore, we believe that the prefixes situational and ideal capture the semantic difference better than using method and methodology, as Bennett et al. (ibid.) propose.

3.1.2 Method versus Process

Process is another concept used in much the same way as method. Obvious proof of this is the name of the method focused on in this thesis Rational Unified Process, even though it is still a systems engineering method. We will, in this thesis, make a distinction in how we use these concepts. Method refers to a prescribed way of working and its resulting documentation, in which the process refers to the complete activity of developing an information system. We will refer to the latter as the system development process. A method is applied, used, during a system development process.

3.2 The Concept of Method

The concept of method has a long history in the field of information systems and it is a concept that has evolved over the decades. A method is in a way an everyday concept, however hard to define. In our daily duties we try to simplify the things we are doing. We use procedures for performing tasks that are common in everyday life – everything from making breakfast to driving or picking up the children from kindergarten. Mostly, we perform procedures without noticing. The procedures are used to structure the many small things we have to do in order to get satisfying results (Berger and Luckmann, 1966). We make our coffee with a few number of steps that are usually the same because we can predict the results – how strong the coffee will be and what taste it will have. The aim is predictability and quality, quality obtained from of a certain amount of resources.

All the small steps for making this and that are based on several, often, unconscious decisions. We base, somewhat incisively, our way of making coffee on an analysis of the size of the coffee cup, the type of percolator, the type of roasting and of course our own taste. We vary our step depending on these parameters (and several others) and make use of different models and frameworks within which we perform actions. Considering our own preferences about ‘good’ coffee we found our actions on values and an overall perspective. Hence, we can find several concepts linked to the concept of method. However, let us focus on the method concept as such and leave the latter concepts to Sections 3.2.1 and 3.2.2.

The author has chosen to define the concept of method as below and it has been expanded in Figure 1.

\[27\] The concept was defined in the opening chapter; see Section 1.1.2 and Definition 1-1.
**Definition 3-1:** A method is normative prescribed actions performed by human actors in order to reach the actors’, or a subset of the actors’, goals.

Even though authors, within the information system field, define ‘method’ slightly differently, there seems to be a common understanding that a method consists of three interrelated parts. First, there is a way of working to guide the performance of activities. The way of working tells a project member what actions to take during a specific phase of information system development, thus these are prescriptive actions. Second, there is some sort of notation used to document the results of the activities. For example, the notation for drawing state charts. Finally, we have a set of concepts used to describe the problem domain (i.e., modeling primitives) and the method itself. These three main concepts are illustrated in Figure 10 below as well as in Figure 1 (see Section 1.1.2). Harmsen (1997) and Nilsson (1991) propose a somewhat broader definition of a systems engineering method, which comprises tools. Tools are often possibilities to automate some part of the development process, and hence ease the workload. The tool is, often, an implementation of delimited parts of a method, and hence the method is a prerequisite for the tool. Therefore, considering tools as part of the method as such seems strange. Most methods are possible to use without computerized tools.

Furthermore, Nilsson (1991) includes the interest group model as one of the three main constituents of a method. This is found in Nilsson (1995) as well, where the constituents of a method are made more precise: perspective, work model, and interest groups. However, the latter is not found in Goldkuhl’s or Brinkkemper’s description of what constitutes a method. Including roles becomes interesting when working with RUP. Goldkuhl states that cooperation principles have a relation to methods but are not part of them. In the author’s definition we have included actors, furthermore roles are found in our expanded discussion about the method concept.

![Figure 10. The constituents of a method (Goldkuhl, 1991)](image-url)
Goldkuhl’s (1991) view of the method concept is illustrated in Figure 10, though it has been changed later on. In Röstlinger and Goldkuhl (1994), we find that these three parts represent a method component, which is a term elaborated further in Karlsson (1997). This expresses a need for more flexibility, but also stability, in the construction and usage of methods than was found in the original thoughts. The requirements for classifying a delimited part of a method as a method component is whether or not it represents a coherent picture for solving a delimited problem in the system development process. According to Goldkuhl, a method is therefore a container for method components.

Brinkkemper (1996) presents a definition that in some sense resembles the way Goldkuhl uses these three concepts described above. He defines a method as:

‘A method is an approach to perform a system development project, based on a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities with corresponding development products.’

To each development activity, Brinkkemper links techniques. A technique embodies procedural aspects of a method, as well as notation. Hence, as Seigerroth (1998) states, procedural aspects correspond to Goldkuhl’s way of working, and thus to our prescribed actions. Techniques could, but do not have to, prescribe a notation, according to Brinkkemper. Consequently, Brinkkemper indicates a weak relationship between procedural aspects and notation. However, a notation embodies the modeling primitives that can be focused on, and rules about how they can be combined (Goldkuhl, 1991). This means that the concepts used in a method become important since they define these primitives. The usage of a certain notation can affect the way in which the conceptualization is performed, and is hence intertwined with the process or the procedural aspect, which we have illustrated in Figure 1 (see section 1.1.2). Making the distinction that we use concepts during a process and they are expressed through a notation improves the understanding of a method. For example, it is important how we define our concepts since a notation has to express its significance.

One reason for this weaker relationship between procedural aspects and notation is Brinkkemper’s (1996) use of the concept method fragments introduced by Harmsen (1994). This is a method-coherent part of a method (Harmsen, 1997). But method fragments are not coherent in the same respect as Röstlinger and Goldkuhl (1994) assert, since Brinkkemper distinguishes different types of method fragments; product fragments, which focus on the structure of a produced artifact and process fragments, which are descriptions of procedural aspects. The individual fragment cannot, on a conceptual level, contain information about relations to other fragments, for example, that process fragment X requires product fragment Y. However, Brinkkemper concludes that product fragments are usually prerequisites for process fragments, as well as deliverables from them. Ågerfalk (1999) relates these concepts to each other saying that a method component consists of ‘the
smallest meaningful assembly of method fragments to address a certain aspect of a problem.'

Figure 11. Method component related to Brinkkemper’s three orthogonal dimensions. An elaboration of Wistrand and Frohm (2001)

When we examine Brinkkemper’s discussion of method fragments we find three orthogonal dimensions. First, we have the level of granularity that a method fragment can reside on. Second, each method fragment can either be a process or a product fragment. Third, we can add the difference between a conceptual and a technical fragment. Hofstede and Verhoef (1997) made a cube of the described dimensions for information modeling, which later inspired Wistrand and Frohm (2001) to a refine it with Brinkkemper’s layer of granularity. The cube is illustrated in Figure 11, above, however we have extended it by adding the method component concept. The shaded area illustrates the method component. The method component covers each method fragment, process, product, and conceptual, up to the diagram level of granularity. Going slightly ahead of the discussion we will see that Goldkuhl does not include phases in the method component. Phases could be translated into Brinkkemper’s stage concept. This seems clear-cut until we consider that a method component can consist of other method components. However, then we are discussing combined method components, not elementary ones. Furthermore, since Brinkkemper makes a distinction between process (way of working) and product (way of documenting) we have to include both of them as building blocks of the
method component. Each of the fragments can then be viewed as conceptual or technical fragments. We will deliberately not add technical fragments to the method component since tools cannot be found in Goldkuhl’s definition of method.

From this illustration, we can conclude that, depending on the level of granularity, method fragments can have a higher precision than a method component. Hence, method fragments have an advantage when it comes to theoretical explanation power. However, method components have the advantage of being more manageable. Since method components hide details there seems to be a better balance between precision and cost. Method fragments offer too many options and considerations that have to be dealt with at the same time that they could become a burden. Details are only of use as long as the value of considering them is higher than the cost. When complexity arises and we have to many options the net value decreases.

3.2.1 Perspective and Methods

According to Brinkkemper (1996), each method is grounded in a way of thinking, which agrees with our view of methods as normative; namely the way of thinking governs the operationalized directions and rules. In addition, we find that an overall perspective is fundamental to Goldkuhl’s (1991) discussions as well. The way of thinking is the guiding light for the definition of important values during system engineering. Perspectives are central to Nurminen’s (1988) as well as Dahlbom and Mathiassen’s (1993) discussions, which center on the existence of different perspectives. From these discussions, we can conclude, as Avison and Fitzgerald (1995) do, that perspectives are normative, and hence methods are normative construction. Consequently, our aim in this thesis is normative.

A perspective means the way the world is perceived by the method engineer and later the method user. Every method or model is based on a perspective (Goldkuhl, 1991; Nilsson, 1995). Often, when we use methods or models we do not reflect on the perspective that is induced. It is certainly possible to find several causes but the perspective might not be expressed explicitly in the method description as such. In many cases, the perspective is implied, which is usually the case with tools. Irrespective of the awareness of the applied perspective, it justifies the way in which the method is constructed.

Goldkuhl (1994) presented an elaboration on the perspective concept and its relation to methods. As illustrated in Figure 12 (below), a perspective consists of definitions, categories, values and goals, which together affect an instance of a method. Since perspective is normative, it is natural to have a starting point in what you want to bring about as a method engineer, that is, your objectives. These are illustrated as goals in Figure 12 and wherever these goals are realized through method descriptions. Consequently, we can conclude that activities are performed for reasons, and methods are said to have an argumentative dimension (Ågerfalk and Åhlgren, 1999). These goals can be based on the most basic concept in the perspective, namely values. One example is the role the user has during the system

28 However, discrepancy does exist between perspectives as ideal types and internalized perspectives.
development process. The role has changed over the decades from passive to a more active and participating one (for example, Dahlbom and Mathiassen, 1993). In his discussion of perspectives, Nilsson (1995), says its constituents are ‘principles and assumptions of how systems work should be accomplished in companies.’ In other words, they are concepts close to the values in Goldkuhl’s model. Values could also affect the definitions that are used within a perspective. Each definition is a fragment of how phenomena in the world are perceived. Values and definitions also have a relationship through categories that can be derived from values. Phenomena can then be abstracted into these categories. A perspective can be reflected in the implementation of a method, since a method can embody categories from the perspective in the used concepts and notation. Hence, categories elaborated in a perspective have relationships to the modeling primitives used in the method. Methods can be criticized, as well as the perspective. A method, as such, can be criticized for not following the implications of a perspective or more fundamentally through criticism of the perspective as such.

Figure 12. Method in relation to perspective (Goldkuhl, 1994)

Working with tailoring methods means that we have to consider two perspectives. The first perspective is the perspective of the systems engineering method, which we are tailoring. This perspective could vary depending on the chosen method. The second perspective is the one that is the foundation for our meta-method. As described in Section 3.2, we could apply a range of different
perspectives to systems engineering methods when performing method engineering. As discussed above, a method can, at least, be thought of as monoliths, method components or method fragments. Hence, we have to be precise about the implications of regarding an ‘off-the-shelf’ method as the starting-point for configuration work and where we stand in relation to other work performed within the field of method engineering (see discussions in Sections 4.1 and 4.2).

3.2.2 Framework and Methods

The ordering of procedures in a method is important and the ordering has to be expressed to the method user. However, prescribed actions need a container to keep them together in a certain order. The cause, as we interpret it, is that each prescribed action in a method does not contain information about where it should, or could, be placed in a sequence of work, that is its relationship to other prescribed actions. Therefore, each method has a framework, which is an outline of the overall structure of what needs to be done and in what order (Goldkuhl, 1991). The framework is then divided into several phases and realized through method components, as illustrated in Figure 13 below. By means of the method components the framework obtains its content, since a framework, as such, only focuses on what should be done, not how.

Figure 13. The relation between perspective, framework, phases and method. Interpretation of Goldkuhl (1991)
The term model is also used with the same meaning as framework in the field of systems and method engineering. One example is Andersen’s (1994) discussion of a development model. A development model is an outline of the development process divided into phases in which methods are linked to each phase. Each phase has its own focus on what part of the task to solve, which agrees with the way Avison and Fitzgerald (1995) use the term framework. A method can be used during different phases, that is, it is a delimited description of a way of working. However, Andersen does not specify the level of granularity of ‘method.’

A framework is linked to a perspective, which is operationalized through the structure of the framework. For example, whether a process is to be based on the waterfall model or whether it is to have an iterative pattern. Furthermore, a framework defines and delimits its area of application. An area of application, according to Avison and Fitzgerald (1995), is ‘some part of the real world that is deemed to be problematical and worthy of investigation.’ Hence, a method component does not define its area of application solely on its own; it is defined through the framework in which it is used. Brinkkemper (1996), in his discussion of method fragment, says that a fragment can reside on different levels of granularity, that is, a method fragment can act as the framework for a situational method.

We can conclude that complex methods, which are methods composed of several method components, need a framework. However, Röstlinger and Goldkuhl (1994) make a distinction between composed method components and elementary ones. Our interpretation is that the latter does not have a framework but the former does, since it is composed. A composed method component could have several elementary method components that need to be related to each other. Much the same line of reasoning would be applicable to method fragments. A method fragment at the lower level of granularity cannot hold a framework. However, if we take a specific systems engineering method as the point of departure, then that method will give us a framework. Hence, our choice of way of working with, and more important perspective on, method configuration affect whether or not an active choice about a framework has to be made.

### 3.3 Method Engineering

This thesis is about the development of a meta-method for method configuration. Since we are moving away from the ad-hoc approach, referred to as a problem statement in Section 1.7, and towards a structured practice we are also moving towards engineering. As described by Ågerfalk (1999), engineering is development following a prescribed and planned practice.

The concept of engineering is used on different levels. It is a central part of turning system development into system engineering. The use of methods is the guideline for moving the project forward in a structured way. Moving to the meta level of software engineering, we focus on the artifacts that turn development into engineering, such as methods, guidelines and tools. Method engineering can thus be

---

29 Of course, we can always decide to change the framework within that system engineering method, but then the question is whether or not it is the same method.
Perspectives on Method Engineering

treated as the engineering of engineering, which we have discussed earlier in Section 2.1.1. Kumar and Welke (1992), introduced the term ‘methodology engineering’ as ‘meta-methodology for designing and implementing information systems development methodologies.’ Their bottom line is that we do not need to reinvent the wheel in infinity. Methods used by practitioners are often adapted to the situation for their specific projects. Instead of developing methods suitable for specific situations we can customize the already existing methods, and base adjustments on experiences from earlier projects.

Brinkkemper (1996) agrees with the idea as such but, as stated above, disagrees with the use of the term methodology. Rather he talks about ‘method engineering,’ which we agree with. Method engineering, as Brinkkemper defines it, is:

‘Method engineering is the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems.’

He includes the construction and adaptation of tools as part of method engineering, and thus uses a somewhat broader definition of method than we use in this thesis. We treat tools as something used to support work with a method. However, we can conclude that a lot of focus has been on tool development in this field, for example (e.g. Rossi, 1998; Grundy and Venable, 1996).

We use the concept method configuration in this thesis when we refer to the planned and reusable adaptation of a systems engineering method. Consequently, we can define it as below:

**Definition 3-2:** Method Configuration is the planned and reusable adaptation of a systems engineering method.

In Chapter 1, we state that method configuration is considered to be a particular form of method engineering. Röstlinger and Goldkuhl (1996), make a distinction between the configuration of a systems engineering method and engineering a method. Method configuration according to them is the natural adaptation done by the method user when using the systems engineering method. Experience gleaned from these adaptations can then trigger a method engineering activity in order to construct a new or modified systems engineering method. We do not question the existence of adaptations during method usage; however, in this thesis we reserve the concept of method configuration for reflected adaptations of a specific method prior to method usage. These planned modifications do not result in a new systems engineering method. Adaptations performed by the method user during the system development process are referred to as method adaptations.

Hence, we have two different sub-concepts of method engineering for turning an ideal systems engineering method into a situational method: method configuration and method adaptation. We do not propose a distinction just as a conceptual issue or that the usage of one concept excludes the other; rather through the distinction we become aware of which activities we primarily intend to support.
Meta-Method for Method Configuration

with the meta-method. For example, method configuration should be performed in such a way that partly reduces the need for method adaptation, partly facilitates the need for method adaptation that will always be there. The former could be emphasized through founding method configuration in experience gleaned from earlier software development projects.

![Figure 14. Degrees of flexibility in Method Engineering. Interpretation, Odell (1996).](image)

We can conclude that great interest has been devoted to building situational methods with a starting point in method components or method fragments (see for example, Harmsen, 1997; Odell, 1996; Punter and Lemmen, 1996; Rolland and Prakash, 1996). Consequently, from this perspective, methods are treated as containers for modules like method components or method fragments. However, this is only one perspective among several others, which affect how work with situational methods is perceived. Odell (1996) presented a continuum of situational method engineering, illustrated in Figure 14 (above). The continuum ranges from the use of rigid methods to modular method construction. In the left most case, we are not actually discussing any modifications of the method, or as Odell says, ‘[it] permits virtually no flexibility.’ The framework and the method descriptions are not possible to distinguish from each other. Taking a step to the right in Figure 14 we work with selections from different types of rigid methods. Despite the fact that there is the potential to conform the choice of method to the project this would not be an adaptation of the method as such. Rather, we are simply exchanging one rigid method for another, which could mean changing everything from the perspective down to the method description. The selection of paths within a method, the middle alternative in Figure 14, permits a type of predefined adaptation. More flexibility is offered through selecting and tuning a method outline. From a selected process model and data model further adaptations are made for the method to suite a specific project. In other words, we are working with modifications within a framework used for structuring the situational method. The most flexible solution, as Odell describes it, is modular method construction. Here, the situational method is built from predefined building blocks; he refers to the term method fragments.

We can find valuable work done in each of these categories. However, the category ‘Use of Rigid Method’ falls short of the situational issue, since there is no room for situational aspects. The research in the second category is about the

---

30 He uses the term methodology.
selection of rigid methods. This is not interesting since each business implementing a standard systems engineering method has already made that choice. Hence, the three most interesting categories to address are ‘Selection of Paths within Methods,’ ‘Selection and Tuning of Method Outline’ and ‘Modular Method Construction.’ They imply a flexible view of systems engineering methods through reuse. As Figure 14 illustrates and the discussion below will reveal, there are differences in the degree of flexibility. In Chapter 4, we will discuss how flexibility is delimited by the use of standard systems engineering methods as the starting point for method configuration. In such a case, we agree with the basic thought in the selection of paths within one method, which in our case is the standard systems engineering method. However, as Chapter 4 will show, a type of modular constructs is valuable in such an approach. It is difficult to determine how much the approach resembles ‘Selection and Tuning of Method Outline’ since the research about that category seems insignificant and not widely accessible, which in turn affects the discussion below. Thus, the discussion centers on the two remaining categories.

3.3.1 Selection of Paths within Methods

The third perspective discussed by Odell (1996), is the selection of paths within methods, a solution also briefly discussed in Harmsen (1997). He addresses its existence, but does not discuss an explicit way of working based on that view. Turning to Hares (1992), we find a presentation of different paths through the Information Engineering method, originating from a set of closely related systems engineering methods dating back to the 1970’s. Although the different paths are his focus, these are not specifically of interest to us. Rather, we have a need for eliciting his way of working when creating these paths. Reading Hares (ibid.), these aims become conflicting since Hares’ used procedures and concepts are not always explicit.

The selection of paths within a method naturally means that the starting point is a specific systems engineering method. Thus, this approach resembles our problem. Further, Hares (ibid.), acknowledges the need for paths by means of his observation that no systems engineering method31 scores ‘ten out of ten’. In other words, he indirectly states that systems engineering methods have to be made situational. Each systems engineering method has to fit the task where it is to be used. The environment concept is used as an abstraction of different types of projects. He identifies seven different environments, centralized processing, object orientation, distributed databases, expert systems, conversational, real time and neural networks, towards which different paths are targeted. The notion of environments is interesting; nevertheless their origin is vaguely described. We interpret the set of environment as dynamic, since Hares describes an enhancement of environments from the original central processing. Furthermore, we interpret paths as situational versions of the systems engineering method. Hares considers the first environment generic, hence the other six derive from or overlap that category. By using this

---

31 Actually he says design method
classification, there is always some aspect of a system engineering project that is to be considered as a core, things the projects have in common.

Hares (ibid.), claims that each category has characteristics that are unique. The characteristics are based on three different, but related, areas: concepts, technologies, and techniques. Each category has a set of specific concepts for capturing the environment, a set of specific technologies that facilitate the implementation and techniques for elaborating the design. As Hares (ibid.) argues, these areas have to support each other. Having appropriate concepts but a lack of technology is of no real use. He states that there is a cascading relationship where concepts are the base for technologies, which he claims precede the techniques. He bases this line of argumentation on the history of development of computer technologies, which has preceded the need for and development of systems engineering methods. The explanation of the relationship between concepts and technologies are not always straightforward. Technology is developed to support concepts and if it does not, the technology is considered unstable.

Considering the central processing the generic category, where concepts, technologies and techniques have generic characteristics, this category has an impact on the adaptation of the systems engineering method. Information Engineering is considered to have a core component usable in other environments. Hares (ibid.), argues that if concepts and technologies originating from central processing are preserved, when switching environments, then the techniques are preserved, too. This is a tempting as well as a straightforward thought; however it is also quite narrow since it does not consider, for example, that user groups could differ and evolve with new technology and the development team has the ability or inability to map new technologies to existing techniques. This demarcated view seems to stem from the basic thought that development of systems engineering methods is technology-driven, that is, focuses on the correspondence between technology and techniques. In contrast, this generic category emphasizes the central point when taking one systems engineering method as the starting point for configurations. Hares (ibid.), states that ‘one need unlearn nothing about Information Engineering when using the method in these new and additional environments,’ in which we exchange Information Engineering for the chosen standard systems engineering method.

An evaluation of generic systems engineering method sections is at the center of Hares’ way of working, in order to reuse as much as possible of existing method knowledge. In this case, we interpret method knowledge as the project members’ internalized version of the method, which we acknowledge as the most important to reuse. The evaluation focuses on eliciting general and specific concepts in systems engineering methods. General concepts are related to the generic category centralized processing, while specific concepts are concepts used in one specific type of environment. Thus the former is used as the core part for projects and is the foundation for the paths or situational methods.

The evaluation is further directed at strengths and weaknesses in the systems engineering method. In this intention, we have similarities with the strength analysis proposed by Goldkuhl (1996) as an important aspect of method evaluation, as a
valuation of the systems engineering method’s positive characteristics. However, Hares’ (ibid.) way of working is not that explicit. He makes a division into general and more specific or systematic considerations. General judgments cannot be attached to one specific part of the systems engineering method; rather they are applicable to all stages. Inversely, the systematic consideration can be related to specific sections of a systems engineering method. Furthermore, each consideration can fall into three categories, the conceptual, the structural and the technical. The conceptual part is related to the evaluation of concepts mentioned above, hence having both a generic and a specific category. Through the discussion of Information Engineering we draw the conclusion that a systems engineering method needs both generic concepts as well as specific ones in order to be flexible. Equally important is that specific concepts can be traced back to their environments and are used in techniques. Hence, different techniques are more or less applicable depending on the environment in question. Evaluation of the structure focuses on the procedural aspects of the systems engineering method, the order between different method sections. Tasks are then solved using different techniques.

Knowing a method’s strengths and weaknesses as well as its generic and specific sections is important. Hares (ibid.), presents workarounds for both general and specific problems arising from the use of specific techniques. Thus, there is a way of improving the systems engineering method. Furthermore, based on the concepts and their origin in specific environments he recognizes needs for enhancing the systems engineering method. Enhancement could involve both adding concepts to existing techniques as well as enhancing with new techniques. These enhanced specific method sections together with the generic method sections constitute a template for the situational systems engineering method applicable in, for example, a distributed system environment.

3.3.2 Modular Method Construction

3.3.2.1 Method Fragments

Brinkkemper (1996) summarizes many of the contributions that have been made to modular method construction. A more extensive description is found in Harmsen (1997). Both authors unite the thoughts from many researchers’ work on how to split systems engineering methods into modules and construct situational systems engineering methods based on them. This type of method engineering is based on the view that a method can be divided into fragments, as described above.

Method fragments are standardized building blocks based on a coherent part of a method. The construction of a situational method is structured into a configuration process, illustrated in Figure 15 below. The method base is the heart of Harmsen’s (ibid.) situational method engineering. In this method base (implemented as a database), method fragments are stored. Round this storage we find a total of five steps, either for managing the content of the method base or for using its content in order to create a situational method.
Meta-Method for Method Configuration

The content of the method base is maintained through the ‘Method Administration’ step. Methods, techniques and tools are classified into method fragments of different types, depending on their characteristics and the chosen level of granularity. A method fragment could reside on one of five layers: method, stage, model, diagram, or concept. Each method fragment is described with the Method Engineering Language (MEL), and is characterized through the use of properties.

![Diagram of the Situational Method Engineering Process](image)

**Figure 15.** The Process of Situational Method Engineering (Harmsen, 1997)

Among these properties, we find the purpose and the goal properties especially interesting, since they contain information about why a method fragment should be performed. The purpose property contains information about the function a method fragment has in a system engineering project. However, this property is delimited to the technical fragments and is introduced to distinguish between different types of tools. Hence, it cannot be used as a pervading property for analysis focused on the systems engineering method’s rationality. In such a case, the goal property seems to be a better choice. Each method fragment has a goal property, conceptual method fragments excluded. This property contains ‘the objectives of the method fragments

---

61
and its content.’ (Harmsen, ibid.). Harmsen concludes that goals have sub-goals, which are ordered in hierarchies. However, both these properties primarily, at least when presented as parts of MEL, focus on the representation of systems engineering methods in the method base. Consequently, they are not used as the primary focal area when determining to what extent prescribed actions should be part of a situational systems engineering method.

As illustrated in Figure 15, method assembly is preceded by a characterization of the situation and selection of method fragments. Characterization of the situation is about an assessment of the project goal, which in turn, affects situational factors as well as performance indicators. These are illustrated in Figure 16 below. The latter describe the expected success with regard to specific situation factors. Harmsen (1997) exemplifies this with the situation factors of low management commitment, which contributes to negative success on the performance indicator organization management. Harmsen’s intention is to use aspects of different scenarios that contribute to the success of the performance indicators in question. Thus, there is an indirect relationship between situation and scenario where heuristics are used for relating performance indicators, situation factors and scenario aspects.

A project scenario is elaborated for each project, which is the foundation for choosing method fragments from the method base. Concretely, the scenario aspects characterizing the project are used, since they relate directly to the properties used to represent a method fragment. As the next step, MEL is used, this time demarking the selection. The selection of method fragments can start from either a product view or a process view. Since products are deliverables from a process this selection will not be completely product- or process-driven. The selection of method fragments can be done either through selection from one method or selections made from more than one method.

Unrelated method fragments do not constitute a situational method by themselves. They must be assembled into a method. The assembly is based on rules.
that guarantee the quality of the situational method. The method should fit the situation, but it should also be consistent, efficient and robust. Therefore, a method assembly should be based on a strategy. For example, we can choose if the strategy should be done top-down or bottom-up. When we are working with an ‘off-the-shelf’ method as the starting point, the strategy would be top-down. The chosen method would then act as the method outline.

A situational method is used in the project environment. The performed development activities are founded in the specified method, and the project performance depends on how suitable the method is with regard to the project characteristics. However, as Harmsen (ibid.) states, the situational method cannot be deployed to the systems engineers without regard to their familiarity with the method. In other words, that is a disadvantage of this type of method engineering where much effort has to be put into training if the situational method differs, to a great extent, from one project to another. Using an ‘off-the-shelf’ method means that we are using a common method-kernel to build competence around.

### 3.3.2.2 Method Chains and Alliances

Nilsson (1999), discusses the need for toolboxes for method application. He does not delimit method to systems engineering methods, and this is important. He considers all types of methods in the field of systems engineering as well as methods from other disciplines. Nilsson (ibid.) means that IT does not exist for its own sake; rather it should be an integrated part of organizations and their business strategies. Furthermore, Nilsson claims there are three levels with different scope and focus. As a consequence, competence from different organizational levels and units are required. One problem is an organization’s ability to get a comprehensive understanding of the improvement areas, since there is a lack of communication.

The limited scope and integration of development levels can be derived from the problem that methods are not made situational. Since methods, as more limited ideal types, do not cover the needs in every possible situation they have to be made situational. Otherwise the actors in a change process will lack proper tools for establishing a common language. However, the major problem is not the lack of situational methods per se, rather Nilsson emphasizes the lack of guidelines for method integration. Thus, Nilsson argues for the development of proper guidelines for method engineering. However, it is interesting that the actor’s method competence is not considered at all. Guidelines both for the meta-level as well as the development work are not the only way methods can exist. Methods are only efficient when they are used by a team, as a team. This requires methods as intersubjective knowledge, and an understanding of each other’s needs in the organization. In essence, a major conceptual contribution is the introduction of method chain and method alliance (Nilsson, ibid.). These are the constituents that facilitate method integration, which is the proposed approach for engineering situational methods. A method chain is an integration of methods from different development levels. The second concept, method alliance, means a horizontal integration of methods; methods on the same developmental level are integrated. Together the aim is to develop the organization in a consistent way. From a system
Perspectives on Method Engineering

engineering view, this would mean development of business opportunities through the use of IT, and is not limited to either a top-down or a bottom-up approach.

Nilsson (ibid.) proposes a meta-model for this sub-type of method engineering, which should be applicable regardless if the need is horizontal or vertical integration. The meta-model consists of three focal areas: Intentions, Concepts and Ways of Working. Recalling Nilsson’s (1995) constituents of a method we could map them with focal areas, as in Figure 17. Intentions focus on modeling goals and visions. However, the concept intention is vague and the examples given mainly focus on goals. People in the interest group have goals, but through the use of methods the perspective justifies them. Hence, we interpret this as mainly meta-modeling of the perspective. The conceptual focus is meta modeling of the essential constructs of a method. Based on our view of perspectives, and in agreement with Goldkuhl (1993), these constructs are highly dependent on perspective. Consequently, the conceptual focus is an analysis of the method’s perspective. The last focal area, the way of working addresses a method’s work tasks, which is the guidelines for managing different issues during the development process. In this case, a method is an aggregate of work tasks, interpreting the concept method as an externalized method description. Summarizing the above discussion, the proposed meta-model is used for the harmonization of methods’ perspectives and work tasks, leaving the interest groups in the background. An indirect focus could be obtained since a method’s perspective affects the interest group as well as the work task. However, these effects are vague from the meta-model’s point of view. Performing an analysis using an overall picture of the methods, and based on this understanding

Figure 17. Three Focal Areas for Meta-Modeling

There has been an exchange in terminology where work task is used in Nilsson (1999), instead of work model in Nilsson (1995).
we could make consistent changes. However, we agree with Seigerroth’s (1998) that the relation between these three focal areas could be improved.

We find it interesting that Nilsson discusses existing methods (from vendors) when constructing a situational method. Existing methods resemble what we discuss as standard systems engineering methods, however we demarcate the discussion to the system engineering discipline. Hence, we are not explicitly concerned with the problem of vertical integration. Furthermore, we assume that integration issues have been considered in the standard method chosen as the starting point for configuration work. In other words, the initial method in MMC could have a larger scope than Nilsson’s ideal type methods, however situational system engineering method in MMC could have a narrower scope than Nilsson’s situational method. Still, we share his basic thought of leaving the refinement of the methods to the vendors and focus on the usage of reusable method assets. However, we operationalize the later differently. Nilsson uses a set of existing methods, while we want to take one single method as the point of departure.

3.4 Summary

In this chapter, we have elaborated on the concept of method and different views of method engineering. The method concept is a complex one, which can be demarcated differently. We can conclude that differences in perspectives determine how a phenomenon, such as a systems engineering method, is perceived. Consequently, the values and goals and their priority are important when designing a method description and these will be considered in the next chapter.

The inventory made by Odell (1996) of the method engineering field shows the range of existing research. Much attention has been given to the reuse strategy. This strategy has been implemented differently where modular construction is the main track, however some research can be found on adaptation with a systems engineering method as the starting point. In the latter we find the notion of a method-kernel interesting, that is, that some parts of a systems engineering method are generic and applicable in every project.

A situational systems engineering method has to be anchored in the business context. Harmsen (1997) utilizes situation factors and their impact on success in order to characterize a situation. We interpret these situation factors as universal for system engineering projects independent of organization, which is one option in a continuum. The opposite pole is characteristics delimited to a specific business context. Consequently, this is a consideration that has to be made in the next chapter.

From the existing research, we find both views interesting, since standard system engineering imposes one method as the starting point. In addition, we find the flexibility of modular construction valuable, but we have to combine such flexibility with the stability of rigorous standard systems engineering methods. We find the research of this combination insignificant and not widely accessible.

Systems engineering method’s rationality as the focal area for configuration analysis is not widely found within the method engineering field. Nilsson (1999)
Perspectives on Method Engineering

argues for using intentions when modeling method-integration, which we find interesting since intentions focus on modeling goals. However, the use of intentions is about a search for goals in prescribed actions that complement each other. With a standard systems engineering method as the point of departure, our main focus is instead on using the prescribed action’s rationality for reducing the method’s content.
CHAPTER FOUR

4 THE THEORETICAL FOUNDATION OF METHOD FOR METHOD CONFIGURATION

This chapter aims at elaborating on the Method for Method Configuration that, so far, has been no more than a vague picture. In the previous chapter, we discussed different perspectives on Method Engineering. Thus, we are at crossroads and have to address the foundation for our meta-method. With them as a starting-point we can start sketching a theoretical version of MMC. Subsequently, the presentation is intertwined with argumentation for design in relation to our perspective and an external theoretical grounding where we relate our concepts to existing method engineering literature. Since the research process has been iterative, as discussed in Chapter 2, we will to some extent address how the meta-method has changed over time. The evolution and refinements are based on the episodes found in Chapter 7. Thus, there is a relationship between the discussions in this chapter and how we have incorporated the experiences of utilizing the meta-method. The result from this chapter is the base for the latest operationalized RUP version of Method for Method Configuration. We address these considerations in Chapters 5 and 6.

4.1 Foundations of Method for Method Configuration

Recalling the concepts in Figure 12, about perspectives, the basic values are important for the method’s design. They are the foundation for defining concepts used in a method description and how we perceive the world when applying the method. In this section, we will clarify our foundations where systems engineering methods are considered as supporting tools in system engineering, and where we apply a top-down perspective on method configuration. Furthermore, reuse and flexibility are central values, but delimited by the use of standard systems engineering methods. Consequently, Section 4.1.1 is devoted to the method as

33 Chapter 7 contains a selection of different episodes from the meta-method project.
enabler, and Sections 4.1.2 to 4.1.5 discuss our standpoint on method configurations in this meta development project.

4.1.1 Method as Obstacle or Support

The use and value of methods as support for information system development projects is the foundation of this thesis. Naturally, this is a foundation that can be criticized, and it is certainly discussed. The use of formalized and large methods is, for instance, discussed by Beck (2000), who presents the framework of eXtreme Programming (XP). He presents a lightweight `method’ which challenges the assumptions of more traditional methods. Attempts have been made to relate XP to RUP, configuring RUP into a lightweight version (Pollice, 2001). Thus, treating XP as a special case of RUP. However, these discussions do not center on whether to use systems engineering methods or not, rather one can conclude that there is no one-size-fits-all method. A lightweight method does not consider all aspects of tentative projects and a full-blown method could be an obstacle if it is not configured.

Introna and Whitley (1997), address what they call a paradox in the field of information systems development. They present limitations of method engineering. Despite the work in the field of method engineering and situational method engineering, methods are not widely used in practice. Their criticism of method engineering is that researchers believe it is possible to incorporate all required knowledge in a method. They argue for a softer approach towards methods based on background knowledge about the world rather than a specific method. We agree with Röstlinger and Goldkuhl (1996) that systems engineering methods should not be in the forefront during systems engineering. However, Introna and Whitley have a narrow view of methods. In their paper, they treat methods and usage of methods as algorithms, a well-defined series of operations with a guaranteed output given certain input. Through the use of these algorithms, understanding of the problem at hand can be obtained. Applied this way, methods will, without a doubt, become obstacles. In this thesis, we do not treat methods as algorithms, rather we see them as heuristic procedures or heuristics (Langefors, 1995). There is a significant difference between these views. Having an algorithm perspective deprives responsibility and power of initiative from the method users. We are no longer considering them as human beings rather we place them on the same level as machines. Therefore, we do not treat methods as a container of all knowledge needed for solving the problem at hand. They contain regulations about system development activities (Lind, 1996), however we still need a competent systems engineering method user (Goldkuhl and Röstlinger, 1988).

One could argue that methods act as an obstacle to creativity and flexibility. Methods exist on different levels. A clear line exists between planned methods, that is the plans for performing a set of activities, and the used procedures. During the actual project, unforeseen problems and opportunities happen that could call for adaptation of the project’s course (Christensen and Kreiner, 1997). However, this flexibility can be provided in a controlled way through methods. If the method, and subsequently the project, is based on iterations the project can be adapted before the
upstart of the next iteration. In order to make adaptations in a controlled manner, they should be based on change requests. Of course, shorter iterations give more flexibility and need to be balanced against the risk of fragmenting tasks.

Methods are often believed to kill creativity, through formalization. Couger et al. (1993) state that this is a misconception. Solving problems is a creative process and thus project members need freedom. Then it becomes important to give these actors the freedom they need, however being explicit with the delimitation that exists. Thus, methods could be a way of setting the scope for freedom, and keeping the focus. It is important that creativity is used on the essential issues in the project. Ideas that do not add value to a solution are not creative. Consequently, methods need to fit the specific project so it does not create a work overload. Otherwise, it is true that methods kill creativity.

4.1.2 Top Down

Working with an ‘off-the-shelf’ method gives a natural, however not always simplified, starting point for configuration work. The point of departure is natural since a decision has been made to use a specific systems engineering method in system engineering projects. As described in Sections 3.2.1 and 3.2.2, above, each method has a perspective and a framework. With the decision to implement an organization-wide systems engineering method comes a simplification in the choice of perspective and framework. The choice becomes simplified because we do not need to choose a perspective and framework each time a situational systems engineering method is to be constructed. Of course, it is not a simple task to choose a suitable systems engineering method for organization-wide implementation. However, as we expressed in Section 1.12, such a discussion is not within the scope of this thesis. Furthermore, we acknowledge that large systems engineering methods, such as RUP, are probably eclectic compromises depending on the original building blocks. Certainly we can find systems engineering methods with multiple, and contradictory, perspectives. This depends on how the systems engineering method was constructed from the beginning and the original building blocks that were used. If a systems engineering method is based on method fragments or components attention has to be paid to harmonization. Furthermore, since we have made a distinction between ideal methods and situational systems engineering methods we also have a discrepancy between an ideal method’s perspective and the perspective used by a system engineering actor. Thus, there might not be one coherent perspective in a standard systems engineering method, but the choice of perspective follows the choice of method.

The pole opposite to top-down is generation of situational systems engineering methods from demarcated method parts, such as method components or method fragments. At least according to the description of a method component each selected component has the potential for a perspective of its own. Therefore, harmonization of method components or method fragments is central when adding method parts to a systems engineering method. Röstlinger and Goldkuhl (1994) state that a method component needs to have well defined interfaces and relations to other
method components. Reading Harmsen (1997), Brinkkemper (1996) and Rolland and Prakash (1996) we can see that the interface issue is equally important when using method fragment and method modules. The problem with method components, at least, is the lack of standards for defining interfaces. The problem is reduced in Röstlinger and Goldkuhl (1994) since they mainly discuss method components originating from the same method family, the SIMM-family.

Consequently, when we limit our configuration work to one single systems engineering method we reduce the harmonization problem. However, at the same time we should be aware of the fact that we impose a restriction on the possible configurations. We are limited to the ability of the chosen perspective and framework. In order to address issues which are not within the perspective or framework’s ability calls for extensions of the systems engineering method and its goals. This becomes evident when reading Hares (1992), however he does not consider the goal issue.

4.1.3 Reuse

Software engineering projects have differences and similarities. This is illustrated both in Hares (1992), and van Slooten and Hodes (1996). We have to take both the differences as well as the similarities into consideration when performing method configuration. When similarities exist we regard reuse as one way not to invent the wheel a second or third time. Reuse is a central concept and value within component-based software engineering, both as implementation and as design patterns. There is argued to improve quality, development productivity, and cost (Pressman and Ince, 2000). Recalling the work done within the field of method engineering we see that reuse is central to that area too. Both Röstlinger and Goldkuhl’s (1996) view of method components and Brinkkemper’s (1996) view of method fragment aim at reuse. Furthermore, Röstlinger and Goldkuhl discuss the trend in the field of system engineering with demands for shorter development cycles. We acknowledge the value of reuse and the thoughts that lead-times, quality and cost are improved if we are able to find a way of reusing method configurations.

Nevertheless, we can conclude that reuse is not simple, neither as a concept nor in practice. As discussed by Christiansson (2000) we have two concepts that are similar to each other: use and reuse. An existing solution can be reused to solve an old problem or to solve a new one. However, the concept reuse should only be used if it is a planned consideration to create and maintain specific resources for this purpose. We find support for this idea in Jacobson et al. (1999). They discuss a reusable subsystem and argue that there should be an intention behind creating reusable assets. According to them ‘we get reusable subsystems by carefully designing them so that they can be used together.’

However, it is delimiting to focus only on the concepts component and subsystem as such when discussing reuse. Considering the reason for externalizing a method the reason is about making knowledge inter-subject and facilitating for similar methods to be conducted over a set of projects. On an abstract level we are searching for patterns (Pernici, B et al, 2000), in which we propose that these
patterns are pre-planed. Thus, when discussing a meta-method for method configuration we are actually searching for patterns that are recurrent. A pattern does not necessarily say that it should be implemented as a method component or a fragment.

Reuse aims at reducing spending on resources. This means that the activity of generating reusable assets should be considered as an investment (Christiansson, 2000). Thus, we should consider both the cost and the value we are generating. While the cost is the sacrifice we make, value is what we hope to gain in future benefits (Kam, 1990). Since future benefits do not exist at the moment when the cost arises our expectations may not be realized. In some cases the cost incurred exceeds the received value.

Subsequently, reuse implies a standpoint, whether it is worth investing the resources needed. This is a consideration that has to be made for method configuration as well, and especially when building a meta-method for tailoring other systems engineering methods. We have to consider the investment both in constructing a meta-method and using it to tailor other systems engineering methods for reuse. Since method engineering is complex and time-consuming we have taken the standpoint that reuse is an important aspect in method engineering. The higher complexity we have in the systems engineering method that we intend to tailor the more resources we spend each time it is to be tailored. However, the value added through reuse has to be considered in each case in which a systems engineering method is to be tailored, since the aim is to reduce resource consumption.

4.1.4 Flexibility

A component-view of reuse often means treating the reused resource as a black box (Christiansson, 2001). With a black box or information hiding philosophy it is possible to create an environment for code-reuse. Each module guarantees certain postconditions irrespective of context as long as each module’s preconditions are met (Main and Savitch, 1995). The possibility to use each such module adds to flexibility. These ideas have been translated into the domain of method engineering (e.g. Röstlinger and Goldkuhl (1996), Karlsson (1997)). Reading Goldkuhl (1991) flexibility is an important aspect of method engineering, and this is emphasized even more in the component-based view in Röstlinger and Goldkuhl (1996). At the same time, we have a trade-off with stability and standardization (Goldkuhl, 1991), which are important aspects of a systems engineering method as well. The smaller modules we intend to work with, the more flexibility we will have, but we are trading stability.

Working bottom-up when constructing situational systems engineering methods gives great flexibility in the sense that methods are constructed from a set of modules. Each module could originate from different systems engineering methods. We trade this type of flexibility for the stability of using one systems engineering method as the starting point. Stability gives a base for inter-subjective knowledge about standards and perspectives within the organization. Subsequently, we will focus on flexibility within the chosen standard systems engineering method.
4.1.5 Rationality

We agree with Goldkuhl (1991) that the main constituents of a systems engineering method are way of working, notation and concepts. Each of the prescribed action in a systems engineering method has a purpose to fulfill, along with the artifacts produced. These actions and artifacts should contribute to the overall goal of the project in question, otherwise they cannot be considered valuable. This idea is in agreement with one of Weber’s (1983) four types of actions, those based on goal rationales. Weber claims that an action is rational if it is oriented towards a deliberate goal or value and when it is, with regard to existing knowledge, based on means that best fulfill that goal or value. However, Weber points out that there is no objective meaning in the notion of ‘best’ since the notion is based on existing knowledge.

The purpose of prescribed actions, consequently systems engineering methods, is discussed in Goldkuhl’s (1999) text on grounding usable knowledge. He claims that each prescribed action is intended to lead to goals. These are goals of the method engineer and thus are normative statements founded in the method engineer’s perspective. Thus, the goals and values of a perspective are the foundation of the systems engineering method’s rationality. Nevertheless, we have to consider that goals are highly dependent on the context. For example, returning to our orange case the importance of piling all searched chairs in the room depends on the amount of chairs. The goal was to obtain knowledge about searched chairs. For the fictitious project team it was considered important in order to sort out a confusing situation. In an extreme situation with no chairs, that specific goal is unimportant and thus the activity is unnecessary to perform. We could argue that the goal had already been met. Furthermore, reaching a goal could differ in complexity depending on the context. Having knowledge about searched chairs was only a sub-goal when gathering oranges, and achieving that sub-goal as well as the overall goal would have been of insignificant complexity without blindfolds.

Stolterman (1991) discusses the importance of creating an understanding of a prescribed action’s rationality. In order to achieve a systems engineering method that supports an actor during a project the method’s rationality has to be usefully expressed. Stolterman (ibid.) exemplifies with a detailed recipe for baking a cake and the advantages when the rationality for each step is expressed. Then the baker has the possibility to tailor the baking to the particular situation. The same thoughts can be applied to method configuration. Since a large standard systems engineering method is complex it also implies a wide range of prescribed actions possible to perform, to perform modified or not perform at all. As discussed in Section 4.1.3, we consider reuse important and treat method configuration as an investment. This investment should be compared to a continuous cost for making decisions on every prescribed action in the method for every project in which the method is used. The basic idea of reuse in MMC is to add a filter-function between the ideal systems engineering method’s rationality and the situational systems engineering method intended for use in a specific project. This should not be interpreted as not believing in competent method users or that rationality is unimportant when internalization a
systems engineering method; rather the intention is to reduce the amount of decisions about the method when using a method. Instead, experiences could be used to sort out the unnecessary prescribed action and to do so based on the systems engineering method’s rationality. In this way, we could emphasize the rationality that should facilitate carrying out a system engineering process, which is important in Stolterman (ibid.) However, this type of method configuration is dependent on the sharpness of the systems engineering method’s rationality. This is only natural since we intend to use the same basic principle as the method user uses in current work. However, if a systems engineering method’s rationality is vaguely presented a reuse and the investment perspective on situational systems engineering methods becomes even more important.

Several authors in the method engineering literature propose modeling goals, nevertheless, differences can be found. Nilsson (1999) discusses modeling intentions in methods. His focus is on goals as supplementary or contradictory to each other, which is an important consideration when methods are to be integrated into a larger method. However, the importance of reaching certain goals is not explicitly discussed in Nilsson (1999). Of course, there has to be a reason for considering goals as supplementary and this is the existence of an overall goal, but this issue is not placed in the foreground during his discussion.

Ågerfalk and Åhlgren (1999) are more explicit in explaining why it is important to model goals, and they argue for a complement to modeling systems engineering methods, focusing on the rationale of methods. By understanding the goals of a specific project or a set of similar projects these goals can be compared with the goals of the systems engineering method, and a better match could be possible to find. Ågerfalk and Åhlgren argue that too much focus is put on conceptual focus when combining methods, and that the rationale aspect is neglected. Reading for example Brinkkemper (1996) and Harmsen (1997) who treat methods as reusable assets we agree with this criticism. Modeling systems engineering methods into method fragments as well as the selection of fragments is based on conceptual investigations. Furthermore, this is not a problem bound to the modular construction perspective; we find it in Hares (1992) as well. He uses concepts as the base for paths through a specific method, but he does not discuss the relation between concepts, values and goals. Reading Mathiassen et al. (1996) we find that it is more important what is done than how it is done. The implication is that the principles behind a method are important, that is, addressing why certain tasks are performed, which agrees with the importance of perspectives in systems engineering methods (Goldkuhl, 1994).

Kukkonen (1996) discusses design rationale as ‘why an artifact has been designed the way it has’ and applies this rationale to systems engineering methods. He defines method rationale as ‘method design and usage rationale and their linkage to design artifacts across various phases of method engineering and use.’ Thus, rationale is important both as an aspect of method engineering as well as an aspect

---

34 Not limited to system engineering methods.
35 They use the concept organization.
of systems engineering method usage. Explicit rationales can improve communication between team members and stakeholders (Kukkonen, ibid.) if they can be derived from the overall goal. The communication aspect of method is pinpointed in Ågerfalk and Åhlgren (1999) where systems engineering methods are viewed from a speech act perspective; consequently emphasizing method descriptions and examples of method usage as communicative actions. The purpose is to externalize methods in such a way that method users can internalize them and use them competently during system engineering. Subsequently, the rationale of the method engineer should be communicated to the method user to facilitate competent usage. However, in many cases this is implicit since rationales behind a certain method may not be found in the method description. Ågerfalk and Åhlgren (ibid.) have proposed a complementing approach in three steps. First, they propose modeling the activity structure and second, a goal analysis. Analysis of activities elicits results and prerequisites for performing actions while goal analysis focuses on relationships between different goals. The third and last step in modeling rationale is identifying relationships between the activity set and the set of goals.

The relations between different goals result in a hierarchy, which implies that goals exist on different levels of granularity. Consequently, goals on a lower level of granularity are prerequisites for goals on a superior level and this maps to the thoughts that systems engineering methods can be focused on different level of granularity. Not meeting or finding a goal on a low level of granularity greatly reduces the possibilities of reaching the overall goal. With MMC we do not see goal modeling as a complement to conceptual modeling, rather we see it is fundamental when conducting method configuration based on the systems engineering method’s rationality. From our top-down perspective on method configuration harmonizing systems engineering methods or parts of them is not the central issue, thus conceptual modeling is placed in the background.

4.2 The Concept Model and The Analytical Tool

We start crafting MMC with a discussion of its conceptual model, which is based on the foundations presented above. In order to give an overview of the concepts used, a simplified conceptual model is presented in Figure 18, below, using UML notation. In the following sections, 4.2.1 to 4.2.5, we address each of these concepts as well as introducing a few more. At the end of this chapter, an extended conceptual model is shown (see Section 4.4). Furthermore, during this presentation we will elaborate on turning these concepts into an analytical tool for addressing a systems engineering method’s rationality. The presentation in this chapter is not to be bound to a specific standard systems engineering method. However, in those cases in which we need to be more concrete in our discussions we will exemplify using the Rational Unified Process.
In addition to the conceptual structure we will elaborate a notation for representation and retrieval of the configuration content. We have founded our notation in XML (Bradley, 2000). Consequently, our notation is an operationalization of the concepts discussed in this chapter. We do not claim the representation is normalized, rather a principle one, which emphasizes overview. Furthermore, we have chosen XML since it facilitates a separation of content and its presentation in an efficient way. Our need is a representation issue, which should not be bound to a specific tool. However, XML is an ideal framework when considering future data exchange with different tools\(^3\).

### 4.2.1 Base Method

MMC is a meta-method for focusing on certain aspects of the configuration work, emphasized on the analysis part of method configuration\(^3\). One part of the aim is

---

\(^3\) Only using a simple browser when assessing, for example, the content of a configuration package is valuable, since one can expand and suppress nodes in the structure, respectively.

\(^3\) The implementation and publishing issues are highly dependent on the chosen standard system engineering method. For example, when working with RUP it is possible to use an extension in Rose to modify and publish a proposed configuration.
also to create reusable configurations based on experience gleaned from earlier projects when using the chosen standard systems engineering method. Since we have a top-down perspective on configurations and focus on making this method situational, subsequently, this method has a specific role in MMC and we have elaborated the concept base method during the meta-method project. Put in another way, the base method is the systems engineering method of interest for configuration, which in our case studies has been RUP. Thus, a base method is defined as:

**Definition 4-1:** A Base Method is the systems engineering method chosen as the starting point for the configuration work.

The base method could be any type of systems engineering method. In this thesis, the center of interest is ‘off-the-shelf’ methods or using the more specific concept of standard systems engineering methods. Consequently, there is no hinder to using an in-house developed systems engineering method as a base method. However, considering the limitations of our perspective the base method should cover the most frequent occurring projects in the organization in order to offer completeness with regard to the development process. When other systems engineering methods are discussed in MMC they are referred to as additional methods, a concept defined as:

**Definition 4-2:** An additional method is any systems engineering method that is not considered as the base method.

When reading method engineering literature we can compare the base method concept with existing concepts. We find differences when comparing the concept of base method with concepts found in approaches applying a bottom-up perspective to configuration or construction of method engineering. Often there is the resulting artifact of the configuration effort, the situational systems engineering method, which is explicitly addressed. For example, Brinkkemper (1996) does not discuss the starting point for method engineering in terms of one method, rather as a set of method fragments (see Section 3.3.2). A method fragment is, according to him, a delimited description of a systems engineering method or a logical coherent part thereof. Of course, we could consider using one of the method fragments as a starting point. However, then we must view method fragment on the method level of granularity to equal a base method. Then this method fragment should consist of other method fragments of finer granularity than is usually considered as the starting point from a bottom-up approach.

Much the same line of reasoning can be applied to method components. Depending on the type of method component we have in mind, we could use a method component as a base method. Viewing a composite method component as a complete method, such a method component could act as a base method. Again we need to view the module as a complete method since we need to have a framework
within the base method. Elementary method components do not contain a framework (Röstlinger and Goldkuhl, 1996).

Harmsen (1997) discusses the concept method outline, which is ‘a composite method fragment without contents.’ The usage of a method outline means applying a sort of top-down view to method engineering. This restricts the method fragments that are possible to combine with the method outline. Nevertheless, a method outline does not seem to have the content of a base method, rather it contains a process model and a data model. Consequently, it is more of an empty framework. A third approach is a selection of paths within one systems engineering method (Harmsen, 1997). In this case, the method has content and could therefore act as the starting point for configuration and subsequently could be considered to be a base method. As described in Section 3.3.1 Hares (1992) uses one systems engineering method as the starting point. The basic idea of this point seems equivalent, but in Hares it is not an explicitly stated concept. One reason could be the fact that he does not propose a meta-method, rather a set of configurations for a specific systems engineering method. Since he never addresses any other systems engineering methods, he does not have these conceptual needs.

4.2.2 Development Situation

A base method exists in a development environment, a business context in which each project is unique in some sense but still similar. It is for that environment we tailor the base method, thus similar projects are grouped into development situations. In order to discuss in terms of a new development situation we need at least one project, existing or future, that has unique characteristics (discussed in Section 4.2.3). We can be even more precise in defining a Development Situation:

**Definition 4-3:** A Development Situation is an abstraction of one or more existing or future software development projects with common characteristics.

Projects are temporary constellations of actors from different parts of an organization or different organizations. One example is the meta-method project team at Volvo IT. It included actors from Volvo IT and the VITS research group. Furthermore, projects have defined goals to achieve within a budget and a specific timetable. With our interest in the system engineering discipline, the scope is mainly set on an information system as the final artifact. However, the final artifact is not the project rather the process has to be included as well (Archibald, 1992). We have chosen to define a project as:

**Definition 4-4:** A project is a complex undertaking with a defined objective, performed during a demarcated duration of time, made up of interrelated activities performed by assigned actors, having budgeted resources.
With this view of projects, a development situation has to be an abstraction of both the final artifact as well as the process. Consider the following simplified scenario: An organization needs a booking system implemented as an Internet-solution. This software is supposed to support a new business, hence the organization does not have an established customer-base. Furthermore, this software becomes business-critical, because without the ability to book via the Internet there is no possibility to book it at all, since the intention is to offer no manual booking (face-to-face or telephone). A development situation could have the following characteristics: First, the final artifact is an Internet-based software artifact. This type of artifact imposes certain technologies, which could differ in maturity. Consider that in this specific case, a mature technology is used, however the development team have never used it before. Furthermore, the time is crucial since the artifact is business-critical, which should affect the development process. For example, one alternative could be rapid development. Consequently, this development situation could be summarized as an Internet-solution that is business-critical, with a low technical risk but with a high competence risk. Reading Archibald (ibid.), we find that each project is unique, still they have things in common and we could certainly find several projects with the profile described above.

Making abstractions of projects is not a new idea in the method engineering domain. Kumar and Welke (1992) use the concept development situation when tailoring systems engineering methods for various project conditions. They exemplify with project conditions such as different stakeholders, corporate culture, and systems with different complexity. This means that conditions for system development projects can vary within an organization as well as the between organizations if they use the same systems engineering method. In MMC the concept has, so far, had a more narrow focus. The concept is delimited to differences between projects within one organization. However, this probably arises from the fact that the industrial partner in the case study mainly performs in-house software development. It is possible to agree with Kumar and Welke’s (1992) use of the concept if we search for project situations outside the organization in question. The main reason then is to gather knowledge about projects that have not been performed within the organization yet, but will be in the future. Thus, the difference is related to operationalization of the characteristic concept, and not as much a conceptual issue. However, it is important when one compares our characteristics (see Section 4.2.3 for the definition) with existing research in the method engineering field.

Hares (1992) allocates projects to environments and subsequently environment is a concept similar to development situations. Referring to our discussion in Section 3.3.1, projects are abstracted through the use of characteristics in three categories: concepts, technologies, and techniques. We interpret the concept environment to

---

58 In Karlsson et al. (2001a) an extensive characterization of the Internet-based software artifact can be found. The characterization is performed using an operationalization of analytical framework based on activability (see Agerfalk, 1999; Agerfalk et al, 2002).
have a broader scope than our development situation. Hares’ generic category ‘central processing’ could contain projects with very different characteristics. However, the concept as such could be used with a finer level of granularity. A more significant difference is Hares focus on the final artifact, which determines the environment. We propose a wider concept in that sense, one that does not solely focus on the artifact, rather considers the project situation, which contains the artifact as well as an assessment of the organization.

We introduce the development situation-element using XML in order to represent a development situation. This element has two attributes, a name attribute for identification and a ctid attribute which references a configuration template-element (see Section 4.2.5). The development situation-element contains one or more common characteristic-elements, which reference a characteristic (see next section) through the name-attribute. A selection-element is enclosed in each common characteristic in order to reference a specific characteristic’s value. Several common characteristics with selected values represent an exact characterization of the abstracted projects.

Based on these elements, a development situation could be represented as below. In this specific case the development situation consists of two common characteristics, knowledge about business objectives and degree of new functionality. We will discuss these characteristics as well as the referenced configuration template in detail later in this chapter.
4.2.3 Characteristics

As exemplified above, each project has a profile and common profiles are abstracted into development situations. The differences in development situations are differences in their characteristics. For instance, projects might differ in the power the development team could exercise on the user group (Karlsson, 2001b). Building e-commerce software versus time-report software for in-house usage differs on this point. In the former situation, developers must act cautiously, since the user’s cost for switching to another e-commerce site is low, just one click away. However, with the time-reporting system there might not be any other options for the users and the decision might be taken that the system is to be used. We can define a characteristic as:

**Definition 4-5:** A characteristic is a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle.

![Diagram](image)

**Figure 19.** The concepts and relationship between our focal area, systems engineering method’s rationality, and the abstraction of projects.

The characteristic-element can be defined as below. The first XML-element is the characteristic as such. It could contain one or more values illustrated by the asterisk. The characteristic-element’s attribute list has one attribute, the name of the
characteristic, which is required. The second element is the value-element. It cannot contain any other elements but it has two required attributes. The first one is the value’s identification, and the second is the reference to exactly one configuration package.

```
<!ELEMENT characteristic (value*)>
<!ATTLIST characteristic name CDATA #REQUIRED>

<!ELEMENT value EMPTY>
<!ATTLIST value valueid CDATA #REQUIRED cpid CDATA #REQUIRED>
```

Carrying out a software development project does mean dealing with these characteristics, and hopefully turning them into an advantage (cf. Harmsen, 1997; Beck, 1999). Projects are complex phenomena and consequently the performance of method configuration is inherently complex. Isolating one characteristic at a time is one way to reduce complexity and if we extend the reasoning reusable configurations can be constructed for each of them. We can define patterns for each of the characteristics and address their impact on the base method. A characteristic has impact on the reusable patterns through the systems engineering method’s rationality. Each prescribed action has a purpose and a characteristic addresses the need to fulfill and the possibilities to achieve that purpose with regard to the delimited part of a development situation. Subsequently, we have a relationship between our intended focal area, systems engineering method’s rationality, and the abstraction of projects. In Figure 19, above, we have exchanged the systems engineering method concept used in Figure 18. Instead, one part of the more detailed view of methods, presented in Figure 1 (from Section 1.1.2), is utilized to illustrate this association.

A characteristic can be thought of as an enumerated type (that is, like an enum in C++), and we refer to the enumerated values as the characteristic’s dimension. A dimension is used to describe each characteristic. An example of a characteristic that we have found when applying these ideas at Volvo IT is ‘Degree of new functionality.’ The characteristic’s dimension consists of two values: ‘Existing functionality with new GUI’ and ‘Completely new system’. This characteristic seems to be frequent in Internet-based development, and especially the value ‘Existing functionality with new GUI’. This characteristic can formally be described as below. In this case, the name of the value (valueid) is shared with the name of the configuration package (cpid), which is not a necessity.

```
<characteristic name = "Degree Of New Functionality">
  <value valueid = "Existing Functionality With New GUI"
    cpid = "Existing Functionality With New GUI"></value>
  <value valueid = "Completely New System"
    cpid = "Completely New System"></value>
</characteristic>
```
The Theoretical Foundation of Method
for Method Configuration

The point of departure for adapting something in general, and a systems
engineering method in particular, to a situation is that the situation can be
characterized in some sense. Van Slooten and Hodes (1996) present a contingency
model based on contingency factors. Their model is based on earlier research on
such factors. Harmsen has, as discussed above (see Section 3.3.2.1), used these
factors in his method for selecting and integrating method fragments into a
situation-based system engineering method. ‘By assigning values to these factors a
project manager or method engineer should be able to sufficiently determine the
situation.’ (Harmsen, 1997) Thus, they shade the differences between development
projects depending on how it is related to certain characteristics. From the bottom-up
view, this provides a foundation for a uniform description of projects and
selection of method fragments.

If we temporarily go ahead of the discussion, the characteristics in MMC, or
rather the assigned value, are used to select a configuration of the base method
suitable for this part of the development situation. This is a bit like selecting a
method fragment, but the link between characteristics, its values, and the
configuration is more straightforward in MMC than for example in Harmsen’s
(1997) approach. For each value there is exactly one Configuration Package (see
next section), not a set of possible Configuration Packages, if an analogy is made to
the selection of method fragments. This is illustrated in Figure 18.

Characteristics include a wide range of success factors if we summarize the
method engineering literature (for example, Harmsen, 1997; van Slooten and Hodes,
1996). They can be deduced from the context as such, the organization or from
technical aspects of the project. This largely corresponds to how characteristics are
used in MMC. Each characteristic is used as part of a battery of questions for
describing a specific project and its need for a systems engineering method.

4.2.4 Configuration Package

The basic idea with the use of characteristics is to narrow the focus on a delimited
part of the development situation. The base method is tailored to fit the characteristic
at hand, thus using the base method as the starting point for the configuration work.
This delimited configuration is named Configuration Package.

Definition 4-6: A Configuration Package is a method configuration of
the base method suitable for a characteristic’s value.

Method configuration is about deciding whether or not prescribed actions in the
systems engineering method should be performed and to what extent. This is done
through the focus a characteristic has on the rationality of a systems engineering
method and its prescribed actions. Each characteristic addresses one or several
prescribed actions and their purposes. Thus, a configuration package is a
classification of prescribed actions with regard to how relevant these purposes are
for a specific value of a characteristic. Figure 20, below, shows these concepts and
their associations.

82
The definition of the configuration package concept using XML is found below. In this definition we have chosen a somewhat redundant operationalization in which the configuration package contains prescribed actions and in which the classifications are allocated to these elements. This is a deliberate design decision in order to achieve an improved readability of the configuration package. In addition, a configuration package could contain require-elements, which are utilized when configuration packages build on each other. The require-element takes one single attribute, the reference to the required configuration package.

![Figure 20. Configuration Package and Classification of Prescribed Actions](image)

The prescribed action-element contains a classification, and it could contain prescribed actions on a lower layer of granularity than the prescribed action in question, and artifacts and roles. The classification-element is declared in Section 4.2.4.1 and artifacts and roles are discussed in 4.2.4.2. Since a prescribed action could contain other prescribed actions on a lower layer of granularity we have captured the 'part of'-relationship between prescribed actions illustrated in Figure 20. Furthermore, the prescribed action-element has two attributes, the first one containing its name and the second one the layer of granularity.

```xml
<!ELEMENT configurationpackage (prescribedaction*, required*)>
<!ATTLIST configurationpackage id CDATA #REQUIRED>
<!ELEMENT required EMPTY>
<!ATTLIST required id CDATA #REQUIRED>
<!ELEMENT prescribedaction (classification, artifact*, role*, prescribed action*)>
<!ATTLIST prescribedaction name CDATA #REQUIRED layer>
```
The Theoretical Foundation of Method for Method Configuration

84

configurationpackage id = "Degree Of New Functionality: Existing Functionality With New GUI">

<prescribedaction name = "Analyze and Design" layer = "Discipline">
  <classification status = "Perform_reduced">A reduced of new architecture since existing Databases are used</classification>
</prescribedaction>

<artifact name = "Database Design" layer = "Activity">
  <classification status = "Skip">No new type of persistent data is developed</classification>
  <artifact name = "Change Request" type = "out">
    <classification status = "Skip">No relevant changes</classification>
  </artifact>
  <artifact name = "Data Model" type = "out">
    <classification status = "Skip">Will not have any content</classification>
  </artifact>
</artifact>

<artifact/>

configurationpackage id = "New Functionality: Newly Designed GUI">

<artifact/>

configurationpackage id = "New Functionality: Existing Functionality with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing GUI with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: New GUI with Existing GUI">

<artifact/>

configurationpackage id = "Existing Functionality: New GUI with New GUI">

<artifact/>

configurationpackage id = "New Functionality: New GUI with Existing GUI">

<artifact/>

configurationpackage id = "New Functionality: New GUI with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing Functionality with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing Functionality with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing Functionality with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing Functionality with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing Functionality with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing Functionality with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing Functionality with New GUI">

<artifact/>

configurationpackage id = "Existing Functionality: Existing Functionality with New GUI">

<artifact/>
We can exemplify with our ‘Degree of new functionality’ case at Volvo IT. However, in order to do that we need to utilize classifications which are discussed in detail in the forthcoming Section 4.2.4.1. In this example, prescribed actions related to database design in RUP are considered unnecessary, if we are dealing with ‘Existing functionality with new GUI’. Hence, the performance of these prescribed actions could be performed in a reduced form, or be skipped altogether, while other prescribed actions might be performed as suggested by the base method. The example above contains a section of a configuration package focusing on this aspect. We use ellipses in order to illustrate that the configuration package is not complete. The characteristic in question affects prescribed actions in the ‘Analyze and Design’ discipline. We have concluded that some prescribed actions are not necessary to perform, hence the discipline is performed with reduced content. The argumentation in this case is a reduced need for new architecture. The workflow detail ‘Design The Database’ is classified as ‘Skip’ together with its prescribed actions on lower layers of granularity. An example of the later is the activity ‘Database Design’ which is classified ‘Skip’ as well. Furthermore, the artifacts found in this activity are also classified, which is discussed in Section 4.2.4.2. Finally, the workflow detail ‘Design Components’ is classified perform as is, if we believe it is necessary during a project with this characteristic.

4.2.4.1 Defining Configuration Packages
Defining configuration packages plays a central part in the proposed meta-method (see Section 4.3 below). Since configuration packages mean classifications of prescribed actions we need a classification system for how the base method and supplementary parts of additional methods should be performed. Table 3 shows such a classification, which has proven useful in our work.

The classification in Table 3 is based on two dimensions. The vertical dimension illustrates how much attention should be devoted to a particular prescribed action. If we find a prescribed action unimportant we give it no attention at all, and it can be classified as a ‘Skip’. This was the case with the database design activities in the RUP-example above. The prescribed action can alternately be performed with the usual prescribed attention (perform), or we can choose to put more effort (a lot) into performing the prescribed action than prescribed in the base method (emphasize).

The three aspects of the horizontal dimension: ‘Extended’, ‘As is’ and ‘Reduced’ cut through the vertical dimension. Exemplifying with the granularity in RUP, these aspects, referred to as required enhancement, tell to what extent the various steps that make up an activity should be performed or not. If we reduce the activity, some steps are suppressed. In contrast, an extended activity means using steps not specified in the base method. These steps could be provided through integration with additional methods or through explicit method development. In this case, we are close to method engineering (e.g., Brinkkemper et al., 1999). However, in that case we propose that the integration or development should be based on rationales that support the overall goal of a development situation.
The Theoretical Foundation of Method for Method Configuration

Table 3. Performance of prescribed actions in the base method deduced from the categories of attention and enhancement.

<table>
<thead>
<tr>
<th>Attention given to prescribed action</th>
<th>Enhancement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reduced</td>
</tr>
<tr>
<td>None</td>
<td>Skip</td>
</tr>
<tr>
<td>Normal</td>
<td>Perform reduced</td>
</tr>
<tr>
<td>A lot</td>
<td>Emphasize reduced</td>
</tr>
</tbody>
</table>

The two dimensions result in a matrix classifying a prescribed action into seven possible categories as shown in Table 3. However, the possibility of reduction depends on the prescribed action in question. We have to exclude the ‘reduction’ and ‘extension’ when discussing the lowest layer of granularity. Yet again returning to RUP as an example, the granularity layer step is not possible to reduce since it is an atomic prescribed action. This is a variant of the known problem of decomposition (Langefors, 1995) Furthermore, when we consider the opposite scenario, which is extending a step, it is not possible with our meaning of extension. An extension is implemented through adding prescribed actions from a lower layer of granularity than the prescribed action in question. Yet again there is no lower layer than step, thus it cannot be enhanced. These restrictions are illustrated in Table 3 and Figure 21 through their shaped areas.

The content in Table 3 is the base when defining our classification-element, which was briefly introduced in the previous section. We use the classes in Table 3 as possible selections for the status attribute. Consequently, this classification-element could be used together with each element that is to be classified in the systems engineering method. Each classification should contain an argument about why a standpoint has been taken, and this possibility is represented by the #PCDATA in the element below.

```xml
<!ELEMENT classification (#PCDATA)>
<!ATTLIST classification status (Skip|Perform_reduced|
Perform_as_is|Perform_extended|Emphasize_reduced|
Emphasize_as_is|Emphasize_extended) "" >
```

In order to classify prescribed actions according to the scheme above, we propose to use the analytic tool shown in Figure 21. With this tool, the classification of prescribed actions is based on the combination of the importance and the complexity of the area of concern that the particular prescribed action addresses. If the complexity is low, we may consider reducing the prescribed action. If it is high, it might need to be extended. If it is unimportant, we may consider skipping it. Finally, if it is very important, it might require emphasis. Note that the center of the cross in Figure 21, which represents the ‘normal case’ of the base method, might
slide within the importance and complexity space depending on what is ‘normal’ in
the current organization; diverging from the normal situation means moving along
the Importance-axis or the Complexity-axis.

Classifying a prescribed action as ‘Skip’ naturally means that we do not have to
decide whether to perform it ‘Reduced’, ‘As is’ or ‘Extended’. A ‘Skip’ is always a
‘Skip’. For example, a prescribed action addressing an irrelevant area of concern for
the development situation, or more concrete for a project, should not be performed.
The other extreme, an area of concern that is more important than usual, might
require an emphasized process. This movement can be combined with movement
along the complexity axis. In the case in which the area of concern was irrelevant,
prescribed action can also be a complex issue to grasp. Then, performing the
prescribed action could be both time and cost consuming and consequently such a
prescribed action could be considered as a ‘Skip’. With reduced complexity, it is
even more intuitive to map a prescribed action as a ‘Skip’. If the situation at hand is
more complex than usual and is central to the success of the project, it is natural to
consider emphasizing the prescribed action. A consideration should also be made if
there is a need to extend the base method or if it can be handled this prescribed
action on an ‘As is’ basis.

Figure 21. Suggested ways to perform the different prescribed actions of the base method derived from
the importance and complexity of the areas with which they are concerned.

A prescribed action classified as ‘Perform’ can be carried out with a reduced,
normal or extended number of prescribed actions on a lower level of granularity.
The Theoretical Foundation of Method
for Method Configuration

Thus, ‘Perform as is’ is the normal ‘off-the-shelf’ case of the base method. The ‘Emphasize reduced’ means reducing, for example, the number of steps in a RUP-activity, but that the remaining steps are more important than usual. Consequently, more effort will be put into the remaining steps than in the normal case. Consider, for example, an activity involving refinement of two artifacts during two separate steps, where the step concerning the first artifact should not be performed (subsequently a reduction of the activity), but where the performance of the second step is crucial (and subsequently deserves more attention and should be emphasized). This way of thinking about the Emphasize categories applies to the remaining two attention categories as well; the performed steps are more important than usual, irrespective of whether they are part of the base method or not.

4.2.4.2 Consequences of Classifications

Referring to our discussion about the concept of method (in Sections 1.1.2 and 3.2) we can conclude that classifications of prescribed actions have consequences for artifacts and roles. The input and output associations between prescribed action and artifact and the association between prescribed actions and roles in Figure 22 represent the reasons.

Consider a certain characteristic, which focuses on knowledge about business objectives during the development process. One option (value) could be known business objectives indicating that business modeling might be unimportant. Thus, in this case the purpose of the business modeling discipline in RUP is not interesting to fulfill and all prescribed actions within this discipline are classified as ‘Skip’. This would also mean that artifacts as well as roles related to this discipline are classified as ‘Skip’. For example, the artifact ‘Business Vision’ in RUP will not be produced.

Figure 22. Possible cascading effects form classifications of prescribed actions owing to of the relationship between the artifact, the prescribed action and the role concepts.
and the role ‘Business-Process Analyst’ will not be performed. Thus, these parts of
the base method are indirectly made situational, through the use of the systems
engineering method’s rationality.

The artifact and role elements are defined as below. Both have a name attribute
identifying specific artifacts and roles. The artifact element has an additional type
attribute to capture whether the artifact is an input and/or output artifact with regard
to the prescribed action. Consequently, this attribute could contain the values in, out,
in-out. Furthermore, both these elements are possible to classify using the
classification element discussed in Section 4.2.4.1.

```xml
<!ELEMENT artifact (classification)>  
<!ATTLIST artifact name CDATA #REQUIRED type (in|out|in-out) #REQUIRED>  

<!ELEMENT role (classification)>  
<!ATTLIST role name CDATA #REQUIRED>
```

One interesting aspect of this classification schema is the possibility to support
method adaptation in addition to method configuration. If we consider a demarcated
part of a method description it does not differ if it is classified as ‘Perform’ or
‘Emphasize’ on the attention dimension. Rather, this is a recommendation on how to
adapt the systems engineering method when it is applied. Subsequently, it tells the
method user something about the importance of reaching the sub-goal and allows us
to improve the rationale of the systems engineering method. Of course, if we return
to our discussion about method engineering, configuration and adaptation in Section
3.3 this is only one part of how systems engineering methods could be adapted
during use. However, we propose that more complex changes to the base method are
incorporated through method configuration in order to externalize situational
versions of the base method and hence spread experiences.

4.2.4.3 Yet Another Modularization Concept?
The meaning, or rather the operationalization, of the configuration package concept
has changed during the development of MMC. By way of introduction in the meta-
method project team this concept constituted a configuration of, or a standpoint on,
every prescribed action in the base method and how they should be performed with
regard to the characteristic at hand. During operationalization, this concept has been
reduced to solely involve prescribed actions that are affected by the characteristic.
However, there is no difference of principle, since prescribed actions not affected by
the characteristic could be regarded as configured in accordance with the base
method.

Operationalized in this way, the configuration package will in some sense
resemble the concepts method components and method fragments in the method
engineering literature. A method component is one or more prescribed actions with
concepts and notation (Karlsson, 1997). Consequently, it is a delimited set of
prescribed actions that are found in a method component. Kumar and Welke (1992)
define a similar concept with the 'methodology component', which again is a difference in the use of the method concept (see Section 3.1.1). A methodology component is a predefined and pre-tested part of a systems engineering method that focuses on a problem area. As discussed earlier, Harmsen (1997) as well as Brinkkemper (2000) use the method fragment concept as the foundation for construction of a situational systems engineering method.

Referring to the discussion from Chapter 3, a method fragment is a delimited description of a systems engineering method or a logical coherent part thereof. Furthermore, method fragments can be of different sub types, either a complete process, one part of a process or a set of products. Method fragments also exist on different layers of granularity. (Harmsen, 1997) Ågerfalk (1999) has related method fragment concept and method component concept to each other. A method component consists of a ‘meaningful assembly of process and product fragments and the concepts used within those fragments’. We have borrowed the idea of layer of granularity, but use it slightly different in MMC. One configuration package could contain classifications of prescribed actions on several different layers of granularity, however we do not say that the configuration package rests on a specific layer.

Actually, we do not have a need for another basic modularization concept used for systems engineering methods. We have two concepts for decomposing methods, which can be related to the configuration package. It is possible to reuse existing concepts, however they do not emphasize the nuances in our usage. If anything, these nuances are founded in how we derive the content of a configuration package, not an objection to the basic decompositions of a systems engineering method. A configuration package is a set of method fragments classified using the characteristic as a screen. This set of method fragments is a logical coherent part of the base method with regard to certain goals the characteristic emphasizes. Consequently, it could also agree with a method component in the sense that it is a ‘meaningful assembly’ (Ågerfalk, 1999) of method fragments. Such an assembly can only be meaningful with regard to specified goals. However, we do not want to use the concept method component since we interpret it as a section of or a complete systems engineering method, which could be detached from the original method context. A configuration package is not intended for use outside the base method context.

Furthermore, configuration packages does not have any distinction between different types of method fragments (Harmsen, 1997). Since processes and products are intertwined with each other a configuration of either part indirectly affects the unfocused part. Brinkkemper et al. (1999) state that 'each product fragment should be produced by a “corresponding” process fragment’. Consequently, in order to create or modify a product fragment, at least one process fragment is needed. This line of reasoning can also be described the other way around; a process results in at least one deliverable, since prescribed actions activities are performed for reasons (Ågerfalk and Åhlgren, 1999). Otherwise, that part of a systems engineering method cannot be justified. Configuration packages include both a configuration of the process and the product. For example, if an analysis of the activity Design Database,
in RUP, is considered as ‘Skip’ its resulting artifact Database Design is also considered as ‘Skip’.

A configuration package also holds information about the base method, which a method fragment by nature does not need. It includes information on how the base method is changed. For example, if the base method should be simplified in order to speed up the development process such a configuration package probably contains prescribed actions classified as ‘Skip’ or ‘Reduced’. Thus, this particular configuration package primarily aims at reducing the content in the base method. This differs from the aim of a method fragment, where we add prescribed actions or artifacts to the situational systems engineering method, in other words expanding the size. Furthermore, every method fragment seems to have the same importance during the process, which is not the case when using configuration packages.

However, a configuration package could include the information held in a method fragment, illustrated by the following. RUP has a discipline describing project management (Kruchten, 1999). If we want to exchange this discipline, or some part of it, for the benefit of an in-house developed project management discipline, this information must be contained in the configuration package. The outline for performing this is a configuration in two steps. First, we have to analyze which prescribed actions and artifacts should be exchanged, in other words classified as ‘Skip’. Thereafter, we have to add the new prescribed actions and artifacts. The first part is in analogy with our discussion above where the configuration package holds information about reduction. In the second part, a configuration package equates with a method fragment, or method components for that matter, in the sense that it adds prescribed actions and/or artifacts to the situational systems engineering method (or more correct the configuration template which is introduced in next section).

4.2.5 Configuration Template

Of course, the world is not simple enough to be caught through one single characteristic. Projects are abstracted into a development situation involving several characteristics. Hence, one single configuration package will in most cases not be sufficient. This is only natural if we refer to the definition of the configuration package concept. Therefore, we need a combination of characteristics, and subsequently configuration packages, to capture a development situation. The configuration template concept does all that (see Figure 23 below), which we have chosen to define as:

**Definition 4-7:** A Configuration Template is a combined method configuration for a set of recurrent project characteristics.
This can be thought of as a predefined combination of configuration packages common within the organization. The use of, and need for, configuration templates is a result of our empirical work with MMC at Volvo IT. Creating and rebuilding a project specific Development Case is possible since it is derived from configuration packages, as long as the configuration packages it is derived from are not changed or overlapped. However, this is not an efficient use of resources. Working with configuration packages only means starting at a lower configuration level than necessary, and subsequently more work is needed in the start-up phase of each project. Configuration packages is one operationalization of reusability in MMC, configuration templates another. Furthermore, configuration packages have a second objective, to facilitate a focused analysis together with the characteristic concept.

At least not without tool support
Thus, since projects within a development situation share the same profile then they could efficiently share a configuration template.

However, more important is the fact that configuration templates provide an opportunity to base complete method configurations on experience gathered during earlier projects. Configuration packages have several possible combinations, which are increased with the amount of configuration packages. There is a selection of the most interesting ones for a specific organization and where the projects typically have certain differences with respect to the corresponding characteristics. This selection has to be based on experience of their frequency and differences in characteristics. Furthermore, combining configuration packages cannot be fully automated. There are situations in which the method engineer has to take a position on a prescribed action and its results, decisions that cannot be represented within a configuration package. For example, configuration packages could cover the same part of the base method, which in turn can cause conflicts between their different classifications. Consequently, we need a container to represent this aspect in our meta-method. Furthermore, the same type of conceptual construct as between configuration package and prescribed action is needed to capture this combined classification of prescribed actions. Thus, we have chosen to introduce an association class (Classification) between the configuration template and the prescribed actions in Figure 23 (above). These classifications follow the classification system in Table 3, but each classification could either be derived from a configuration package associated with the configuration template in question, or be a reclassification caused by conflicting configuration packages.

The configuration template element is defined below. We have used the selected element to capture the aggregate of combined configuration packages. Each selected element is a reference to a selected configuration package, and the asterisk represents the possibility to combine several configuration packages. Furthermore, we have to capture the classification relationship between the configuration template and the relevant prescribed actions. Hence, we have introduced the combination element. The combination element contains one or more prescribed actions. Referring to the definition of the prescribed action element we see that it contains a classification and we can capture classifications made for the configuration template.

```xml
<!ELEMENT configurationtemplate (reoccurentcharacteristic*, combination)>  
  <!ATTLIST configurationtemplate id CDATA #REQUIRED>

<!ELEMENT reoccurentcharacteristic (value*, selected)>  
  <!ATTLIST characteristic name CDATA #REQUIRED>

<!ELEMENT selected EMPTY>  
  <!ATTLIST selected id CDATA #REQUIRED>

<!ELEMENT combination (prescribedaction*)>
```
The Theoretical Foundation of Method

**For Method Configuration**

```xml
<configurationtemplate id="Web Publishing">
  <reoccurentcharacteristic name="Knowledge about Business Objectives">
    <selected id="Known"/>
  </reoccurentcharacteristic>
  <reoccurentcharacteristic name="Degree of New Functionality">
    <selected id="Existing Functionality With New GUI"/>
  </reoccurentcharacteristic>
  ...
  <prescribedaction name="Design The Database" layer="Workflow_detail">
    <classification status="Skip">New or updated databases will not be implemented</classification>
    <prescribedaction name="Database Design" layer="Activity">
      <classification status="Skip">No new type of persistent data is developed</classification>
      <artifact name="Change Request" type="out">
        <classification status="Skip">No relevant changes</classification>
      </artifact>
      <artifact name="Data Model" type="out">
        <classification status="Skip">Will not have any content</classification>
      </artifact>
    </prescribedaction>
    ...
  </prescribedaction>
  ...
</configurationtemplate>
```

---

The combination of new functionality with new GUI is selected. The database design is skipped due to new functionality is not developed. However, the Data Model will not have any content.
As an example of a configuration template, consider the case in which a legacy system is to be transformed into a Web application. One of the characteristics to consider for these types of projects is the degree of new functionality that is needed in this application; another characteristic could be knowledge about business objectives. The configuration template can be formally described as above. The ellipses illustrate that the example only contains sections of a possible configuration template.

In this example, we have chosen the dimension value ‘Existing functionality with new GUI’ as the configuration package from which to decide the performance of the base method’s prescribed actions. Note that there will be a statement of chosen value (with corresponding configuration package) for each characteristic identified in the organization, replacing the ellipses. This assumes that ‘not applicable’ is an implicit generic value possible for all characteristics, since there are obviously situations in which some characteristics are totally irrelevant. The combination of configuration packages into a configuration template is, at least partly, based on heuristics, thus we need to document the classification of the ‘combined’ prescribed actions as well. We do this with the same notation as the classification of activities of a single configuration package. The configuration template can later be reused for fine-tuning into a situational systems engineering method (see Figure 24 above). The reason for fine-tuning is the small individual differences that can be found for each project. Furthermore, there is a conceptual difference between the configuration template and the situational systems engineering method since MMC addresses the descriptions of systems engineering methods. The later is the systems engineering method used by the project members, thus it includes method adaptation. We define the concept Situational Systems Engineering Method as below, which is a modification of Definition 1-14:

Figure 24. Situational Systems Engineering Method as a Selection from Configuration Templates

In this example, we have chosen the dimension value ‘Existing functionality with new GUI’ as the configuration package from which to decide the performance of the base method’s prescribed actions. Note that there will be a statement of chosen value (with corresponding configuration package) for each characteristic identified in the organization, replacing the ellipses. This assumes that ‘not applicable’ is an implicit generic value possible for all characteristics, since there are obviously situations in which some characteristics are totally irrelevant. The combination of configuration packages into a configuration template is, at least partly, based on heuristics, thus we need to document the classification of the ‘combined’ prescribed actions as well. We do this with the same notation as the classification of activities of a single configuration package. The configuration template can later be reused for fine-tuning into a situational systems engineering method (see Figure 24 above). The reason for fine-tuning is the small individual differences that can be found for each project. Furthermore, there is a conceptual difference between the configuration template and the situational systems engineering method since MMC addresses the descriptions of systems engineering methods. The later is the systems engineering method used by the project members, thus it includes method adaptation. We define the concept Situational Systems Engineering Method as below, which is a modification of Definition 1-14:
Definition 4-8: A Situational Systems Engineering Method is a fine-tuned and adapted Configuration Template used in a project.

The usage of templates in method engineering literature is not new. One example is the discussion about paths through a systems engineering method (Harmsen, 1997; Odell, 1996). Other examples are method fragments and method components that are templates to be retrieved and combined into a situational systems engineering method. If configuration templates are to be compared with method fragments, it is with method fragments on the method granularity layer. Consequently, a configuration template is the template for a complete situational systems engineering method. Brinkkemper (2000) presents the Baan Development Method and uses the concept Project Model. This is a concept similar to ours, with a description of the deliverables to be produced by work instructions. The Web-enabled system for method engineering seems to have project models available, and hence they are templates for situational systems engineering methods.

4.3 The Framework
The concepts described above are a structural view of MMC. Based on this view we are able to present the framework for MMC. Since we want to obtain reuse we need to have different using frequency of the operationalized concepts described above. Therefore the MMC-framework in Figure 25 is divided into three sections with regard to the occurrence frequency they have during configuration work.

The most frequent section is tuning a configuration template into a situational systems engineering method (Section 3). The second section with regard to occurrence is combining existing configuration packages into a new or modified configuration template based on a different selection of configuration packages (Section 2). Thus, this new configuration template has a different profile if we examine the configuration packages closely. The third and least frequent section is the administration of specific configuration packages (Section 1). It involves creating a new or modified configuration package in order to meet a new project characteristic that has not been considered in earlier projects.

Let us change focus and study the meta-method as we would when using it for the first time. Then, we have to elaborate at least one configuration package. The starting-point is the implemented base method. The procedure for choosing a base method is not illustrated in Figure 25 since it is outside the scope of MMC. How many configuration packages we need to work with initially depends on the development situation and characteristics we identify. Abstractions are made in ‘Identify Development Situations and Characteristics’ based on the organization’s historical and future project portfolio. The result is stored in the ‘Base for Development Situations and Characteristics’ and is used as input for the analysis of configuration packages, where dimensions of each relevant characteristic determine the configuration of configuration packages. This is done in ‘Administration of Configuration Packages’ which also involves following up projects and using these
experiences for changing configuration packages. The pre-made method configurations are stored in the ‘Base of Configuration Packages’.

Figure 25. The Method for Method Configuration Framework.

As stated in Section 4.2.5, few, if any, projects can be found with only one single characteristic that affects the total configuration of the situational systems engineering method. Thus, a project will involve several configuration packages,
The Theoretical Foundation of Method
for Method Configuration

and so will a configuration template. The section for creating a configuration template (Section 2) consists of two blocks of activities, ‘Selecting Configuration Packages’ and ‘Combining Configuration Packages’. The selection of relevant configuration packages for a configuration template is based on a development situation’s characteristics, thus it is the base for the activity ‘Selecting Configuration Packages’. Combing configuration package involves the tasks for solving classification conflicts between different configuration packages and an evaluation of the template’s consistency. The result is stored in the ‘Base of Configuration Templates’.

Starting up a specific project involves, from a method configuration point of view, an investigation of the project situation. ‘Identifying Project Characteristics’ represents this task in Figure 25. The analysis has to be focused in order to classify the project against characteristics of configuration templates. The characteristics of the specific project can be represented as an array in the same manner as a configuration template, in order to create a profile. If a matching configuration template is found, and we are using RUP as the base method, the template is the base for tuning a project specific Development Case and thus a situational implementation of the base method. This is illustrated by ‘Matching Project and Configuration Template’ in Figure 25.

The resulting situational systems engineering method is used during the project. However, the competent method user will continuously adapt the method, thus we have two arrows between the situational systems engineering method and the project. They illustrate used in and affected by, respectively. The project will result in at least one artifact and experience. From the method configuration perspective, it is important to have feedback based on these experiences. Otherwise, the process becomes linear and it will not be possible to improve the configuration packages, configuration template, and most important of all, future projects.

The presented framework has similarities with established methods for generating situational systems engineering methods. As presented in Section 2.2.3, Research Process, method engineering domains have influenced our method. We have been influenced when constructing the framework as well as the conceptual level. Using repositories for storing reusable resources for future retrieval and assembly of situational systems engineering methods are common. See, for example, Harmsen (1997), Brinkkemper (1996) and Oinas-Kukonen (1996). Since we are working with two different concepts for reusability there are two different repositories in our framework for this purpose.

However, differences in the frameworks can be found. Our starting-point with a specific base method differs from building each situational configuration upon method fragments. Since we have a base method as a shell, the team members can build their core competency around this base method. Furthermore, since we use configuration packages and configuration templates, we start at a more aggregated level compared to method fragments. Through an accumulation of experiences, the organization will generate a more precise selection of configuration templates and thus manage method configuration more efficiently. A configuration template does not have to involve a new configuration package, just another selection among them.
Thus, with recurrent development patterns in an organization, the configuration process should become more efficient. The use of configuration templates also yields a better foundation for managing project experience.

4.4 Summary

In this chapter, we have elaborated on the theoretical foundation of MMC. We started out with a discussion about the main values behind MMC. The use and value of methods as support for information system development projects was discussed along with our starting-point in a single systems engineering method. We have addressed the standpoint that method configuration is viewed as an investment. Furthermore, we have discussed our view on flexibility and how flexibility is restricted through the use of standard system engineering method. Finally, our fifth value, rationality, has been discussed.

In Section 4.2 we presented a simplified conceptual model of MMC as an initial overview. It consisted of the basic concepts used in MMC, which have been defined in this chapter. As a complement, we have used demarcated conceptual models to show the evolvement of our simplified model. These models have been merged and the result can be found in Figure 26 below, illustrating the main concepts and relationships of MMC. Compared with the simplified model, we have elaborated on the relationship between the MMC specific concepts and the constituents of a systems engineering method. The latter were introduced in Section 1.1.2. Through these constituents of a method we have become more explicit on how to use the systems engineering method’s rationality as the focal area for method configuration.

Characteristics address the purpose of prescribed actions facilitating an analysis of a certain problem or aspect in the development situation which the method configuration aims to solve or handle. The relationships between configuration package and systems engineering method as well as configuration template and systems engineering method have been made explicit as a classification of prescribed action. Furthermore, we have improved the semantic relationship between configuration package and configuration template as well as between characteristic and development situation through the use of aggregates.

However, MMC is just not a conceptual model. One important result from this chapter is the analytical tool and the classification system through which analysis of the systems engineering method’s rationality is made possible. Consequently, we have the possibility to classify prescribed actions and indirect artifacts and roles, and hence make them dependent on the business context. These classifications have been possible to represent through the use of XML and the elaboration of XML elements. Finally, we have proposed a framework divided into three sections. These sections are used with different frequency, which is important when working with reuse.
The Theoretical Foundation of Method for Method Configuration

Figure 26. Main Concepts of MMC
5 THE RUP SHELL

This chapter is the first of three chapters in which we will discuss the operationalization of MMC. RUP is the method used in the forthcoming operationalization of MMC as well as meta-method usage (see Chapters 6 and 7). Before discussing the details we examine the MMC framework and how to make it applicable with the base method we are using. Therefore, we will initially describe a few basic concepts in RUP, and the notation we will use for structuring the presentation.

5.1 The Concepts in Rational Unified Process

RUP is an instance of method among others. Nevertheless, it has some unique characteristics and concepts that have to be considered. Despite the fact that our principles for method configuration should apply to a wide range of base methods, these concepts and their relationships become important to harmonize with the concepts and structure in RUP. Therefore, some key concepts in RUP, central to method configuration, are discussed below.

RUP is an architecture-driven approach with an early focus on risks. It is important to settle a baseline for the architecture and through incremental development work on the highest risks. If we return to the layers of granularity, described in Table 1, we have five different concepts for guiding the performance of prescribed actions. These concepts have their representation in RUP: process\(^40\), discipline, workflow details, activities, and steps (see Figure 27 below). These different method constructs describe the systems engineering method with different precision and subsequently have a focus shifting from overview to details. This is useful when different workers use the systems engineering method with different purposes. For example, a worker, such as the business analyst, performs a delimited

\(^{40}\)Rational Software uses the process concept for the highest level of granularity where we have chosen the method concept. However, since we present Rational’s concepts in this section and Figure 27 we will use the process concept temporarily.
The RUP Shell

task in the project and uses the activity level, while a project manager might use the workflow detail level for planning. Furthermore, we can conclude that the worker and role concepts have a relationship. In RUP, the worker concept is used as a representation of a role or set of roles in the business.

Figure 27. Basic Concepts in RUP

Discipline is the concept that gives the overview of a delimited part of the process; it is a sequence of activities showing possible paths through a configuration (and the standard RUP if we treat that as a configuration). Within disciplines, we find workflow details, which show activities often performed together. Furthermore, workflow details are performed in an iterative pattern through the use of phases.
RUP has four phases: Inception, Elaboration, Implementation, and Transition. Workflow details are visualized in diagrams or tables showing the workers, the main input and output artifacts, and the activities performed. One or more workers modify or use the artifacts according to the descriptions in the workflow detail (see Section 5.4). Workers modifying an artifact can differ from the worker who is actually responsible for it. Therefore, there is both a modifying relationship between worker and artifact as well as a responsible relationship. The activity provides a meaningful result, an artifact or refinement of an artifact, to the project and consequently has a clear purpose. Each activity is then finally broken into a couple of steps, which are precise descriptions about how to perform a delimited part of a task. These concepts are illustrated in Figure 27.

Activities have supporting artifacts. Attached to each activity we find specific key concepts. With support from these concepts, the performed activity receives a certain focus. Thus, we can relate patterns to how concepts are used within a method fragment, described above. A described activity contains a number of steps, which are context-free descriptions. However, activities are performed in contexts, and they differ slightly depending on the situation. Therefore, guidelines are attached to activities, steps, and artifacts. A guideline is an artifact sub-type, containing descriptions about practical considerations. For example, guidelines could be tool documentation or proposals for how to start-up a storyboarding session. Such issues are not covered in an activity description.

Each of these concepts is directly or indirectly affected by method configuration. Since delimited parts of the method are modified in one way or another their surrounding artifacts, workers, and guidelines are affected (see Section 4.2.4.2). For example, suppressing the activity ‘Design Real-Time Component’ probably means that the resulting artifacts ‘Design Class’ and ‘Design Subsystem’ should not be produced. The situational systems engineering method, or instance of RUP, is documented in a Development Case.

5.2 Layer of Granularity

Until now, we have discussed ‘activities’ and ‘steps’ of the base method in a rather intuitive fashion. In this section, we will be more specific by following the above-mentioned classification of method fragments into layers of granularity and perspective (as suggested by Brinkkemper et al., 1999). Table 1 shows the mapping of RUP concepts in a process perspective into the ‘layer of granularity’ hierarchy.

<table>
<thead>
<tr>
<th>Layer of Granularity</th>
<th>RUP from a process perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>RUP</td>
</tr>
<tr>
<td>Stage</td>
<td>Core Workflow</td>
</tr>
<tr>
<td>Model</td>
<td>Workflow Detail</td>
</tr>
<tr>
<td>Diagram</td>
<td>Activity</td>
</tr>
<tr>
<td>Concept</td>
<td>Step</td>
</tr>
</tbody>
</table>
Choosing the appropriate layer of granularity for the analysis and the content of a configuration package is an important decision to make. Not because of the configuration package’s construction as such, since a single configuration package could contain a mixture of configured prescribed actions on different layers, rather the choice is a recommendation for where to start the analysis. The method engineer could then move upwards in the layer structure, if possible, or downwards, if necessary.

This must be done with an accurate balance between precision and cost (cf., ter Hofstede and Verhoef, 1997). Choosing concept as the unit of analysis will yield high precision, but might also yield intractable amounts of work and data produced. At the other extreme, stage, the precision is probably too low to be useful. In our empirical work at Volvo IT, we have tried to use the model layer as the unit of analysis. However, when using RUP as the base method, it has proven difficult to present the results from method configuration at a workflow detail level. This is due to the way the RUP suggests we represent the configured method in a Development Case. It has not been possible to show what results are produced during the workflow detail, why they are produced, and what effects they have on other parts of the method. The latter aspect is very important to keep in mind. If a part of the method is classified as ‘Skip’, then it can, and very likely will, have effects on other parts of the method. The most evident case is artifacts not produced that later are presupposed as input to other parts of the method. In such a case, there will be a domino effect, where later parts of the process are necessarily classified as ‘Skip’ or ‘Perform reduced’. Alternately, these latter parts of the method might be considered important, implying that preceding parts can actually not be excluded.

The diagram layer (i.e., the activity level in the RUP) gives us the possibility to manage this situation with an acceptable balance between precision and cost. Each activity has a pragmatic purpose and the results are presented through the artifact list. In RUP, artifacts used as input to an activity and the activity’s resulting artifacts are visualized in artifact lists in Development Cases, showing how the activities are intertwined. However, one should be aware that by choosing this level, details on exactly how, and what parts of the results are used in following activities, are lost.

### 5.3 The Framework in a RUP Shell

MMC can be operationalized to fit a wide range of methods. The natural choice when operationalizing the meta-method is to do it through the chosen base-method. In our case, we are working with RUP, and hence it becomes the natural choice. The main target group of this thesis is method engineers since we want to improve the support for this role. In the forthcoming chapters this role is specialized into the RUP’s process engineer. With the aim of presenting a suitable meta-method for this role, we have to put MMC into the RUP context discussed above. Consequently, we have chosen to speak of MMC as a separate discipline. Following the presentation of

---

41 Of course, Method would be the real extreme. However, whether or not to perform the method at all is another type of discussion that should be settled before implementing the method in the organization. Otherwise, MMC is a waste of resources.
RUP concepts, above, we see that a discipline has workflow details, activities, and steps. Therefore, we have chosen to operationalize, and present, a version of MMC based on this structure.

As discussed in Section 4.3, there are three possible starting-points when working with MMC, illustrated by the three sections. Therefore, it should be possible to take at least three different paths through the meta-method. The top-most diamond in Figure 28, above, illustrates this choice. Depending on the needs of the
initial request we might chose the right, middle or left path. Section 1, in Figure 25, is operationalized into two workflow details: ‘Manage Change Request for Configuration Package’ and ‘Define Configuration Package’. In the first workflow detail, a decision is made whether or not a new or modified characteristic, and accompanying configuration package, are needed. Thus, this is a mapping of the ‘Identifying Development Situations and Characteristics’ in Figure 25. The workflow detail ‘Define Configuration Package’ corresponds to ‘Administrating Configuration Packages’ in Figure 25 involving activities for creating and updating configuration packages based on experiences from earlier projects. Section 1 in the general MMC framework includes two repositories (see Figure 25 in section 4.3), which are operationalized as artifacts in this version of MMC. ‘Base of Development Situation and Characteristics’ and ‘Base of Configuration Packages’ correspond to the artifacts, ‘Characteristic List’ and ‘Configuration Package’ respectively. These are not illustrated in Figure 28 but are found in the workflow details presented in Chapter 6.

‘Selecting Configuration Packages’ and ‘Combining Configuration Packages’ in Figure 25 are operationalized into the middle path in Figure 28. However, as illustrated in Figure 25 there is an indirect link between ‘Identifying Development Situations and Characteristics’ and the content of MMC-section 2. Some parts of this structure have been incorporated into the workflow detail ‘Manage Change Request for Configuration Templates’. In this part of the method, we decide whether or not a new or modified configuration template is needed, and such a decision has to be based on information about development situations. Furthermore, if the change request results in the decision to create a new or modified configuration template this work is performed in the workflow detail ‘Define Configuration Template’. This workflow detail is an operationalization of ‘Selecting Configuration Packages’ and ‘Combining Configuration Packages’ in Figure 25. The latter results in an updated ‘Base of Configuration Template’, which is operationalized as artifacts. We will refer to this base as ‘Configuration Template’ artifacts in Chapter 6.

The third section in Figure 25, is supposed to be the most frequent used one, selecting an existing configuration template. The basic idea is to reuse the same characteristic to characterize the specific project as used when selecting configuration packages for a configuration template. The activities ‘Identifying Project Characteristic’ together with ‘Matching Project and Configuration Template’ in Figure 25 are transformed into the workflow detail ‘Select Configuration Template’ and ‘Fine-tune Configuration Template’ in our RUP version of MMC. Since we receive a template of a systems engineering method from the selection we have added a workflow detail for fine-tuning the template into a situational systems engineering method. The foundation for this workflow is mainly based on the same principles as for defining a configuration package and/or template. Thus, it is based on the classification system for configuration packages, presented in Section 4.2.4.1. The main difference is that in this workflow detail we should only perform minor adaptations. In those cases when we have major, recurring, adaptations this might be an indication that a configuration template is missing, since we do not obtain a
Meta-Method for Method Configuration

sufficiently good match between the project situation and the existing configuration templates.

Hence, we must be able to handle the lack of configuration templates or configuration packages. Therefore, we are able to trigger the workflow detail ‘Manage Change Request for Configuration Template’ and ‘Manage Change Request for Configuration Package’ when there is need for a new version of one of the two or both. Of course, we could regard this situation as a completely new configuration and start with a change request, but with the link between the workflow details the design becomes more explicit. Thus, if we do not find a matching configuration template during the ‘Select Configuration Template’ workflow detail this could be the catalyst for creating a new combination of configuration packages. Further, if there is a need for a new configuration template this could in turn trigger the first section and a need for a new characteristic, with values and configuration packages.

5.4 Operationalization of Workflow Details

A workflow detail in RUP is activities that are performed together, but they are not performed in a sequence pattern or all at once. However, often a cyclical fashion is found. (Kruchten, 1999) A discipline, as MMC represents in our case, is divided into workflow details, which are presented in workflow detail diagrams. Each diagram shows possible activities to perform, which artifacts that are considered to be prerequisites and results. Figure 29 contains a brief example of this notation. Two types of worker are used: the light gray worker is responsible for this specific workflow detail and the dark gray worker is a participant. Activities are presented as a white block-arrow. Finally, we find artifacts as document symbols, where the arrows illustrate if they are prerequisites, results or both.

---

**Figure 29. Example of Workflow Detail Diagram**
The RUP Shell

Through this notation, it is possible to describe who is doing what, how and when with the method (Kruchten, 1999). However, this notation is missing the important question of why a specific activity is performed. Therefore, each activity has to be described further in an activity description. An activity description contains the purpose, the artifacts used and produced/modified during the activity and a listing of steps. These steps are the operationalization of the purpose. This presentation is standardized into a table enabling a quick overview. In addition, the steps are elaborated more in detail in separate paragraphs.

5.5 RUP’s Rationality

The systems engineering method’s rationality is central to MMC, since we use it as the focal area for method configuration. Therefore, we need to consider how the rationale can be obtained in the base method we intend to use. Depending on the chosen base method obtaining the method’s rationality could involve different amounts of reconstruction. However, RUP reduces these considerations, since each prescribed action has an explicit purpose (with the exception of steps). Consequently, each purpose can be reformulated into a goal statement, which can be used for matching the goals needed for fulfilling a characteristic of a development situation.

As an example, we could use the activity ‘Capture a Common Business Vocabulary’ found in RUP. Its activity description is found in Table 5 below and includes a purpose for what to do and why, two steps, three input artifacts and one resulting artifact. If we consider a characteristic such as the project team’s knowledge of business objectives this activity could be classified differently depending on the value (configuration package) we are working with. Consider one option where the business is highly specialized and the project team has only brief knowledge about the business objectives. Hence, it becomes important to define a common language in order to facilitate communication within the project team as well as with the business as such. One possible solution could be classifying this activity, its steps and resulting artifact as ‘Emphasized as is’.

Table 5. Activity: Capture a Common Business Vocabulary (RUP 2001)

<table>
<thead>
<tr>
<th>Purpose:</th>
</tr>
</thead>
<tbody>
<tr>
<td>To define a common vocabulary that can be used in all textual descriptions of the business, especially in descriptions of business use cases.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Common Terms</td>
</tr>
<tr>
<td>Evaluate Your Results</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Vision</td>
<td>Business Glossary</td>
</tr>
<tr>
<td>Vision</td>
<td></td>
</tr>
</tbody>
</table>
5.6 Summary

In this short chapter, we have discussed concepts in RUP that are central when turning the MMC into an RUP specific version. We have discussed the importance of choosing a layer of granularity before starting the analysis, and in the RUP case we have recommended the activity layer. The chosen layer is equivalent to the diagram layer in Brinkkemper (1996). The MMC framework presented at the end of the previous chapter has been turned into six workflow details facilitating three different paths through the meta-method. Finally, we have operationalized our thoughts about the systems engineering method’s rationality as the purpose of the prescribed actions in RUP.
The objective of this chapter is to present each of the three prescribed sections in MMC. They are operationalized into the RUP shell presented in the previous chapter, hence turned into a meta-method applicable with RUP as the base method. An operationalization means turning the meta-method framework into a more detailed method description containing prescribed actions on a finer layer of granularity as well as presenting prerequisites and deliverables.

In Chapter 4, we presented the MMC concepts, the analytical tool and the classification system. They constitute the core of MMC and we need to propose administrative procedures for these thoughts. These procedures are based on practical considerations, mostly earned from the selection of experiences presented in the forthcoming Chapter 7. However, in order give a straightforward presentation of the meta-method, and subsequently making it useful as a method description, we will leave discussions of design issues to the next chapter.

This chapter is structured into three main sections constituting the sections found in the MMC framework. MMC as such is treated as a discipline, which is decomposed into workflow details (see Figure 28 in previous chapter), activities and steps. Each activity is presented under separate headings and contains a list of steps, and prerequisites and deliverables. Furthermore, the MMC rationality is made explicit on the workflow detail and activity level of granularity, consequently complying with the guidelines in RUP. In order to support reusability, we have stated the frequency of each activity.

Notice that in this chapter we will use the concepts method as well as base method when we refer to systems engineering methods used as the starting point for configuration. The Method for Method Configuration will be referred to as MMC. Furthermore, we will use the concept process engineer instead of method engineer in this chapter.
6.1 MMC Section 1

The workflow details, activities and steps in this section are an operationalization of ‘Manage Change Request for Configuration Package’ and ‘Define Configuration Package’ in the RUP-specific MMC-framework (see Figure 28 in Chapter 5).

6.1.1 Manage Change Request for Configuration Package

Figure 30. Workflow Detail: Manage Change Request for Configuration Package

The purpose of this workflow detail is to:

- Evaluate the impact the change request should have on the list of characteristics.

6.1.1.1 Activity: Identify Potential Changes

<table>
<thead>
<tr>
<th>Purpose:</th>
</tr>
</thead>
<tbody>
<tr>
<td>To identify potential changes or elicit and their magnitude in order to consider them as tentative changes.</td>
</tr>
</tbody>
</table>
Meta-Method for Method Configuration

Steps:
- Consider Type of Change
- Consider Experiences and Future Projects

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Request</td>
<td>Tentative Changes</td>
</tr>
</tbody>
</table>

Frequency: Once per change request which aims at a new or changed characteristic or at a new or updated configuration package.

Worker: Process Engineer

Concepts:
- Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration [Yes, No]
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.

Workflow Details: Manage Change Request for Configuration Package

CONSIDER TYPE OF CHANGE
A change request could be expressed in many different ways. One way could be suggestions for new or refined characteristics, however usually they are expressed in more general terms. Furthermore, the change request could involve changes of different magnitude if they are implemented. Therefore, it is necessary take a position on what type of change is requested in order to facilitate the use of the MMC activities and steps together with the specific need. The distinction should be made between:

- Elaboration of a new or changed characteristic
- Changes in one or more configuration packages

Changes of the former type will add to or change the characteristic list, as well as involve elaboration or changes of one or several configuration packages. The latter case imposes changes in one or more configuration packages leaving the characteristic list unchanged. Subsequently, these two types of changes, which the latter is a subset of the former, have different impact on the MMC artifacts. Furthermore, in the latter case we need to re-formulate the change request into a characteristic or a refined version of an existing characteristic.
CONSIDER EXPERIENCES AND FUTURE PROJECTS
The change request could address an aspect in a development situation that, potentially, has not had the right focus with regard to the goals that are found valuable. However, it is important to consider whether or not the change request actually concerns configuration problems possible to abstract into a development situation or if it is a project specific occurrence. Consequently, it is necessary to investigate the frequency with which this configuration occurs. Occurrence should be interpreted in two ways:

- Consider the frequency with which this configuration issue has occurred during earlier projects.
- Consider the frequency with which this configuration will occur in future projects. This might be a reason for the reconfiguration of existing characteristics, configuration packages, or the creation of new sets of these tools.

If the occurrence is or will be frequent, reformulate or use the change request as the tentative change. It might be necessary to divide the change request into several tentative changes. Since we can have changes of different magnitudes, the list of changes could consist of tentative characteristics, values and configuration packages. If the specific situation is not and will not be frequent in the organization it might be better to fine-tune an existing configuration template.

6.1.1.2 Activity: Evaluate Potential Changes

| Purpose: | To evaluate, label and describe a set of unique changes in order to enable the creation or update of the list and the content of available configuration packages |
| Steps: | Crosscheck Changes | Create Unique Identifications | Describe Each Characteristic |
| Input Artifacts: | Tentative Changes |
| Resulting Artifacts: | Tentative Changes (updated) |
| Frequency: | For each tentative change |
| Worker: | Process Engineer |
| Concepts: | Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration [Yes, No] | Configuration Package - a method configuration of the base method suitable for a characteristic’s value. |
Meta-Method for Method Configuration

- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.

**Workflow Details:** Manage Change Request for Configuration Package

**CROSSCHECK CHANGES**

The list of tentative changes has to be evaluated internally. The reasons for this are twofold, the list can contain duplicates of changes and changes can have dependencies. One reason for scanning for duplicates is that the change request as such could be used as descriptions of tentative changes, and their formulations could hide similar changes. Dependencies between changes can occur when changes are specified with a finer level of granularity. In such a case, a good strategy would be to group changes that belong together. Thus, it would be possible to present changes that must be performed as one unit. An evaluation of changes should focus on:

- Which of the changes are unique?
- Are the tentative changes expressed in different ways but with the same meaning?
- Which goals during the development process do the changes address?

This means performing a crosscheck within the list. If you find duplicates, group them together.

**CREATE UNIQUE IDENTIFICATIONS**

The characteristic list used for selection of configuration packages should contain unique characteristics and values. If the change involves elaboration of a new characteristic, this characteristic needs identification. This identification should be short (two or three words) and informative. Until now, you have worked with a tentative name, several tentative names for the same characteristic. They could act as the foundation for naming a characteristic:

- Starting with the brief description focusing on words that are essential for the described situation, an important aspect to consider is the purpose of the characteristic;
- Select the most significant name from the list of tentative names;
- When you have one tentative name consider if it covers your intention with the characteristic. The name should reflect the characteristic of one or several development situations.

In cases in which the change is about updating the content of a characteristic no new identification is necessary. However, in such a case each change has to be related to one unique and existing characteristic.
Characteristics may very well be expressed as questions. Used this way, they become unique questions that can be asked when building or retrieving configuration templates later on. The characteristics’ values then act as possible answers.

**DESCRIBE EACH CHARACTERISTIC**

Describing each unique characteristic is important. The description should contain the essential goals of the development situation that this characteristic focuses on. Such a description has two purposes:

- You have to articulate wherein the characteristic is to be considered as unique.
- The description is input for later usage of the characteristic. Thus, it supports the interpretation of the characteristic when the development situation is used during later method configuration tasks.

### 6.1.1.3 Activity: Identify Potential Values

<table>
<thead>
<tr>
<th>Purpose:</th>
</tr>
</thead>
<tbody>
<tr>
<td>To identify a set of potential values for a characteristic to facilitate future choices of configuration packages and configuration templates.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify Tentative Values</td>
</tr>
<tr>
<td>Describe Each Value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tentative Changes</td>
<td>Tentative Changes (updated)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequency:</th>
<th>For each definition or redefinition of a characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worker:</td>
<td>Process Engineer</td>
</tr>
</tbody>
</table>

**Concepts:**

- Base method - is the systems engineering method chosen as the starting point for the configuration work.
- Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration {Yes, No}
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.
- Dimension – a set of possible values for a characteristic. Example: {Yes, No}
• Value – a possible choice in a characteristic. A value corresponds to one configuration package.

**Workflow Details:** Manage Change Request for Configuration Package

**IDENTIFY POTENTIAL VALUES**

Each characteristic in the characteristic list has a set of values constituting a dimension. Through the selection of a value, the dimension is used when characterizing a development situation. Thus, each value has a different impact on the situational systems engineering method since it corresponds to a specific configuration package.

In order to find the values for a specific characteristic, focus on how the characteristic can impact or should have an impact on the final situational systems engineering method, that is to what extent and how the goals in question are to be achieved. The dimension should have the same meaning as the characteristic’s description, thus you have to cover the extreme cases. Furthermore, consider nuances between the extreme cases, nuances that are relevant to your business. Together these constitute possible values. Input about possible values can be found in the change request, if it is exhaustive. Otherwise, important sources are Project Managers and Project Members from projects having this characteristic.

**Example:**

One potential characteristic for a development situation could be the following: Consider a development situation in which the resulting information system is to be integrated with existing systems. This is not an unusual situation since many organizations have legacy systems to maintain. A successful integration is based on knowledge about the existing systems; hence there might be a need for emphasizing reverse engineering issues. In this example, we can identify a clear distinction between two different development situations. Either there is a need for information system integration or there is not, which gives us two potential values for the characteristic defined as below, apart from the dummy value N/A.

```
<characteristic name = "Legacy Integration?">
  <value valueid = "Yes" cpid = "Perform Legacy Integration">
  </value>
  <value valueid = "No" cpid = "No Legacy Integration">
  </value>
  <value valueid = "N/A" cpid = "N/A"></value>
</characteristic>
```

As the list of characteristics grows some characteristics might not be relevant for every development situation. In order to reduce the workload during configuration, it is necessary that characteristics can be ignored if they are not considered relevant. Such a standpoint should be explicit and each characteristic needs a dummy value. In each dimension, it should be possible to select N/A, i.e.,
A Detailed Description of Method
for Method Configuration

this characteristic is not relevant to a development situation or a configuration template.

DESCRIBE EACH VALUE
In order to make the characteristic and its dimension useful tools during method configuration, the significance of each value has to be documented. This description will contribute to a richer description of the characteristic as such. Further, the description is a support when you are to match a project situation with a characteristic’s value. Then, it is easier to interpret how the values are to be used. The description should focus on the implication of characterizing a project with this value, to what extent and how the goals, focused by the characteristic, are being fulfilled. If the characteristic is documented as questions, it will be natural to formulate the values as answers.

6.1.1.4 Activity: Evaluate Values

<table>
<thead>
<tr>
<th>Purpose</th>
<th>To evaluate tentative values for a characteristic in order to reach relevant and coherent configuration packages.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steps</td>
<td>Evaluate Impact and Relevance</td>
</tr>
<tr>
<td></td>
<td>Evaluate Uniqueness</td>
</tr>
<tr>
<td></td>
<td>Evaluate Fragmentary</td>
</tr>
<tr>
<td>Input Artifacts:</td>
<td>Tentative Changes</td>
</tr>
<tr>
<td>Frequency:</td>
<td>For each set of potential values (new or redefined)</td>
</tr>
<tr>
<td>Worker:</td>
<td>Process Engineer</td>
</tr>
<tr>
<td>Concepts:</td>
<td>Base method - is the systems engineering method chosen as the starting point for the configuration work.</td>
</tr>
<tr>
<td></td>
<td>Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration (Yes, No)</td>
</tr>
<tr>
<td></td>
<td>Configuration Package - a method configuration of the base method suitable for a characteristic’s value.</td>
</tr>
<tr>
<td></td>
<td>Configuration Template - a combined method configuration for a set of recurrent project characteristics.</td>
</tr>
<tr>
<td></td>
<td>Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.</td>
</tr>
<tr>
<td></td>
<td>Dimension – a set of possible values for a characteristic. Example: (Yes, No)</td>
</tr>
</tbody>
</table>
EVALUATE IMPACT AND RELEVANCE
In order to reach a relevant set of configuration packages, it is important to use values that are significant to your business. Not every conceivable value is of immediate interest.

Since configuration packages are a reflection of values, they have a direct impact on method configuration as reusable assets. Significance can be divided into impact and relevance. First, consider, on an overarching level, which impact a tentative value has on the base method. What sections of the method will be affected if a project is characterized this way? If you cannot find any, this value and its tentative configuration package are of minor importance for the base method. Refrain from adding such values to the characteristic’s dimension.

Second, a value and its configuration package have to be an interesting nuance in at least one development situation. A value might have significant impact on method configuration as such, but if you cannot find development situations in your business using this value it is of no use. This holds true irrespective of the impact a value has on the base method. Do not add such values to the characteristic’s dimension.

However, be aware of the fact that the base method might not be able to address goals focused through a characteristic. A method is a finite set of prescribed actions and hence finite types of goals are covered. In such a case, the tentative value should be added to the dimension since the base method probably needs enhancement in order to fulfill these goals.

EVALUATE UNIQUENESS
You have to balance the possibilities to characterize a development situation with nuances and magnitudes in differences, that is the value’s uniqueness. Therefore, crosscheck the values. Is the impact of a value, to a large extent, covered by another value in this dimension? In such cases consider if it is a reduction in quality if you drop one of the values, and its potential configuration package.

EVALUATE FRAGMENTARY
Each value has been evaluated with regard to impact, relevance and uniqueness. Nevertheless, it is important to evaluate whether or not the value’s impact is scattered over several sections of the base method. A configuration package originating from such a value will probably affect prescribed actions and artifacts throughout the base method. A combination based on this type of configuration packages often results in overlapping configuration packages. Consequently, overlapping configuration packages could necessitate decisions when the
overlapping activities have conflicting classifications. Such a situation could be a sign of poor performing values and/or characteristics.

### 6.1.1.5 Activity: Check with Existing Characteristics

<table>
<thead>
<tr>
<th>Purpose</th>
<th>To update the characteristic list in order to incorporate the new characteristic.</th>
</tr>
</thead>
</table>
| Steps   | • Compare with Characteristic List  
         | • Add Characteristic |
| Input Artifacts: |  
         | • Tentative Changes  
         | • Characteristic List |
| Resulting Artifacts: |  
         | • Characteristic List (updated) |
| Frequency: | Each time the change involve a potentially new or redefined characteristic |
| Worker: | Process Engineer |
| Concepts: |  
         | • Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration {Yes, No}  
         | • Configuration Template - a combined method configuration for a set of recurrent project characteristics.  
         | • Development Situation – an abstraction of one or more existing or future software development projects with common characteristics. |

**COMPARE WITH CHARACTERISTIC LIST**

The characteristic list should only consist of unique characteristics. Unique has to be understood in a somewhat broader sense, since each characteristic is a qualitative description of a development situation. The match between characteristics and actual situations can never be perfect. Thus, a characteristic should not address too narrow aspects of a development situation. Otherwise, you run the risk of getting too many ways to characterize a development situation. In the long run, you have to make a decision on characteristics that are relevant during the method configuration for a specific project, when building and retrieving configuration templates.

Therefore, you have to check the new or redefined tentative characteristic(s) with those existing in the characteristic list. Use the description of each characteristic to compare them. Furthermore, each characteristic has a set of values, which is another way of describing the characteristic. These values and their descriptions can also be used in the comparison of characteristics. If you find a characteristic similar to a proposed one, consider in which way an additional characteristic will be a quality improvement to your method configurations. Maybe
the initial change request should be reformulated into a redefinition of one or more characteristics instead of creating a new one. In the case of a refined characteristic, exclude the older version from the comparison.

**ADD A CHARACTERISTIC**
Add the new characteristic by adding the characteristic’s name and its description to the characteristic list. It is important to add the description to the characteristic list. The description is valuable when the characteristic is interpreted during configuration work, e.g., when selecting configuration templates. If the characteristic replaces an older version drop the latter from the characteristic list.

### 6.1.2 Define Configuration Package

![Workflow Diagram](image)

**Figure 31.** Workflow Detail: Define Configuration Package

The purpose of this workflow detail is to:

- Define a configuration package for a specific characteristic’s value which could be reused during configuration work.
6.1.2.1 Activity: Identify Relevant Prescribed Actions

<table>
<thead>
<tr>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>To identify relevant prescribed actions for classification.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify Relevant Prescribed Actions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristic List</td>
<td>Configuration Package (tentative)</td>
</tr>
<tr>
<td>RUP</td>
<td></td>
</tr>
</tbody>
</table>

**Frequency:** For each new or updated Configuration Package.

**Worker:** Process Engineer

**Concepts:**
- Base method - is the systems engineering method chosen as the starting point for the configuration work.
- Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration {Yes, No}
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.

**Workflow Details:** Define Configuration Package

**IDENTIFY RELEVANT PRESCRIBED ACTIONS**

The starting-point is the base method. Each recommended prescribed action and belonging artifact in the base method is not relevant to perform or produce with respect to the definition of a configuration package. However, when working with configuration packages certain sections of the base method are not relevant to take a detailed standpoint on. Subsequently, the configuration package is coarsely demarcated through identification of the characteristic’s values.

List the prescribed actions and the belonging artifacts that potentially could be affected by this project characteristic. It is recommended to start the analysis on the activity level of granularity with respect to balance between precision and cost. We can move upwards in the layer structure, if possible, or downwards, if necessary. The selection should be based on the prescribed actions’ purpose and the goals addressed by the characteristic. If they address the same aspects of the development situation then the prescribed actions are within the scope of the characteristics. Focus on rather large building blocks in order to avoid fragmentary configuration packages. All sections of the base method that are placed outside this demarcation will automatically be classified, as N/A. N/A is a dummy value and these prescribed actions and their belonging artifacts are not relevant to classify from the characteristic’s point of view. Hence, they should not affect further configurations in which this configuration package is used.
**6.1.2.2 Activity: Consider Importance and Complexity**

<table>
<thead>
<tr>
<th><strong>Purpose</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>To classify the prescribed actions and their belonging artifacts in order to produce or update a configuration package.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Steps</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze Prescribed Actions and Artifacts</td>
</tr>
<tr>
<td>Classify Prescribed Actions and Artifacts</td>
</tr>
<tr>
<td>Extend with Parts from Additional Methods</td>
</tr>
<tr>
<td>Document Configuration Package</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Input Artifacts:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Package (tentative)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Resulting Artifacts:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Package (refined)</td>
</tr>
</tbody>
</table>

**Input Artifacts:** Configuration Package (tentative)

**Resulting Artifacts:** Configuration Package (refined)

**Frequency:** For each new or updated Configuration Package.

**Worker:** Process Engineer

**Concepts:**
- Base method - is the systems engineering method chosen as the starting point for the configuration work.
- Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration {Yes, No}
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.
- Dimension – a set of possible values for a characteristic. Example: {Yes, No}
- Value – a possible choice in a characteristic. A value corresponds to one configuration package.

**Workflow Details:** Define Configuration Package

**ANALYZE PRESCRIBED ACTIONS AND ARTIFACTS**
The purpose of configuration packages is primarily to create demarcated and reusable configured sections of the base method. These sections are the coarse list of prescribed actions that has been elicited. The next step is to take a standpoint on whether or not they are relevant to perform activities and produce artifacts, respectively, with regard to the goal(s) in focus. However, it is important to be aware of the fact that prescribed actions in the base method and their artifacts are not independent of each other, since artifacts are deliverables from procedures. Consequently, we need to classify each prescribed action on the chosen level of granularity and its belonging artifacts in the temporary list in order to create a
tentative Configuration Package. For a classification schema, see the step Classify Activities and Artifacts, which can be used for both activities and artifacts.

Analyses of activities and their artifacts should be performed in a structured manner. The decision to analyze should be based on the prescribed action’s area of concern. The central issue is the prescribed action’s purpose and how the prescribed action’s deliverables support the goals made focal by the characteristic’s value. In order to address this question properly use Figure 32, below, as a tool. In the horizontal direction, we grade the importance of the prescribed action or the artifact in the development situation compared with the goals in the base method. On the Y-axis, we grade the complexity of the area in respect to this development situation. In the intersection, we find the current situation with normal importance and complexity.

The tool in Figure 32 has one restriction. Each systems engineering method has atomic parts, which are not relevant to decompose. In RUP, the lowest layer of granularity is step. The shaded areas in Figure 32 illustrate movements that are not possible to perform on a step. Subsequently, we have two restrictions. First, it is not possible to reduce a step since it is an atomic prescribed action. Second, a step cannot be extended since an extension is implemented by adding prescribed actions from a lower level of granularity than the prescribed action in question and there is no level lower than step.

![Figure 32. Analysis of Prescribed Actions](image-url)
Example:
A development situation exists in which the final information system is not a database-driven solution. The information system takes the users’ input and transforms it into a response, but the goal is not to memorize these responses. This situation has to be characterized, and a configuration package is defined.

In the ‘Analysis and Design’ discipline and the ‘Design the Database’ workflow detail, we identified the ‘Database Design’ activity. Its purpose is described in the RUP documentation (2001) as follows:

- ‘To ensure that persistent data is stored consistently and efficiently.’
- ‘To define behavior that must be implemented in the database.’

When characterizing our development situation, which does not aim to implement a database, this activity is of low importance. The complexity of the activity is probably not possible to discuss, since there is not need for a database. Overall, this gives a horizontal movement to the right in Figure 32 and a ‘Skip’ should be considered. When switching perspectives to the product view, this activity has two deliverables, the Data Model and the Changes Request, where the former is the primary deliverable. Since we have classified the activity as a ‘Skip’ it is natural to delete (‘Skip’) the artifact from the list of artifacts to produce. Furthermore, changes cannot arise from a non-performed activity; subsequently we do not need the change request.

CLASSIFY PRESCRIBED ACTIONS AND ARTIFACTS
The classification of activities and their artifacts is based on the two dimensions in Table 6, below. In the vertical dimension, we classify how much attention should be devoted to a prescribed action or an artifact. We use the heuristics in Figure 32 as input. The attention dimension is connected to the importance of fulfilling the goal of a prescribed action. As default, the goal should be fulfilled and hence the prescribed action and the artifacts have to be performed and produced, respectively. If we have considered a prescribed action to be of low importance for this type of project characteristic, then the action can be classified as ‘Skip’. The opposite case with a highly important goal means that the prescribed action should receive more attention than usual and the action is emphasized. From the artifact view, the line of reasoning is much the same, however the question is about how much attention a specific artifact should be given during the project. An artifact of low importance should have rather minimal attention, and could be classified as ‘Skip’. Highly important artifacts should be in focus during the project and consequently their usage should be emphasized.

This classification is combined with the horizontal dimension, as is the heuristics in Figure 32. Subsequently, we have a relationship between the complexity dimension in Figure 32 and the enhancement dimension in Table 6. The central question is how extensive the performance of a prescribed action and/or the usage of the artifacts should be. The question remains whether or not all steps in the
activity should be performed, or if any steps should be added. The default is the performance of all prescribed actions on a lower layer of granularity than the prescribed action in question – the ‘as is’ alternative in Table 6 below. For example, in a situation characterized with low complexity a ‘Perform reduced’ is introduced on the activity level, when some steps in activity are not found useful and the amount of steps is reduced through ‘Skip’. The opposite situation would be an extension of an activity with steps that are not part of the base method.

Together, the attention and the enhancement dimensions result in the matrix illustrated in Table 6. These basic principles can be used for artifacts as well, for example, where sections in an artifact are classified as ‘Skip’ and the artifact as such is classified as ‘Produced reduced’. However, this classification should be guided by the classification of the related prescribed actions.

Table 6. Performance of Prescribed Actions

<table>
<thead>
<tr>
<th>Attention given to prescribed action</th>
<th>Enhancement</th>
<th>Reduced</th>
<th>As is</th>
<th>Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Skip</td>
<td>Skip</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td>Perform reduced</td>
<td>Perform as is</td>
<td>Perform extended</td>
<td></td>
</tr>
<tr>
<td>A lot</td>
<td>Emphasize reduced</td>
<td>Emphasize as is</td>
<td>Emphasize extended</td>
<td></td>
</tr>
</tbody>
</table>

Thus, the application of the process perspective means that each artifact in the resulting set of artifacts is analyzed. Reducing the scope of a prescribed action means:

a) that the remaining prescribed actions are sufficient to deliver the artifact;
b) that certain sections of the artifact cannot be delivered or refined with the remaining prescribed actions. Consequently, the artifact is reduced in scope;
c) that an artifact cannot be delivered or refined with the remaining prescribed actions.

Extending the scope of a prescribed action on the other hand means:

d) a new artifact is delivered or refined through this extension;
e) a new section of an artifact is delivered or refined through this extension.

However, since we have a restriction on possible movements in Figure 32, these restrictions also guide possible classifications in Table 6, with regard to the layer of granularity. It is not possible to use the shaded areas when working with the granularity level of step.
EXTEND WITH PARTS FROM ADDITIONAL METHODS

The right-hand column in Table 6 indicates that method configuration is not always enough in order to adapt the base method to the development situation at hand. The significance of an extension is that the base method’s ability is not enough to fulfill the goals addressed by the characteristic’s value. Subsequently, the base method’s rationality is not enough and in such situation it would be necessary to add prescribed actions and/or artifacts from additional methods.

Different implications arise depending on how the complementing parts are added to the base method. Adding prescribed actions and/or artifacts, which have a focus similar to existing parts in the base method could mean either of two things:

- An exchange of prescribed actions and/or artifacts
- A mix of existing and new prescribed actions and/or artifacts

In the former case, we have implicitly classified the exchanged part as ‘Skip’. In the later case, we have to classify the original parts as anything other than ‘Skip’ and the additional parts as ‘New’. Overlapping prescribed actions or artifacts have to be paid extra attention during configuration since there should not be duplicates within the configuration package (or later configuration templates for that matter).

Three basic considerations have to be made before adding complementing parts to the configuration package. First, consider how important it is to address this issue during projects using this configuration. Second, the purpose of these parts has to be investigated. Do they fulfill the desired purpose of the configuration package? There is no rationale for adding them to the configuration package if they do not. More concretely, consider whether or not these prescribed actions and/or artifacts facilitate capturing the complexity of the development situation or improving the way it is done.

Third, a tentative method part has to fit the base method on a conceptual basis if it is to be added. The method part must use concepts that are consistent with those in the base method. Before adding parts from additional methods there is a need to investigate them with regard to their perspective, values, and concepts. If they do not correspond do not add such parts to the method, unless they can be mapped in some way. The possibility of mapping depends on existing differences.

DOCUMENT THE CONFIGURATION PACKAGE

Each prescribed action and artifact should be documented as they are classified. Together with the classification itself there should be a documented reason for the classification. Argue for classification with regard to the importance of goal-fulfillment and the complexity in reaching the focused goal, respectively. This description is used in later MMC-activities such as combination of configuration packages, and fine-tuning configuration templates into a situational systems engineering method. Then, it is possible to recall why the prescribed action and/or artifact have a given classification.
A Detailed Description of Method
for Method Configuration

Furthermore, it is important to document the specific parts of the base method that are emphasized, reduced or skipped. If one part of the base method is classified as ‘Performed reduced’ on a chosen level of granularity it is important that lower layers contain which parts that are classified as ‘Skip’. Otherwise, the configuration package will not be consistent.

The purpose of configuration packages is to facilitate reusable configuration through combing configuration packages, thus they need interfaces that correspond with each other. Each procedural aspect of method takes artifacts as input in order to modify or create additional artifacts as output. Hence, you have a flow of artifacts through the method, and the prerequisites for the succeeding prescribed actions have to be met. Therefore, the interface of each configuration package has to be specified, both its need for input artifacts as well as its resulting artifacts.

One aspect of documentation would be implementation of the configuration package using Rational Process Workbench (RPW). Classified prescribed actions and artifacts then act as modeling actions in RPW. For example, an activity classified as ‘Skip’ can be suppressed using RPW.

### 6.1.2.3 Activity: Configuration Package Validation

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>Validate a configuration package in order to achieve consistency.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steps:</td>
<td>Validate Internal Consistency, Validate Output</td>
</tr>
<tr>
<td>Input Artifacts:</td>
<td>Tentative Configuration Package</td>
</tr>
<tr>
<td>Resulting Artifacts:</td>
<td>Configuration Package</td>
</tr>
<tr>
<td>Frequency:</td>
<td>For each new or updated configuration package</td>
</tr>
<tr>
<td>Worker:</td>
<td>Process Engineer</td>
</tr>
<tr>
<td>Concepts:</td>
<td>Configuration Package - a method configuration of the base method suitable for a characteristic’s value.</td>
</tr>
<tr>
<td>Workflow Details:</td>
<td>Define Configuration Package</td>
</tr>
</tbody>
</table>

### VALIDATE INTERNAL CONSISTENCY

Each configuration package should consist of at least one prescribed action and one resulting artifact. Furthermore, each prescribed action has a purpose that contributes to the purpose of the configuration package on an aggregate level. In order to fulfill its purpose each prescribed action needs certain inputs as pre-requisites. We have to validate that sequential prescribed actions in the configuration package have the pre-requisites they need. Focus on tracking artifacts produced or modified by prescribed
actions in the configuration package. The activity layer of granularity would be preferable to use for this task since artifacts are made explicit on that layer. If an activity in the late part of the configuration package needs an artifact’s origin in an earlier activity, this activity cannot be classified as ‘Skip’. As a complement, investigate whether the configuration package contains artifacts that are used neither as input nor output from activities within the configuration package. Such artifacts should be classified as ‘Skip’, if they are not used as intermediate work products during the prescribed action, since they are not created or modified within this specific configuration package.

VALIDATE OUTPUT
To fulfill a purpose, each configuration package has to contribute to the total process. Thus, each configuration package has to have one or more deliverable results – artifacts (new or modified). Therefore, check the configuration package to validate that it has at least one resulting artifact. In those cases when it does not have any resulting artifacts consider the overall purpose of this configuration package. Does it have any natural artifacts? If it has, then check activities classified as skipped. It is possible that relevant artifacts have been lost through such a classification.

6.2 Section 2
The workflow details, activities and steps in this section are an operationalization of ‘Manage Change Request for Configuration Template’ and ‘Define Configuration Template’ in the RUP-specific MMC-framework (see Figure 28 in Chapter 5).

6.2.1 Manage Change Request for Configuration Template

Figure 33. Workflow Detail: Manage Change Request for Configuration Template
A Detailed Description of Method for Method Configuration

The purpose of this workflow detail is to:

- Evaluate what impact the change request should have on the list of configuration templates.

### 6.2.1.1 Activity: Identify Potential Configuration Template

**Purpose:**
- To characterize a development situation based on one or several change requests, in order to select configuration packages for configuration template.

**Steps:**
- Classify Change Request
- Characterize Development Situation
- Document Characterization

**Input Artifacts:**
- Change Request for Configuration Template

**Resulting Artifacts:**
- Tentative Configuration Template

**Frequency:** For each change request about Configuration Templates

**Worker:** Process Engineer

**Concepts:**
- Base method - is the systems engineering method chosen as the starting point for the configuration work.
- Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration {Yes, No}
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.
- Dimension – a set of possible values for a characteristic. Example: {Yes, No}
- Value – a possible choice in a characteristic. A value corresponds to one configuration package.

**Workflow Details:** Manage Change Request for Configuration Template

**CLASSIFY CHANGE REQUEST**

Re-occurring development situations should be supported, to begin with, by equivalent situational systems engineering methods. Each configuration template is to aim at one such re-occurring development situation. A change request concerning a configuration template could originate in different reasons:
Experiences about lacking support for a development situation;
A shift in development situations, i.e., new needs.

Different reasons have different implications for what is to be considered as proper measure. A change request could result in a decision about a new or updated configuration template or a decision that no measure should be taken. In order to take a proper measure the change request has to be classified as a request for a new or an update configuration template (it can be reclassified to new if we discover that the old configuration template can coexist).

Furthermore, there are two types of possible updates. In the first category, we find situations in which a new value for a specific characteristic is chosen and subsequently a configuration package is exchanged. This type of change should be treated in the same way as the creation of a new configuration template, since we have a new characterization of the development situation. As a second category, we have change requests regarding the combined content of the template’s configuration packages, where the chosen configuration packages are left unchanged. In these situations, focus should be on the combining activity in MMC (see Section 6.2.2.1), since the characterization of the development situation is left unchanged.

The second situation could occur for two reasons. First, a direct change request to consider a recombination of the configuration template’s content. Second, a change request for redefining a configuration package, which is part of a configuration template. Consequently, the latter situation is triggered indirectly by another change.

CHARACTERIZE DEVELOPMENT SITUATION
Irrespective of whether or not we are creating a new configuration template or updating an existing one, this activity involves characterizing the development situation’s uniqueness. The configuration template is characterized using the characteristic list as input. Thus, we take each characteristic and select the most significant value from the characteristic’s dimension. The characteristic list can be a battery of questions utilized when designing the development situation, assuming that useful characteristics exist (otherwise section 1 should be triggered through a change request). If a characteristic is not relevant for illustrating the development situation, then we select the null value (N/A). A tentative profile of the development situation is elicited when we iterate the characteristic list, taking a standpoint on tentative configuration packages. The selection of configuration packages (or values) is the base for a new configuration template.

If we have a change request for an update, we select the affected configuration template. We then identify the method section affected. This is done through the characteristic list. We iterate the characteristics in the list and decide on proper characteristic values that correspond to the development situation. We then exchange these values and their related configuration packages.
A Detailed Description of Method
for Method Configuration

DOCUMENT CHARACTERIZATION
Each selected characteristic’s value has to be documented. Each selection is a selection of a configuration package, since each value has a corresponding configuration package. The complete characterization of the development situation is thus a tentative configuration template. Documentation for updating a configuration template focuses on the changed characteristic and its selected values.

6.2.1.2 Activity: Evaluate Potential Configuration Template

<table>
<thead>
<tr>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>To evaluate if the configuration template is common throughout the organization and if it exists in the list of configuration templates.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate Earlier Projects</td>
</tr>
<tr>
<td>Evaluate Future Projects</td>
</tr>
<tr>
<td>Check with Existing Configuration Templates</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Template (tentative)</td>
<td>Configuration Template (intermediate)</td>
</tr>
<tr>
<td>Characteristic List</td>
<td></td>
</tr>
</tbody>
</table>

| Frequency: | For each tentative configuration template |
| Worker: | Process Engineer |

Concepts:
- Base method - is the systems engineering method chosen as the starting point for the configuration work.
- Characteristic - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration {Yes, No}
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.
- Dimension – a set of possible values for a characteristic. Example: {Yes, No}
- Value – a possible choice in a characteristic. A value corresponds to one configuration package.

Workflow Details: Manage Change Request for Configuration Template

EVALUATE EARLIER PROJECTS
The tentative configuration template is an expression for the need for a new or updated pre-configuration of the base method. However, since configuration templates should constitute pre-configurations of common development situations this aspect has to be taken into consideration. Does the new or updated tentative
configuration template reflect a commonly occurring development situation in your business?

Focus on earlier projects. Information from Project Managers with project experience is valuable. Focus on projects that have similarities. If necessary, characterize each project according to the characteristic list. Do these projects constitute a significant part of the business? Do they frequently occur? Does the change request contain such an assessment?

EVALUATE FUTURE PROJECTS
The aim of a new or updated configuration template might not be to conform to earlier projects. Future projects could reveal new or changing needs. Therefore consider if the development situation and thus the new or the updated configuration template will be commonly used in the future?

Base your standpoint on the order stock and system engineering project plans. Since they have not occurred yet they tend to be abstract. Therefore characterize them as future projects according to the characteristic list. The profile we receive can be matched with the profile of the tentative configuration template. Consider the volume/frequency of these projects and relate it to the total volume of your business. However, frequency should not be the only metric grounding the decision to elaborate a new configuration template. For example, single projects could arise with specific configuration needs; in such situations consider the extra effort needed to create a configuration template.

CHECK WITH EXISTING CONFIGURATION TEMPLATES
Each configuration template should be unique and it is considered unique if it does not exist in the list of configuration templates. Thus, the potential configuration template is compared to each configuration template in this list. Since each configuration template has a profile, the selected characteristic’s values, these can be compared. As result, we obtain a list of divergences between different configuration templates and the tentative one. If we receive an almost matching configuration template, we consider what we gain in quality by adding the tentative configuration template to the list. Adding several similar configuration templates makes the choice between them harder when it comes to the selection of a configuration template for fine-tuning. Then, we risk getting a selection consisting of several configuration templates that almost fit, but in different aspects. In cases in which the combined content of the template’s configuration packages is updated (hence, involving the same configuration packages) this consideration is not relevant.
6.2.2 Define Configuration Template

The purpose of this workflow detail is to:

- Define a configuration template to facilitate selection of reusable method assets.

6.2.2.1 Activity: Combine Configuration Packages

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>To combine selected configuration packages into a configuration template.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steps:</td>
<td>Identify Overlapping Parts</td>
</tr>
<tr>
<td></td>
<td>Combine Configuration Packages</td>
</tr>
<tr>
<td></td>
<td>Document Combination</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Template</td>
<td>Configuration Template (intermediate)</td>
</tr>
<tr>
<td>Configuration Packages</td>
<td>List of Development Situations (updated)</td>
</tr>
</tbody>
</table>
Meta-Method for Method Configuration

| Frequency: | For each configuration template |
| Worker: | Process Engineer |

**Concepts:**
- **Base method** - is the systems engineering method chosen as the starting point for the configuration work.
- **Characteristic** - a delimited part of a development situation, focusing on a certain problem or aspect which the method configuration aims to solve or handle. Example: Legacy Integration {Yes, No}
- **Configuration Package** - a method configuration of the base method suitable for a characteristic’s value.
- **Configuration Template** - a combined method configuration for a set of recurrent project characteristics.
- **Development Situation** – an abstraction of one or more existing or future software development projects with common characteristics.
- **Value** – a possible choice in a characteristic. A value corresponds to one configuration package.

**Workflow Details:** Define Configuration Template

**IDENTIFY OVERLAPPING SECTIONS**
Combining several configuration packages into a configuration template entails working with sections of the base method, and sometimes sections from additional methods. Each configuration package covers aspects of the method relevant according to its characteristic. Since a development situation, and specific projects, often cannot be characterized by one single characteristic it is natural to receive configuration packages with overlapping prescribed actions, i.e., they focus on the same section of the systems engineering method. We need to identify these parts of the configuration template. However, if overlapping sections frequently occur they may be an indication of poor performing characteristics and a poor demarcation of configuration packages.

**COMBINE CONFIGURATION PACKAGES**
Combining configuration packages and their content, prescribed actions and artifacts, has to be adapted to the pattern of how configuration packages are chosen. Figure 35, below, illustrates the main patterns we have to consider. Five patterns with heuristics for combinations are presented. However, the third and fourth patterns are variants of the second pattern, that is, overlapping sections.
A Detailed Description of Method
for Method Configuration

Figure 35. Example of Selection Patterns When Combining Configuration Packages to a Configuration Template

1) One single configuration package is selected for a demarcated section of the base method. This is the ideal situation with no overlapping configuration packages. Consequently, it is straightforward to use the classification of the configuration package’s prescribed actions and artifacts, respectively, in the configuration template. For example, an activity classified as ‘Skip’ in the configuration package is classified as ‘Skip’ in the configuration template.

2) More than one configuration package is selected for the same section of the base method. Consequently, a combination of these classifications is needed. The first question is whether or not these classifications are in conflict with each other. If all configuration packages have the same classification of the overlapping prescribed actions there is no problem, actually we can use pattern number 1. In the opposite situation, where the classifications are in conflict with each other, we try to give priority to each of the chosen configuration packages, on the basis of the characteristics’ origin. We ask ourselves: Which characteristic is the most essential? Furthermore, if the answer does not solve the conflict, we prioritize the classification with the highest demand on the method. For example,
Meta-Method for Method Configuration

‘Perform as is’ is a higher demand of method-support than a ‘Skip’ with regard to reach the prescribed action’s purpose.

(3) More than one configuration package is selected for the same section of the base method, and at least one of the configuration packages has a require clause. A require clause is a dependency between two configuration packages. The classifications in the required configuration package are used as the base for the configuration package requiring. For example, consider two configuration packages, named CP1 and CP2. CP2 requires CP1 since the later contains an enhancement of the base method, which CP2 tailors more in detail. Hence, CP2 uses the classification of the content in CP1 as input, but the classification of CP2 is prioritized in conflicting classifications.

(4) At least two configuration packages have partly overlapping sections of the base method. A combination is needed if the classifications are conflicting. However, it is a variant of pattern number 2.

(5) No configuration package is selected for a specific section of the base method. This situation has two possible causes. First, there is no characteristic addressing this section of the base method. Second, the N/A-value could have applied to a characteristic resulting in a non-choice. Nevertheless, this non-choice is not an active classification of the base method and thus it is not possible to perform a classification of these parts in the configuration template. This problem should trigger a change request for a configuration package and an active classification of the base method.

DOCUMENT COMBINATION

The configuration template constitutes classified prescribed actions and artifacts that are a pre-configured base method. Since the combination found in configuration templates, in certain situations, involves reclassification of prescribed actions and artifacts, the classification cannot always be inherited from related configuration packages. Thus, it is important to document these classifications if the configuration template is to be reusable. Together with the classifications there should also be a short description of each classification, which complies with the guidelines for documentation of configuration packages. Thus, a description could in most cases be derived from the configuration packages, besides the reclassifications. When we document the configuration template, we add its profile to the list of development situations, since the configuration template corresponds to exactly one development situation.

Configuration templates do not have a clear-cut counterpart in RPW. We can choose to publish a tailored version of the RUP, a situational systems engineering method. However, since we receive a situational systems engineering method, it is difficult to use the workflow detail ‘Fine-tune Configuration Template’.

137
### 6.2.2.2 Activity: Validate The Configuration Template

**Purpose:**
- To validate the tentative configuration template in order to achieve a consistent configuration.

**Steps:**
- Validate The Configuration Template

**Input Artifacts:**
- Configuration Template

**Resulting Artifacts:**
- Configuration Template

**Frequency:** For each new or updated configuration template

**Worker:** Process Engineer

**Concepts:**
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.

**Workflow Details:** Define Configuration Template

---

**VALIDATE THE CONFIGURATION TEMPLATE**

As the configuration packages are combined into a configuration template their consistency has to be evaluated in the combined form. This evaluation focuses on a vertical integration and the prerequisites of each configuration package. We validate that each configuration package has sufficient input (artifacts), when it exists in the combination; each worker has to have necessary prerequisites in order to perform the prescribed action efficiently. If we find prescribed actions that do not have necessary inputs, re-classifications of one or more prescribed action and/or artifacts have to be considered. The central question is which prescribed actions and artifacts to re-classify. In this situation we have two options:

1. Consider the importance and complexity of the prescribed actions that do not have sufficient input (according to Figure 32 in the MMC-activity ‘Consider the Activity’s Importance and Complexity’). Can these prescribed actions be reduced in performance and thus change the prerequisites?
2. Trace the missing artifact to prescribed actions in earlier parts of the base method. Re-consider these classifications and ensure that the missing artifact is produced. Since prescribed actions have sequential dependencies, we might have to reconsider a chain of prescribed actions.
6.3 Section 3

The workflow details, activities and steps in this section are an operationalization of ‘Select Configuration Template’ and ‘Fine-tune Configuration Template’ in the RUP-specific MMC-framework (see Figure 28 in Chapter 5).

6.3.1 Select Configuration Template

Figure 36. Workflow Detail: Select Configuration Template

The purpose of this workflow detail is to:

- Select a configuration template in order to facilitate fine-tuning.

6.3.1.1 Activity: Characterize Project

<table>
<thead>
<tr>
<th>Purpose:</th>
</tr>
</thead>
<tbody>
<tr>
<td>To characterize a specific project to enable a comparison with existing development situations and their related configuration templates.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Create Project Profile</td>
</tr>
<tr>
<td>- Document Project Profile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Characteristic List</td>
<td>- Project Profile</td>
</tr>
</tbody>
</table>

Frequency: Before upstart of each project
Worker: Process Engineer
Concepts:
A Detailed Description of Method
for Method Configuration

- Base method - is the systems engineering method chosen as the starting point for the configuration work.
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.
- Value – a possible choice in a characteristic. A value corresponds to one configuration package.

Workflow Details: Select Configuration Template

CREATE PROJECT PROFILE
The project has to be characterized in order to select a configuration template. Since a configuration template is related to a development situation, which is an abstraction of projects, we try to conform the project in question to an existing development situation. Thus, we use the characteristic list and ask questions about the project, and make the project manager’s needs explicit.

Select a value for each characteristic in the characteristic list. The selected value should be the most significant for the project. If the characteristic is irrelevant for characterizing the project select the not applicable value (N/A). When the characterization results in several N/A values this might be a trigger for reconsidering one or several characteristics, characteristic values and configuration packages. Furthermore, a project profile and its closest match of available development situations might not yield a satisfying result. Subsequently, this could trigger a change request for a new or updated configuration template, where a new or updated combination of configuration packages is considered. Reworking an existing configuration template or creating a new one indirectly affects your set of development situations, since they have a relationship. An updated or new configuration template result in an updated or new development situation, respectively.

DOCUMENT PROJECT PROFILE
Characterizing a project results in a project profile. Each development situation as well as configuration template has a profile, which is used for comparison and for the selection. In order to compare the project with existing development situations the project has to be documented in the same way. A simple way is listing the selection for each characteristic since each development situation and configuration template contain such lists.

Documentation of project profiles has a second objective. Continuously documenting projects into profiles yields uniform material for the abstraction of new or modified development situations. If we have several projects with the same profile they could be a trigger for a new or updated development situation with the related configuration template.
Example

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Selected Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy Integration</td>
<td>Yes</td>
</tr>
<tr>
<td>Degree of Interaction</td>
<td>Dynamic/Affectable</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Characteristic&lt;sub&gt;n&lt;/sub&gt;</td>
<td>Value&lt;sub&gt;n&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

6.3.1.2 Activity: Match With Configuration Template

**Purpose**
- Select a configuration template in order to enable further fine-tuning.

**Steps**
- Map Discrepancy
- Select Best Match

**Input Artifacts:**
- Project Profile
- Configuration Templates
- List of Development Situations

**Resulting Artifacts:**
- Configuration Template (selected)
- Discrepancy Matrix

**Frequency:** Before upstart of each project

**Worker:** Process Engineer

**Concepts:**
- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Development Situation – an abstraction of one or more existing or future software development projects with common characteristics.
- Value – a possible choice in a characteristic. It corresponds to one configuration package.

**Workflow Details:** Select Configuration Template

**MAP DISCREPANCY**
Use the set configuration templates and/or the list development situations as input. Each development situation and its configuration template have a unique profile with regard to the characteristic list. The selection of configuration templates should be based on a comparison between the project’s profile and the development situations’ profiles.\(^{42}\)

\(^{42}\)There is a one-to-one relationship and thus makes no difference whether we compare the project’s profile with the development situation or the configuration template’s profile.
A Detailed Description of Method for Method Configuration

Use the characteristics as the starting-point and map the discrepancies between the project and development situations. Discrepancies can be classified as none or existing. Iterating each configuration template gives a matrix with differences. When a discrepancy exists, document the type of difference (the value of the configuration template) and whether this discrepancy is important or not. Use the description of each of the configuration templates and their values as an aid for understanding the purpose of the template in broad outlines.

SELECT BEST MATCH

The matrix illustrating discrepancies between the project profile and development situations is the starting-point for rating configuration templates. Give the highest rating to the configuration template with the fewest discrepancies. The basic rule is to select the configuration template with the highest rating. However, a list of rated configuration templates is not unambiguous. Our project may almost fit into several development situations and then we will have a range of possible configuration templates. Select the configuration template that is the best approximation of the project profile and involves an insignificant amount of fine-tuning to turn it into a situational systems engineering method. However, it is important to be aware of the fact that there is no direct inheritance between configuration packages and the configuration templates. Hence, the values illustrating the configuration template are an approximation of its content, since reclassification could have been made when creating the configuration template.

If we cannot find a configuration template that has an acceptable match this could indicate a situation where the MMC-activity ‘Manage Change Request for Configuration Template’ is triggered.

6.3.2 Fine-tune Configuration Template

![Figure 37. Workflow Detail Fine-tune Configuration Template](image-url)
The purpose of this workflow detail is to:

- Create a situational systems engineering method to be used in a specific project.

6.3.2.1 Activity: Fine-Tune The Configuration Template

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>To adjust a configuration template in order to fit a specific project.</th>
</tr>
</thead>
</table>
| Steps:   | Identify Relevant Discrepancy  
|          | Re-classify Prescribed Actions and Artifacts  
|          | Document Classification |

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
</table>
| Configuration Template  
| Project Profile  
| Discrepancy Matrix | Situational Systems Engineering Method (intermediate) |

Frequency: For each selected configuration template
Worker: Process Engineer

Concepts:

- Configuration Package - a method configuration of the base method suitable for a characteristic’s value.
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Situational Systems Engineering Method - is a fine-tuned and adapted Configuration Template used in a project.

Workflow Details: Fine-tune Configuration Template

IDENTIFY RELEVANT DISCREPANCY

From the selection of configuration templates, any discrepancy between the selected configuration template and the project profile has been documented. Consider which of these discrepancies are relevant to fine-tune. Initially, focus on the discrepancies that are considered to be rather large interference in the original configuration template. One possible starting-point is the matrix with the mapped discrepancies. If there are any ‘existing’ classifications they could indicate fine-tuning needs. Of course, there can be a need for minor adjustments that we find through sequentially considering each prescribed action and artifact in the configuration template. However, it is important to balance the quality improvement of these adjustments and their cost.
RE-CLASSIFY PRESCRIBED ACTIONS AND ARTIFACTS
Discrepancies between a configuration template and the project’s needs are due to differences in classifications. When these differences are significant reclassifications in the selected template should be considered. Re-classification of a prescribed action or artifact is made according to the importance and complexity aspects. The importance and complexity regard the prescribed action or artifact’s role in the specific project. Consequently, we agree with the same line of reasoning as used for defining configuration packages (see Section 6.1.2.2). Thus, the analytic tool in Figure 38 is utilized for the reclassification. As when configuration packages are defined, the X-axis constitutes the consideration of importance; whether this prescribed action has higher or lower importance than in one of the organization’s normal projects. The second aspect is about the project’s complexity. The movement on the Y-axis describes how complex the project is with regard to fulfillment of the prescribed action’s purpose. The intersection represents a possible classification in Table 7, and thus a possible reclassification. Importance and complexity are mapped to the attention dimension and the enhancement—dimension, respectively.

```
Importance

Low            High

Low

Consider reduced

Consider skip

Perform as is

Consider emphasize

High

Consider extended
```

Figure 38. Analyze of Prescribed Actions

It is important to consider both the process as well as the product view during the tentative reclassification. The prescribed action and its deliverables are intertwined since activities need artifacts as prerequisites for producing new or
modified artifacts. For example, reclassifying a prescribed action from ‘Skip’ to ‘Perform as is’ could also involve reclassification of the artifacts belonging to the prescribed action.

The same restriction on the tool in Figure 38 and the classification system in Table 7 is imposed as when a configuration package is defined. It is not possible to reduce or enhance a step, since it is a prescribed action on the lowest layer of granularity. The shaded areas in Figure 38 and Table 7 illustrate this restriction.

### Table 7. Performance of Prescribed Actions

<table>
<thead>
<tr>
<th>Attention given to prescribed action</th>
<th>Reduced</th>
<th>As is</th>
<th>Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Skip</td>
<td>Skip</td>
<td>Skip</td>
</tr>
<tr>
<td>Normal</td>
<td>Perform reduced</td>
<td>Perform as is</td>
<td>Perform extended</td>
</tr>
<tr>
<td>A lot</td>
<td>Emphasize reduced</td>
<td>Emphasize as is</td>
<td>Emphasize extended</td>
</tr>
</tbody>
</table>

### DOCUMENT CLASSIFICATION

The situational systems engineering method constitutes classified prescribed actions and artifacts aimed at supporting a specific project. In order to make the situational system engineering method a tool for guidance these classifications have to be documented. Since we reuse a selected configuration template, it is the starting-point. Modifications of reclassifications have to be added in order to make the inherited configuration project specific.

#### 6.3.2.2 Activity: Validate The Situational Systems Engineering Method

**Purpose:**
- To validate the situational systems engineering method in order to achieve a consistent configuration for a specific project.

**Steps:**
- Validate The Situational Systems Engineering Method

<table>
<thead>
<tr>
<th>Input Artifacts:</th>
<th>Resulting Artifacts:</th>
</tr>
</thead>
</table>
A Detailed Description of Method  
for Method Configuration

<table>
<thead>
<tr>
<th>Frequency:</th>
<th>For each situational systems engineering method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worker:</td>
<td>Process Engineer</td>
</tr>
</tbody>
</table>

**Concepts:**
- Configuration Template - a combined method configuration for a set of recurrent project characteristics.
- Situational Systems Engineering Method - is a fine-tuned and adapted Configuration Template used in a project.

**Workflow Details:** Fine-tune Configuration Template

**VERTICAL EVALUATION**

A situational systems engineering method based on a configuration template is a sequence of prescribed actions. These prescribed actions have inter-dependencies with each other. Validate that each activity has sufficient input (artifacts) to be performed efficiently. If we find activities that do not have necessary inputs re-classifications of one or more activities and/or artifacts have to be considered. The central question is which activities in which part of the situational system engineering method should be re-classified. The situation can be solved by:

1. Considering the importance and complexity of activity that does not have sufficient input (according to Figure 32 in the MMC activity ‘Re-classify Prescribed Actions and Artifacts’). Can this activity be reduced in performance and as a result will not have any use of the artifact?
2. Tracing the missing artifact back to its producing activity. Re-consider its classification and ensure that the missing artifact is produced. The artifact might have to be traced back to more than one activity. There can be a chain of activities classified in such a way that the artifact cannot be produced.
CHAPTER SEVEN

7 EPISODES OF USE

During development efforts different tests of the meta-method have been performed in order to validate theoretical knowledge through practical use. As with every development project, its resulting artifact passes through different stages of maturity and thus our tests have ranged from smaller pieces of the meta-method to the more complete version. To some extent, these tests constitute different episodes of use, however with different degrees of realism.

In an honest way, we can conclude that each version of the meta-method has had flaws in different aspects, and this is natural. The purpose of these tests has been to sort out the weak as well as the strong parts of MMC. The interesting questions are how, why and what influences these tests have had on the meta-method design. Therefore, the objectives of this chapter are to present three selections of these episodes as arguments for the design decisions taken as well as illustrate the meta-method’s value during use. Doing so we are able to point at issues where MMC needs to be or has been improved as well as the type of prerequisites that are important, before starting the configuration work. This chapter is based on log-files from two workshops and one system engineering project where configuration work was conducted (see Section 2.2.3). The focus of each of these analyses differs, since they had different aims. The vertical axis in Table 8 contains three different focuses, and together with the units of analysis they constitute a matrix. This combination was justified in Section 2.2.3.3, but the discussion is recapped in brief below.

During the first workshop, the meta-method project team tested the core parts of MMC. Therefore, we chose to structure this analysis as a comparison between prescribed actions and their instantiated actions and effects (Goldkuhl, 1999). However, one should be aware of the fact that these parts of the meta-methods were only rough prototypes at this point in time. Hence, ‘prescribed actions’ should be liberally interpreted. A two-day workshop was set aside for testing and further development of concepts and drafted procedures. Small configurations were made on demarcated sections of the standard systems engineering method. The input for configurations was different projects at Volvo IT. The practitioners from Volvo IT
had made an inventory of necessary projects to support with configurations. However, during this workshop, the resulting configurations were considered to be a bi-product, and no complete configurations were delivered.

Table 8. Units of Analysis and Focus

<table>
<thead>
<tr>
<th>Focus</th>
<th>The Early Workshop</th>
<th>The Large Workshop</th>
<th>The Personnel System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action-Effect</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Concept Integration</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Method/Tool Integration</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

The second workshop was more complex in its set-up. The plan was to create one brick in Volvo IT’s battery of reusable method assets, a configuration package. Thus, this meant a test of section 1 in MMC. Returning to the MMC framework in Chapter 5 we find that this section is about managing a change request for configuration packages and to defining a configuration package. However, this test also included the use of tool support from Rational Software, using Rational Process Workbench. Thus, the focus was not solely on the meta-method as such; rather it was a test of how well MMC could complement this tool. Therefore, we were interested in the meta-method project team’s work pattern as well as the correspondence between concepts in the meta-method and the tool and the focal areas of the latter. A Grounded Theory approach was a suitable strategy for elaborating on the pattern of this second workshop, eliciting the actual way of working.

The third episode was the most realistic one, at least with regard to its size and final product. It was from the development of a web-based personnel system at Örebro University and the Department ESI. The author played the roles as project manager and method engineer during this project. From a method configuration point of view, this project resulted in five characteristics with related configuration packages. The analysis of this episode shared focus with our first workshop, however in this case we would be dealing with a larger configuration and a later version of MMC.

7.1 The Early Workshop
The development of MMC has in several phases been carried out as workshops. In this way, we have managed to gather relevant competence to a demarcated space in time. This episode is from an early workshop at Volvo IT. As described above it was a two-day session with five people from the meta-method project team; three researchers from the VITS research group, the author included, and two practitioners from the MAPS team at Volvo IT. The latter team had prepared brief ideas about configuration needs with an origin in Volvo IT’s project portfolio. The meta-method
project team used different selections of these ideas depending on the tested parts of MMC. For example, when testing the development situation and characteristic concepts a range of projects was discussed, while only a delimited selection of configuration packages where discussed in detail.

The purpose of this workshop was to test MMC’s concepts and prescribed actions when working with configuration packages. Consequently, during this workshop, the meta-method project team gathered experience of the initial usage of the meta-method. Furthermore, the team also gathered experience of how to elaborate the concepts, the prescribed actions and the framework even further. Naturally, these experiences have been used to improve the meta-method. The main focus, if we refer the tentative framework in Section 4.3, was on MMC sections 1 and 2. However, at that point in time there was no RUP specific version of MMC. Thus, we will refer to the framework found at the end of Chapter 4 during this analysis, if nothing else is explicitly stated. Concretely, MMC sections 1 and 2 involve ‘Identifying Development Situations and Characteristics’, ‘Administrating Configuration Packages’, ‘Selecting Configuration Packages’ and ‘Combining Configuration Packages’.

7.1.1 The Process

At the beginning of this workshop, there were no characteristics or configuration packages to extend or update; the meta-method project team started out with a blank sheet. Therefore, a description of the situation at Volvo IT was the starting point, trying to find development situations and characteristics. As input, the team used the rough inventory of configuration needs at Volvo IT, complemented with a deeper characterization of the Internet-based software artifact (see Karlsson, 2001a for details). The latter was mainly based on experience of the configuration work described in Section 1.6. Consequently, the meta-method project team analyzed present and future projects of different sizes and complexity. The team elicited tentative characteristics through a brainstorming approach, based on the project descriptions, which resulted in a list of characteristics. This list contained several doubles, hence it became important to crosscheck characteristics. Recalling the definitions of the characteristic and the configuration package concepts it is not in the interests of the MMC user to have more than one characteristic for a specific configuration package.

The description of each characteristic became valuable; because then it was possible to conclude in which cases one or more characteristics had the same meaning. Nevertheless, it boiled down to how the team wanted to design the content of the reusable method assets. When several similar characteristics existed, a choice had to be made (thus making design decisions about the content of the characteristic set) on which one to use. The meta-method project team based their decision on the characteristics that they thought illustrated the development situation best. During this task, another design decision was encountered. The characteristics themselves were not the only doubles to deal with. Throughout the workshop, characteristics were elicited with several sets of possible dimensions, which all could be used for
Episodes of Use

illustrating that specific part of the development situation. Yet again a decision had to made, this time about which set of values to use. This analysis and design pattern was an iterative process in which both the characteristics and the dimensions were questioned and reformulated several times.

The meta-method project team used characteristics formulated both as short statements and as questions, where the latter could be answered with one of the characteristic’s values. For example, the same characteristic came up designated as ‘Maintenance’ which was reformulated into a question ‘Is this a maintenance project?’ In some cases, the use of questions made it easier to find and discuss the values constituting a possible dimension.

During analysis, the team chose activity as the level of granularity. This decision was mostly based on earlier experience of the amount work each level would lead to. Furthermore, each classification had to be documented. Throughout this workshop, the meta-method project team used Excel spreadsheets as the tool. One reason was partly that Rational Software’s Process Workbench had not been released yet, partly because Excel spreadsheets facilitated modifications of the documentation pattern at low cost. In addition, more advanced tools would impose an implicit way of focusing the base method, which would have restricted the development process. However, the obvious drawback with the chosen tool was the non-existence of automated support for tracing dependencies between prescribed actions, in our case primarily activities. Hence, the meta-method project team spent a lot of time discussing which activities depended on each other and how they were affected by the configuration.

An example of a selected section from the meta-method project team’s documentation is found in Table 9, below. It illustrates three tentative configuration packages, and a possible combination. Thus, the combination is a configuration template. Each activity in the base method has been listed and classified using the analytical tool and the classification system elaborated in Section 4.2.4.1, which was used as input in this workshop. The name of the characteristic is stated in the header of columns four to six, before the colon. After the colon, we have specified the value and together with the characteristic’s name they constitute the name of the configuration package. The first of these columns contains the characteristic Reuse Strategy, with the value Component Based Development. In the second column, we find the characteristic Legacy Integration, and the value Yes. Finally, the third column illustrates the Lifetime characteristic, with the value Fast Path. The Lifetime characteristic centers on the need for developing a project with a tight time-limit, and subsequently the amount of overhead should be minimal\(^{43}\). The priority is to deliver the most essential functionality of the artifact. Furthermore, the table below presents a demarcated section of RUP. Combined with the configuration packages, it illustrates tentative classifications of activities in the Requirements discipline, more specifically, the workflow details ‘Manage the Scope of the System’ and ‘Refine the System Definition’.

\(^{43}\) One can argue that this should always be the case, however here it is considered as an important goal of the development situation.
<table>
<thead>
<tr>
<th>Workflow</th>
<th>Workflow Detail</th>
<th>Activity</th>
<th>Reuse Strategy: Component Base Development</th>
<th>Legacy Integration: Yes</th>
<th>Lifetime: Fast Path</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Manage the Scope of the System</td>
<td>Prioritize Use Cases</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td>Emphasize as is</td>
<td>Perform as is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Find Actors and Use Cases</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td>Perform reduced</td>
<td>Perform as is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage Dependencies</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td>Skip</td>
<td>Perform as is</td>
</tr>
<tr>
<td>Refine the System Definition</td>
<td>Detail a Use Case</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Detail the Software Requirements</td>
<td>Emphasize as is (Inc./Elab.)</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td>Emphasize as is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Model the User Interface</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td>Skip</td>
<td>Perform as is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prototype the User Interface</td>
<td>Perform as is</td>
<td>Perform as is</td>
<td>Skip</td>
<td>Perform as is</td>
<td></td>
</tr>
</tbody>
</table>

Table 9. Tentative Configuration
Episodes of Use

Starting out with ‘Manage the Scope of the System’ this specific workflow detail contains three activities, which are classified for each of the configuration packages. However, the first two characteristics do not have any affect on this part of the base method. We have to find use cases, prioritize between use cases and manage their dependencies regardless of whether we are performing Component Based Development or not. Therefore, they are of normal importance and not considered more complex than normal simply because of the reuse strategy. It might be possible to argue that the importance of investigating dependencies between use cases could be higher than normal but every classification must be considered in the specific business context.

The third configuration package, Fast Path, has a different focus than the other two. With a focus on the most essential functionality in an artifact, it becomes highly important to prioritize between use cases. At the same time, the meta-method project team wanted to reduce efforts put into gathering use cases; hence focusing on the core user groups. Thus, the complexity in the activity ‘Find Actors and Use Cases’ is reduced. When it comes to ‘Manage Dependencies’ the team considered the activity to have low importance in the respect that there was no intention to manage changes with this focus. Consequently, this activity was not considered complex at all, and combined with low importance it was classified as ‘Skip’.

Turning our attention towards the workflow detail ‘Refine the System Definition’ we have another four classified activities. These are ‘Detail a Use Case’, ‘Detail the Software Requirements’, ‘Model the User Interface’ and finally ‘Prototype the User Interface’. For the configuration package ‘Component Based Development’ the meta-method project team made a classification of ‘Refine the Software Requirements’ that differs from the base method. The characteristic’s value emphasizes a reuse strategy, a strategy that had so far not been frequently used within the organization. Subsequently, with regard to the organization’s current status it was considered as more important than usual development activities. Hence, the team suggested that this activity was classified ‘Emphasize as is’, thus it would be performed with specific care, i.e., it would receive more attention than normal.

For the configuration package Legacy Integration Yes and the presented section of RUP, the meta-method project team did not make any classifications that diverged from the base method. Thus, according to this classification the configuration package does not affect a forthcoming combination. However, this is not entirely true, as we will see from the discussion below. Moving on to the last configuration package, Fast Path, the activities ‘Model the User Interface’ and ‘Prototype the User Interface’ have been classified as ‘Skip’. The purpose of the former activity is ‘to build a model of the user interface that supports the reasoning about, and the enhancement of, its usability.’ (RUP 2001 documentation) However, with a focus on quick implementation usability is not prioritized. In such a situation, reasoning and enhancement together with the information system’s users

---

The classifications made in this section should not be considered as the final classification and naturally classifications are context specific.
Meta-Method for Method Configuration

(considering the second of these activities) is kept at a minimum, subsequently these activities are considered to be of low importance.

7.1.2 Design Implications

This workshop resulted in a foundation for a more profound description of the MMC framework, and how it could be turned into a detailed description. The work with development situations and characteristics was an operationalization of ‘Identifying Development Situations and Characteristics’ in Figure 25. During this workshop, ‘Identifying Development Situations and Characteristics’ became three activities ‘Identify Potential Changes,’ ‘Evaluate Changes’ and ‘Check with Existing Characteristic’ found in Section 6.1.1. ‘Identifying Development Situations and Characteristics’ was later to become ‘Manage Change Request for Configuration Package’ in the RUP specific version. The reason for presenting these activities in a workflow detail, is the iterative pattern used during this workshop. The meta-method project team had to continuously shift focus between these activities in order to elicit characteristics, their values, and to eliminate possible doubles.

During this early workshop, the meta-method project team used an operationalization of configuration packages, which included the complete base method. This meant that the team made a classification decision about each and every activity (the chosen level of granularity) in the base method. Thus, the work was time-consuming, especially since the team discussed a classification for each combination of activities and phases in the base method (inception, elaboration, construction, and transition). This meant four classifications of every activity. In later versions of the MMC description, we therefore recommend rough demarcations of affected parts of the base method, in order to reduce the amount of work. This also improves the work with combinations, which is explained below. Furthermore, when using MMC in later episodes (for example, in Section 7.3) we do not create a classification matrix where activities are discussed for each phase. Instead we focus on the extent to which a specific activity adds value during any phase, and aligns with the phase structure in the base method. For example, the question is not whether the activity ‘Assess Target Organization’ should be performed during the inception or elaboration phase, rather whether performing such an assessment at all adds value.

The meta-method project team did not have any tool support that incorporated the basic thoughts in MMC. Thus, the documentation task was not efficient and we can conclude that tool support is needed. Several options could be found. At this stage, the attention was on Rational Process Workbench, which at that point in time was a forthcoming tool from Rational Software. Consequently, it aligned with the chosen base method at Volvo IT. A second option is an in-house developed tool supporting the foundation in MMC, where the base method is not forgone. Finally, plug-ins could complement existing tools and hence incorporate MMC into other tools. However, with regard to the demarcation in Section 1.12 it is not within the scope of this thesis to develop a tool.
One of the more practical experiences gleaned from the efforts to combine configuration packages was how to handle parts of the base method that were not affected by the specific configuration package. Using a process perspective and the activity level of granularity as the starting point for classification generates about two hundred standpoints. Many of these are ‘Perform as is’ classifications; they should not have any affect on forthcoming combinations, at least not from the specific configuration package’s point of view. However, when we have a set of selected configuration packages to combine, a ‘Perform as is’ classification is ambiguous. Since we have to prioritize between classifications made in separate configuration packages, it is not obvious when an activity really matters for a characteristic other than when it is classified as deviant from the base method. Of course, the description of each classification is useful, however it is not efficient if we have to read the argumentation for each classification. Especially if we consider an argumentation for performing activities according to the base method since the characteristic at hand does not affect that part of the situational systems engineering method. If we put this situation into practice: Why argue for the classification of an activity that is not relevant for the characteristic in question? It is not far-fetched to consider situations in which this type of argumentation is left out. Hence, to distinguish between activities that are and that are not affected by a characteristic we needed another classification term. Therefore, we introduced ‘n/a’ for the former and reserved ‘Perform as is’ for situations in which we have actively made a classification. This extension of MMC was not implemented during this workshop. However, as an effect, the activity ‘Identify Relevant Activities’ was introduced into the workflow detail ‘Administrating Configuration Packages’, see Section 5.3 (later turned into Define Configuration Package in the RUP specific version).

### 7.1.3 Experiences Gleaned

Finding durable characteristics was, as one team member expressed it, ‘tricky’. Subsequently, much of the initial time had to be spent on the elaboration and evaluation of these characteristics. In order to structure the search for characteristics the meta-method project team needed a common screen to use for each project in question. The selection of a screen was based on our earlier pre-case work at Volvo IT in Skövde. One significant part of this work was to characterize the Internet-based software artifact, which we have referred to above. However, it was part of the operationalization of the analytical framework based on actability elaborated by one of the VITS-researchers participating in this project (see Ågerfalk, 1999; Ågerfalk et al., 2002; Karlsson et al., 2001a). Thus, we can conclude that it is important to use an explicit framework in order to have focal areas when elaborating characteristics of development situations.

The meta-method project team started this workshop based on rather brief knowledge about the present situation at Volvo IT, if we consider inter-subjective knowledge. Thus, the brainstorming approach was highly important. However, it is recommended to make a detailed inventory of the organization’s project portfolio, in which the method engineer considers historical, present and future projects. This
would facilitate grouping projects into development situations and finding characteristics. The meta-method project team obviously had problems with the first part due to the rough inventory.

One experience from this workshop is that the description of how a characteristic should be used is important. Hence, when using change requests in MMC it is important that they really illustrate the changes needed and why they are needed. Thus, this type of description should address the problems or aspects in projects that are not efficiently supported through present configurations. Furthermore, formulating characteristics as questions could make them more intuitive to work with, since the purpose is to use them as a battery of questions when selecting a configuration template.

As stated above, the meta-method project team used activity as the level of granularity during analysis. From the discussion during this workshop we found it important to decide on a guideline for where to start an analysis. This is incorporated into MMC in Section 5.2 when the MMC-framework is converted into an RUP-specific version.

None of the configuration packages or their classifications, found in Table 9, is obvious. They caused a lot of discussion in the meta-method project team. During the work with the configuration package ‘Fast Path’ a discussion in principle about classifications was raised. MMC as such does not state whether or not the classifications, and consequently the situational systems engineering method, should be based on existing or a recommended way of using the base method. The used procedures in the organization might not be the desired ones, especially not when the base method is under implementation. The ‘Fast Path’ configuration package has existed in several versions. The meta-method project team has gradually moved from classifications based on existing procedures in the organization towards using desired ones. However, this is not a simple decision since the desired procedures have to be adopted. Otherwise, there is no point in making such configurations if they are only placed on the shelves. Therefore, such a change might be performed gradually. In our case the ‘Lifetime’ characteristic and its configuration package have been deprecated during later configuration work.

The consequence of this deprecation is that the combination in Table 9 is not in use. Subsequently, it is relevant to question why it is used as an example. The reason is that it illustrates the fact that every selection of configuration packages might not be relevant. It is essential to question the produced combinations, so that we can find a development situation that justifies a specific selection of configuration packages. From our test configuration, this is obviously not always a simple question to answer. In our case it also involved a discussion about the existence of one of the configuration packages as such. However, it was not that specific question that triggered the discussion of whether or not it was a relevant combination of configuration packages. Rather the trigger was an issue that grew out of occasions when the meta-method project team intended to combine classifications and ended

---

45 This is a matter of implementation; how to generate organizational change when implementing a system engineering method. Thus it is beyond the scope of this thesis.
Episodes of Use

up with priority-problems between the chosen configuration packages. There was no natural priority, and several times the classifications were contrary. When examining the characteristics we can conclude that ‘Reuse Strategy’ and ‘Life-time’, at least with the values ‘Component Based Development’ and ‘Fast Path’, is a badly chosen combination. The former configuration package aims at a long lasting and flexible solution, while the latter emphasizes generating a quick solution. The intention behind a quick solution is often to produce an artifact for temporary needs. Hence, the meta-method project team had contradictory goals and this combination never became anything else than experiences about MMC usage.

7.2 The Large Workshop

This workshop was held during the latter part of the development process of MMC. It was planned as a final, more complete, test of the meta-method. At this point in time Rational Software had delivered a new tool, Rational Process Workbench (RPW), to the software market. Thus, the meta-method project team had a tool suitable for modeling the content of our base method, RUP. This tool had been developed independent of the MMC project. However, the meta-method project team intended to use it when implementing configuration packages and templates. An assumed benefit when using an ‘off-the-shelf’ systems engineering method is the availability of tools. Consequently, this was a natural decision. From the meta-method project team’s view there was no need to develop their own Computer-Aided Method Engineering tool (as for example in Rossi, 1998), if MMC could complement RPW. Therefore, during the MMC-project the meta-method project team had to pay attention to Rational’s thoughts about how they intended to implement changes to RUP through this tool. Time had been spent before this workshop, on mapping the MMC concepts to the concepts in RPW. However, when the workshop started, this work was not finished. It should also be stated that an early version of the CAME tool was used, which could differ from the final release.

Since we were going to test several new things during this workshop, the meta-method project team was extended. In addition to the normal constellation of the meta-method project team, consultants from Rational Software Nordic AB as well as the chief developer for RPW from Rational Software Corporation in Vancouver participated. During this workshop, two researchers participated, and the author’s role was mainly to document the way of working and the concepts used in order analyze how MMC could complement RPW.

With this set-up we dare say that the workshop team consisted of competent method users. Four days were set aside for conducting this test, which included training in the CAME tool. The first two days were spent on tool introduction. At the same time we got the opportunity to map our concepts closer to each other. During these days we also considered what would be a suitable target for a configuration package. The workshop team’s target was a configuration package in which the project management discipline was tailored for Volvo IT’s needs, implementing an in-house developed project management model, designated PCM, into RUP. Thus, the author’s goal existed side by side with the workshop team’s
Meta-Method for Method Configuration

goal. The characteristic for a development situation was not explicitly specified during the workshop. Reconstructed, as below the workshop team’s concern was the configuration package implementing the Yes value.

```xml
<characteristic name = "PCM-based Project Management?">
  <value valueid = "Yes" cpid = "Project Management Based on PCM"></value>
  <value valueid = "No" cpid = "Project Management Based on standard-RUP"></value>
</characteristic>
```

Unfortunately, the workshop team did not, during this workshop, reach a completed configuration package nor was MMC empirically grounded i.e., all steps in meta-method were not tested. Looking back at the workshop we can be even more frank and state that our meta-method was given rather low priority.

7.2.1 The Process

The process of the second workshop is illustrated in Figure 39, below. It has been elicited from the author’s log-files and structured into the paradigm model prerequisites-action-consequence (Strauss and Corbin, 1990). This choice is reasonable for method-work since it includes prescribed actions that transform input into output. Through this pattern we are able to address strengths and weaknesses in the workshop team’s way of working and where MMC is or could be a complement to RPW.

At a quick glance this process seems well structured. It contains two blocks of activities named ‘Configuration’ and ‘Validation’ that together constitute an iterative pattern. Nevertheless, this pattern did not result in the expected results. The main reason for this is found in the ‘Analyze’ activity and the workshop’s resources.

By ‘Analysis’ we mean the investigation of the measures needed to create a configuration containing PCM (a configuration package using the MMC concept or a Plug-In using the RPW concept). For example, suppressing an activity within a workflow detail to the benefit of a prescribed action in PCM. These measures were carried out through RPW in the ‘Implementation’ activity. Subsequently, specific concepts, prescribed actions, and artifacts were deprecated, while others were introduced. Technically, this involved modeling method elements using Rational Rose (RPW is a tool plug-in). Furthermore, activities and their steps were stored as HTML-pages, thus these had to be created or rewritten during implementation. These new prescribed actions were based on supporting documents containing PCM.

The second block of activities, ‘Validation’, contains two activities: ‘Publishing’ and ‘Evaluation’. By publishing we mean the generation of a new version of the implemented configuration package built with HTML pages; hence, presented according to RUP-standard46. On several occasions, the workshop team made the decision to test publish the tentative Plug-In. The Plug-In was not

46 Assuming that we designed the web pages according to Rational Software’s guidelines.
complete, however it made it possible to review the results produced so far and consider whether or not they aligned with the workshop team’s intentions. This type of decision became recurrent in order to make reviews, the ‘Evaluation’ activity, of the implementations in the CAME tool. The pervading characteristic, of all the published versions, was that none of them completely corresponded to the team’s intentions.

Figure 39: The Process During The Second Workshop

During this specific workshop, tool-competence was the bottleneck, which affected the process described above. Of course, each team member learned to use the CAME tool little by little, but the main body of competence was still the tool developer from Rational Software in Vancouver acting as tool mentor. He was only available during this workshop and could not participate in a post-workshop session, which lead to a lower priority of MMC to the benefit of learning the tool. Nevertheless, a great deal was learned about how to integrate MMC with the CAME tool, its concepts and implicit procedures; what was possible to do in the tool and the focal areas of the tool.

7.2.2 Analysis Based on RPW and MMC

Analysis can be performed with different focuses. Thus, it is interesting to investigate discrepancies between the focal areas incorporated into RPW and the proposed focal area in MMC. We can conclude that there was no one-to-one
mapping of the focal areas between RPW and MMC. Otherwise, we could have stated that MMC had been used through the use of RPW.

Returning once again to the author’s log files we find a conceptual focus of the process model using RPW. We stated above that the ‘Analysis’ activity was about the investigation of the measures needed to create a configuration of PCM. Behind this statement two types of analysis are found, one with a content focus and one with a technical focus. Into the first category we sort all measures with enhancement or reduction of the method content as well as conceptual issues, thus mapping our concept of project management to concepts in RUP. Even the exchange of prescribed actions in RUP is modeled as elements with content, which in the most basic form is descriptions based on concepts. In the second category, we allocate technical issues concerning the implementation. Here, we find measures concerning graphical issues, relevant when the method is presented in the RUP shell using HTML pages. Of similar concern are the supporting artifacts that the online RUP documentation links to, which are built from different files. Many of these considerations have an impact on how efficient future maintenance of the configuration packages will be. A discussion was raised on how Volvo IT’s project management model should be presented incorporated into their versions (configuration templates) of RUP. The workshop team had two main options, either to present the complementing parts at one place in the method or to spread them over different parts of RUP. The latter choice entails a larger amount of integration of the introduced parts; this was a matter of balancing integration and maintenance costs.

If we compare this focal area with that of MMC we find a difference. Using MMC we intend to focus on the differences in goals that exist between a characteristic of a development situation and the base method. Based on these differences we derive the need for reducing or enhancing prescribed actions in the base method, where these actions could exist on different layers of granularity. Referring to Brinkkemper (1996) we can see that prescribed actions, in this case activities, are built on concepts. Since MMC does not contain conceptual modeling we do not have a complete bridge in MMC to the focal areas used in RPW. However, the configuration package in MMC is a list of classified prescribed actions, which could be used for modeling prescribed actions in RPW. Explicit goal-oriented conceptual modeling could be proposed in order to strengthen MMC. Then, the conceptual modeling would have a focus founded in why concepts are relevant to use and how they are valuable during a project of a specific kind.

We could exemplify this through the goal of the activity ‘Lifecycle Milestone Review’ found in the RUP’s project management workflow. The purpose of that activity is ‘to review the state of the project at the end of a phase, and determine whether the project should proceed to the next phase.’ This goal can also be found in Volvo IT’s PCM model and is not a question of enhancing the base–method, but

---

47 Should not be mixed up with the mapping of the concepts used for the final artifacts, configuration packages, templates and plug-ins. This investigation is about the focal areas used during the process of reaching the final artifact.
rather of fulfilling the goal using a different set of concepts. This means that the activity is 'performed as is' according to MMC. Regarding concepts, the ‘milestone’ concept is used in RUP for achieving this goal, while in PCM it is named ‘gate’. Subsequently, an exchange of concepts would be possible.

7.2.3 *Mapped Concepts*

This workshop reduced the uncertainty about the differences between concepts in our meta-method and concepts in the tool provided together with RUP. During this workshop these two sets of concepts were discussed and mapped closer to each other. Thus, this part could be seen as a combination of grounding MMC empirically and externally.

Rational Software’s main concept for building reusable method assets is Plug-In. This concept is comparable to our configuration package, and technically it facilitates visual modeling of the systems engineering method’s content. In the workshop team’s version of RPW this meant modeling the base method from disciplines down to activities, based on the process view. In RPW, the focal areas are the elements in RUP’s process model (see Figure 27 in Section 5.1), for example a specific activity, a role and their relationship. However, in RPW it was not possible to model steps. Each activity is implemented as a HTML page in RPW, and this HTML page contains the step description. Thus, in order to configure steps these HTML pages had to be updated (using a tool other than RPW). However, we consider this to be a technical obstacle, and not a conceptual issue. Thus, it was possible to map our interpretation of the level of granularity when classifying prescribed actions.

The classification system in MMC was enabled mapping to implementation actions in RPW. Through the tool, it is possible to suppress selected elements from the base method, which is necessary in order to implement a ‘Skip’. An element could be a workflow detail, an activity as well as an artifact. The same technique is used in order to accomplish a ‘reduced’ section of the base method, if the layer of granularity regards an activity or above. Sections on the layer below the current layer are suppressed modeled which results in a reduced prescribed action. When an activity is to be performed, reduced selected steps are suppressed. In order to enhance the base method (RUP) it is possible to build completely new steps and relate them to the Plug-In and subsequently build new aggregations of prescribed method actions. The attention dimension in MMC (see Table 6 for example) only involves one further consideration. We have already addressed the ‘Skip’ alternative, which is quite straightforward to implement. However, the implementation of an ‘Emphasize’ is not that evident how to support. Actually, the attention dimension in MMC is a recommendation for how to perform a prescribed action, not a change in the prescribed action as such. Therefore, this recommendation can be placed in the method description, which could then be realized through the step description. Hence, these descriptions are complemented with additional text that discusses the importance of different activities, which could improve method adaptation. Extending this line of reasoning it would be possible to
facilitate the support of further method adaptation during the implementation of the configuration in RPW.

In RPW it is possible to add a Plug-In to a Plug-In as long as they are not contradictory. Thus, this aligns with the intention of the configuration package as well. However, depending on the RUP version, the method is constructed of method components to different extent. Rational Software’s final intention with RPW is to divide RUP into a core with surrounding Plug-Ins. In order to construct a situational systems engineering method a selection of Plug-Ins will be added to this core. Consequently, when reducing the base method to a minimum we also reduce the need to classify activities as ‘Skip’ in the base method. Thus, the Plug-In concept here is more in line with the method component as discussed in Röstlinger and Goldkuhl (1996).

The configuration template concept does not have a clear-cut counterpart. We can treat a published version of the base-method, i.e., a combination of Plug-Ins, as a template. Thus, in that sense they are equivalent. However, the intention with configuration templates is to fine-tune them before they are applied to projects. A combination of Plug-Ins can be used to generate a situational version of RUP runtime through RUP Builder. RUP Builder is an additional tool where the method engineer, through a simple point-and-click interface, can select Plug-Ins and add them to the published systems engineering method. If the Plug-Ins are reduced in size, it is possible to build very exact versions of the systems engineering method and hence enable a precise fit to an individual project. However, yet again this is a choice between precision and cost. The reasons for introducing configuration templates in MMC were to reuse the experience of relevant combinations and the problem of resolving conflicts between configuration packages. The latter problem is something the method engineer has to solve despite the availability of RUP Builder. Furthermore, in our meta-development project we concluded that a configuration template is the map of a desired configuration for a recurrent set of characteristics. Whether or not it is realized through RUP Builder is an implementation issue and highly dependent on the choice of base-method (indirectly the ability of the supplier’s CAME tool).

7.2.4 Experiences Gleaned

The concepts used in RPW as well as MMC have some common ground. Of course, differences can be found and the most fundamental one is Rational Software’s forthcoming decomposition of RUP into method components. Nevertheless, it is not clear either how to analyze the content of these Plug-Ins or how to retrieve a relevant selection back in order to build a configuration. The proposed use of characteristics in MMC could improve both the demarcation of Plug-Ins as well as the retrieval of relevant combinations using the characteristic as a set of questions for the method engineer to ask.

This workshop illustrates the importance of inter-subjective competence. Considering each actor in the workshop team the needed competence existed, however during this short workshop the team had problems making relevant
Episodes of Use

knowledge inter-subjective to a sufficient extent. Here, we have to distinguish between knowledge that is inter-subjective and knowledge that is subjective. This is fairly obvious when relating to our discussions about method as knowledge in 3.2. This affected the team and their ability to make efficient decisions about the configuration. Thus, we can conclude that it is important to make both knowledge of the tool and the meta-method inter-subjective when using workshops as the collaboration technique.

Much of the workshop team’s discussions centered on method integration versus method association. This is an area weakly supported in MMC, since our meta-method focuses on analyzing to what extent the base method’s ability is sufficient to support a specific characteristic of a development situation. Method-integration or method-association is an important question, however we believe decision should be based on the result of an MMC analysis. Then, we would have a more precise picture of the existing method content’s value and the enhancements needed. Subsequently, the main shortcoming of the process illustrated in Figure 39 is the combined analysis of content and technical issues. This combination limited the team’s possibility to get an overview of the configuration work.

7.3 The Personnel System

This episode is from Örebro University and the Department ESI. A project team of four people was assigned to develop a web-based personnel system. This was a small project of about 350 hours during three calendar months. The author acted as both project manager and method engineer in this team, a team we refer to as the ESI team. In addition, up to five users participated during different phases of this project.

In-house development is not considered as normal day-to-day business at ESI, thus the method engineer had no existing projects to categorize into development situations. Therefore, the personnel project as such was investigated and a situational meta-method was drafted. With regard to our definition of the development situation concept (see Definition 4.3 in Section 4.2.2) this is not a problem, however practically such a development situation will be agile. The elaboration of this specific configuration was not done as a delimited event rather it was drafted continuously during the project. Each part of the used configuration was linked to the characteristics that the method engineer considered to be the trigger. Consequently, the configuration work performed was more in line with Röstlinger and Goldkuhl’s (1996) definition of method configuration than the definition used previously in this thesis. However, the project is still interesting since the configuration work made is another way to approach the upstart and the elaboration of characteristics, configuration packages and templates. The essential ideas in MMC were used and the reusable configurations were based on an analysis of the base method’s rationality, hence the ideas discussed in Section 4.2.4. Furthermore, these configuration packages could be used as the foundation for creating a set of reusable configurations.
The following is a short summary of the conditions when the project started: An older system was used for storing information about personnel. During an earlier project when the department’s web site was re-designed\textsuperscript{48} a discussion started about the benefits of a database-driven web-solution for personnel information. The web project found unused potential through coordination of this information, but it was beyond the scope of that project. Furthermore, for technical reasons it was not a suitable solution to integrate the old system with the web site. Thus the ESI-team was assigned to this task.

The ESI team members had different experiences from web development and more specifically building component-based solutions combining Active Server Pages (ASP) and COM+. The technical platform for this project was given since the existing web site was built on ASP-technology. Hence, a complete environment existed for deployment as well as development. The latter was used during courses at the university. Of course the reason for the environment’s existence might be quite unusual, however not the situation with an existing development environment. Nonetheless, it was favorable to use this environment during this project.

Below, we present a selection of configuration packages resulting from the method configuration work. This is not a complete set with regard to those that were actually used during the efforts to develop the personnel system. Neither is it a complete set if we consider different choices in the characteristics’ dimensions. The intention behind this selection is to cover the different configuration situations found in Table 6, Chapter 4, ranging from ‘Skip’, ‘Perform as is’ to ‘Emphasize as is’.

7.3.1 Known Business Objectives

The base method for this project, RUP, prescribes that business modeling should be performed during the upstart of a project. The purpose of business modeling is, according to Kruchten (1999), to ‘understand the structure and the dynamics of the organization’, thus to generate a common understanding of the organization and its requirements. The method engineer discerned a characteristic concerning the present knowledge about business objectives. Since preparations existed for this project the ESI team considered the business objectives as known. Therefore, the method engineer derived two preliminary values for this characteristic.

\[
\text{<characteristic name = "Knowledge About Business Objectives?"} \]
\[
\text{<value valueid = "Known" cpid = "Unknown Business Objectives"/>}\]
\[
\text{<value valueid = "Unknown" cpid = "Known Business Objectives"/>}\]
\[
\text{<value valueid = "N/A" cpid = "N/A"/>}\]
\[
\text{</characteristic>}
\]

In the ESI team’s situation it was sufficient to characterize the situation as an unknown and a known situation. Of course, it can be discussed when we should

\textsuperscript{48} The author was responsible for the technical implementation of the re-design.
Episodes of Use

consider either of the two situations to exist. However, such problems occur irrespective of the derived values and afterwards one can always argue that the business objectives could have been known even better.

The project needed a configuration package for the second value above; hence we examine the configuration more closely. The configuration work was quite straightforward. The method engineer demarcated the affected activities to the ‘Business Modeling’ discipline in RUP. If we return to the purpose of this discipline it is about understanding the organization and in a development situation where the business objectives are known, the importance of these prescribed actions could be considered as low. Subsequently, according to the heuristics in Section 4.2.4.1 we should in such a case consider a ‘Skip’. For this configuration package the method engineer chose to classify the entire discipline as ‘Skip’. This classification also brought about an inherited classification of all activities and steps within that discipline. Thus, the chosen level of granularity was rough and one should be aware of the fact that, at least in RUP, business modeling can be performed on different levels, dealing with differences in complexity. If we would have managed such cases then it might have been better to choose the activity level as the level of granularity. Then it would have been possible to classify each activity individually and the discipline could have been classified as ‘Performed reduced’ assuming that the method engineer’s intention was a light version of the ‘Business Modeling’ discipline. Furthermore, such an approach also calls for a revision of the characteristic’s values making finer distinctions between them.

7.3.2 Modeling Web Application

The ESI team had to capture the information system requirements. When examining the results from the initial discussion about this intended system the ESI team did not consider these requirements as extraordinary. However, the final artifact had one distinct characteristic; it was a web application, which is a specialization of the more generic Internet Based Software Artifact (Karlsson, 2001a). A web application is one type of thin client, where the user interface is based on web pages.

It is important to design these applications with a natural navigation structure. The ESI team members had experience from earlier web application projects and knew that it is not easy to balance the content on each web page with the amount of pages needed to accomplish a defined task in the system. Thus, one of the ESI team’s sub-goals, during this project, was a good balance between the content on each web page and the need to navigate. The team needed some procedures to map use cases to web pages. Furthermore, these web pages had to be related to each other, illustrating possible navigation paths for the users and making these issues focal. The method engineer’s comprehension was that the base method did not support these requirements through the activity ‘Model the User Interface’. It can be argued that storyboards, which are used in this RUP activity, can describe a flow of events and hence fulfill the ESI team’s sub-goal. Furthermore, the proposed activity in RUP has an optional step for describing navigation paths between boundary objects and actors. Nevertheless, the ESI team considered this activity difficult to
use in this specific situation. Therefore, the method engineer chose an in-house developed alternative, a navigation map. The navigation map makes an improved visual overview possible during modeling, which in addition is a useful tool when working together with the user group during workshops.

In order to capture the information system requirements the characteristic below was defined. During this personnel project the method engineer only defined one actual value, web applications. Therefore, for further use it will need to be elaborated with complementing values as well as related configuration packages.

```xml
<characteristic name = "Modeling of Application Type?">
  <value valueid = "Web Application" cpid = "Modeling if Web Application"></value>
  <value valueid = "N/A" cpid = "N/A"></value>
</characteristic>
```

The demarcation of the configuration package Web Application was set to the ‘Requirements’ discipline in RUP. This might be a somewhat broad demarcation if we focus on this single value. However, if we examine the characteristic, it affects activities within this discipline.

<table>
<thead>
<tr>
<th>Discipline</th>
<th>Workflow Detail</th>
<th>Activity</th>
<th>Modeling of Web Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Refine the System Definition</td>
<td>Perform extended</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Detail a Use Case</td>
<td>Perform as is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Detail the Software Requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Model the User Interface</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Model Navigation Map]</td>
<td>New</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prototype the User Interface</td>
<td>Perform as is</td>
<td></td>
</tr>
</tbody>
</table>

Table 10. Configuration of Modeling: Web Application

Table 10, illustrates a selection of affected activities in the ‘Requirements’ discipline. The selection constitutes the workflow detail ‘Refine the System Definition’ and its activities are found in the third column from the left. In the right-hand column we can find the classification of activities in the configuration package Web Application. On the workflow detail level of granularity, the method engineer classified ‘Refine the System Definition’ as ‘Perform extended’, since the workflow detail contains an exchanged activity. The method engineer does not argue that modeling this type of user interface involves a higher degree of complexity than usual, rather it involves another type. Focusing on the actual activity ‘Model the User Interface’ its purpose is ‘To build a model of the user interface that supports the reasoning about, and the enhancement of, its usability’ (RUP 2001
Episodes of Use
documentation). Compared to the ESI team’s sub-goal it is not the purpose as such that is irrelevant to fulfill. Rather it is the operationalized steps and the proposed deliverables that the team found insufficient. Consequently, the team needed an enhancement of the base method. At the same time, the method engineer reduced the base method. One solution would be to classify the prescribed action as an ‘Exchange’. The chosen solution for illustrating this configuration was to classify ‘Model the User Interface’ as ‘Skip’ and add the new activity. The implication is that other configuration packages, that require this specific package, will treat this activity as part of the input to configure. This configuration issue is similar to the one discussed in Section 7.2. In both cases, the intention is to exchange sections in the base method, however to a different extent. Returning to the meta-method description this is the reason for the introduction of the require-statement (see Section 4.2.4.1).

When extending the base method through a configuration package the method engineer used support from other method engineering approaches because MMC does not have any support for analyzing whether or not concepts will be in conflict with each other. However, before the conceptual analysis the method engineer analyzed the rationality behind tentative extensions. Hence, the set of possible extensions was primarily based on whether or not they would contribute to the intended purpose.

7.3.3 Quality Assurance

The project environment put limitations on how the project could be conducted. In the existing development environment the ESI team did not have any available tools for automated testing and resources for new tools was not allocated to the project. Several of the recommended procedures in the base method are about the preparation or execution of tests with such tools. Therefore, the method engineer distinguished between two different situations when conducting tests. The first one using automated tools, and the second one performing tests manually. The following characteristic was defined and the ESI team needed a configuration that aligned with the second value:

```xml
<characteristic name = "Type of Quality Validation?">
  <value valueid = "Automated" cpid = "Automated Testing"></value>
  <value valueid = "Manual" cpid = "Manual Testing"></value>
  <value valueid = "N/A" cpid = "N/A"></value>
</characteristic>
```

The demarcation of these two configuration packages is the ‘Test’ discipline in RUP. This discipline contains several workflow details with single activities; consequently several workflow details were affected by a configuration for manual

---

49 Of course, to some extent the environment can be created. Nevertheless, there is always an outer limit for what is possible.
testing. The first workflow detail ‘Plan Test’ in Table 11 has a purpose to fulfill, regardless if we use an automated or manually executed testing. Thus, both activities in the workflow detail are classified ‘Perform as is’.

The activities ‘Implement Test’ to ‘Implement Test Components and Subsystems’ focus on, e.g., the generation of reusable test scripts, test components and defining interfaces for the automated tools. Since the ESI team did not intend to use automated tools these activities provided very limited values to the project. Hence, these activities were classified ‘Skip’ and the related workflow detail was also classified ‘Skip’. Moving on further in the list of workflow details ‘Execute Tests in Integration Test Stage’ and ‘Execute Tests in System Test Stage’ both contain an instance of the activity ‘Execute Test’. The difference is if the test is performed on a developed component as such and how it collaborates with its context or if it is a test of the system’s complete functionality. The ESI team found both these tests relevant to perform. The first test was important since the ESI team was working on a component-architecture and the second test had an obvious need as well; the team had to know whether or not the system met the requirements. However, all descriptions about automated testing were ignored. Subsequently, these activities are classified ‘Perform reduced’. The last workflow detail ‘Evaluate Test’ is essential to perform in order to assess the status of the system. Change requests are issued during this activity and it is natural to classify it ‘Perform as is’.

Table 11. Configuration of Quality Validation: Manual Testing

<table>
<thead>
<tr>
<th>Discipline</th>
<th>Workflow Detail</th>
<th>Activity</th>
<th>Manual Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>Plan Test</td>
<td>Plan Test</td>
<td>Perform as is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prioritize Test Cases</td>
<td>Perform as is</td>
</tr>
<tr>
<td>Design Test</td>
<td>Design Test</td>
<td>Perform as is</td>
<td></td>
</tr>
<tr>
<td>Implement Test</td>
<td>Implement Test</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Design Test Packages and Classes</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implement Test Components and Subsystems</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td>Execute Tests in Integration Test Stage</td>
<td>Execute Test</td>
<td>Performed reduced</td>
<td></td>
</tr>
<tr>
<td>Execute Test in System Test Stage</td>
<td>Execute Test</td>
<td>Performed reduced</td>
<td></td>
</tr>
<tr>
<td>Evaluate Test</td>
<td>Evaluate Test</td>
<td>Perform as is</td>
<td></td>
</tr>
</tbody>
</table>

Regarding the activity ‘Implement Test’ the test scripts are proposed to be automated when feasible. Thus, there is potential to create manual test scripts or checklists. However, we regard such scripts more as procedures, and hence they the test scripts are developed during the activity ‘Design Test’ and more specifically in the step ‘Identify and Structure Test Procedures’.

167
Episodes of Use

7.3.4 Existing Environment

The ESI team had the fortune to have both an established deployment as well as a development environment. One part, the ‘Environment’ discipline, in the base method concerns the set up of a development environment. Subsequently, the method engineer found a need to tailor the base method to the project situation. The characteristic Prepare Environment, defined below, was demarcated to activities in the ‘Environment’ discipline. However, the workflow detail ‘Prepare Guidelines for an Iteration’ was excluded from these configuration packages, which we will argue for later on. The method engineer identified two different situations, which were used as values. New Environment represents situations in which we need to prepare the project environment; thus the opposite situation to the project in question. The second value Existing Environment is the configuration used during the personnel project.

<characteristic name = "Need to Prepare Environment?”>
  <value valueid = "New Environment Is Used" cpid = "Prepare New Environment"></value>
  <value valueid = "Existing Is Used" cpid = "Prepare Existing Environment"></value>
  <value valueid = "N/A" cpid = "N/A"></value>
</characteristic>

The demarcated activities can be divided into three categories. The first one concerns the choice and set up of tools to use. In the second category we find the configuration of the systems engineering method, and in the last category we have the elaboration of templates and guidelines for the project. Hence, the method engineer has been somewhat rough when identifying this characteristic, when the initial focus was on the technical environment. In the future it might not be possible to group all these three sections into one characteristic, or we might have to define additional values in order to mirror the nuances where, for example, the technical environment exists but not the documentation templates.

Since the development environment existed the method engineer did not find it necessary to evaluate and select tools for the project. The existing technical environment fulfilled the ESI team’s need and in the project manager’s opinion an additional assessment would have had marginal effects\(^5\). Furthermore, it was also considered unnecessary to assess the current organization. In Table 12 the method

\(^5\)This argument is somewhat contradictory to the knowledge one has at the initial or the pre-elaboration stage of a project. For example, if we consider the development environment, we assume sufficient knowledge of how and what tools to use during implementation. Hence, we are not without assumptions when we start such a project. In contrast, it is not often a project can start from a blank sheet. The workflow detail ‘Prepare environment for Project’ also directs attention to an assessment of the organization, and the competence of the project team. When skipping the workflow detail we did not assess the project group’s competence in the implementation techniques used and the effort needed to train project members. Subsequently, we increased the project risk, which unfortunately became a highly tangible problem during some stages of the project and the implementation issues could not be divided between the project members in an efficient way.
engineer has classified all activities in the workflow detail ‘Prepare Environment for Project’ as ‘Skip’ including the ‘Develop Development Case’ activity. The latter classification could seem a bit contradictory to our developed approach, however the method engineer argued that a formal Development Case was not necessary. Furthermore, MMC would be used instead of this activity; subsequently MMC could have been added instead of the suppressed activity. The reason for not doing so was the awareness of the informal use of MMC; the method engineer applied its principles continuously during the project to capture an initial set of Configuration Packages. One way of arguing is that this illustrates a situational use of MMC that fit the situation better than the complete meta-method. Of course, one can argue that it should be clear from the configuration package that MMC should be used in every project, at least for the selection of configuration templates.

Table 12. Configuration of Prepare Environment: Existing Environment

<table>
<thead>
<tr>
<th>Discipline</th>
<th>Workflow Detail</th>
<th>Activity</th>
<th>Prepare Existing Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>Prepare Environment for Project</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asses Current Organization</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Develop Development Case</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Develop Project-Specific Templates</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select and Acquire Tools</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prepare Environments for an Iteration</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Develop Development Case</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Develop Project-Specific Templates</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Launch Development Case</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set Up Tools</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Verify Tool Configuration and Installation</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td>Support Environment</td>
<td>During an Iteration</td>
<td>Skip</td>
<td></td>
</tr>
<tr>
<td>Support Development</td>
<td></td>
<td>Skip</td>
<td></td>
</tr>
</tbody>
</table>

Since the activities in the workflow detail ‘Prepare Environment for an Iteration’ are related to the previously discussed workflow detail the method engineer used much of the same argumentation for this classification. The risk with this approach is that we do not deal with the preparation of a specific iteration, which could have more specific needs than the project as a whole. These needs
Episodes of Use

could also change between different iterations. However, since this was a short project this risk was considered low. Thus, one should be aware that this configuration package has many specific considerations that might not be present in a larger abstraction. Yet again, it is probable that this configuration package and the characteristic will change if and when reused.

The small project group reduced the need for development of guidelines. Furthermore, the ESI team had the policy of relating the configuration of guidelines in the workflow detail ‘Prepare Guidelines for an Iteration’ to characteristics, which affected activities using these guidelines. Hence, many of these activities have been discussed above.

Finally, there is one more workflow detail to consider in this configuration package. The purpose of ‘Support Environment During an Iteration’ is to support the developers during the development efforts. The method engineer could not argue that this was not necessary, however when investigating the current situation for this project, formal support resources did not exist outside the project team. Consequently, classifying this activity as anything other than ‘Skip’ would have no actual effect. The configuration would not have mirrored the project or its surrounding environment; rather it would have been wishful thinking. Maybe this is an unusual circumstance and might not hold when this configuration package is based on a development situation consisting of more than one project.

7.3.5 Thin Client Deployment

As mentioned in Section 7.3, this development project started with certain limitations, or put in another way it was to fit into other existing parts of the organization’s information systems. One part that limited possible implementation was the existing web site at ESI, which could be considered a legacy system in this case. The user interface of the new information system was to comply with the existing web site. Subsequently, it affected the team’s freedom to design, and more fundamentally it restricted (or simplified) our decision about deployment. Web pages are thin clients and can be deployed through one single server; hence it was possible to deploy the software using one copy. Therefore, the Deployment Strategy characteristic was added to the list of characteristics. The method engineer distinguished between the values Thin Client Deployment and Local Client Deployment, the latter representing deployment with the installation of client side software.

```xml
<characteristic name = "Type of Deployment Strategy?">
  <value valueid = "Thin Client Deployment" cpid = "Thin Client Deployment"></value>
  <value valueid = "Local Client Deployment" cpid = "Local Client Deployment"></value>
  <value valueid = "N/A" cpid = "N/A"></value>
</characteristic>
```

Of course, it is possible to find other deployment strategies. The deployment strategy for web applications might not be this straightforward and can have many variants (Karlsson et al, 2001a)
Table 13. Configuration of Deployment Strategy: Thin Client Deployment

<table>
<thead>
<tr>
<th>Discipline</th>
<th>Workflow Detail</th>
<th>Activity</th>
<th>Prepare Environment: Existing Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment</td>
<td>Plan Deployment</td>
<td>Develop Deployment Plan</td>
<td>Perform reduced</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define Bill of Materials</td>
<td>Perform as is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td>Develop Support Material</td>
<td></td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td>Manage Acceptance Test (At Development Site)</td>
<td>Develop Training Materials</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Develop Support Materials</td>
<td>Perform as is</td>
</tr>
<tr>
<td></td>
<td>Produce Deployment Unit</td>
<td>Manage Acceptance Test</td>
<td>Perform as is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write Release Notes</td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Develop Installation Artifacts</td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td>Manage Acceptance Test (At Installation Site)</td>
<td>Manage Acceptance Test</td>
<td>Perform as is</td>
</tr>
<tr>
<td></td>
<td>Package Product</td>
<td>Release to Manufacturing</td>
<td>Perform as is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Verify Manufactured Product</td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td>Provide Access to Download Site</td>
<td>Create Product Artwork</td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Provide Access to Download Site</td>
<td>Skip</td>
</tr>
</tbody>
</table>

The configuration of the Thin Client Deployment is illustrated in Table 13 above. Irrespective of the choice of deployment strategy it is important to plan it, which is the purpose of Develop Deployment Plan. During this activity the how and when of the deployment of the software is considered. In this case, the how-aspect was roughly set, but the ESI team still had to plan when. Therefore, this activity was classified ‘Perform as is’. However, the second activity in the workflow detail ‘Plan Deployment’ was skipped. Its purpose is to create a list of documentation needed for installation, which in this specific project was unnecessary since the ESI team was supposed to do the installation. Continuing with the material issues, the development of training and support materials were also classified as ‘Skip’. Since this was a web
application, the team had very limited opportunities to train the users; consequently such material as well as support material would be of no real use\textsuperscript{3}.

The base method was configured in such a way that acceptance tests were supposed to be performed both in the development environment as well as in the installation environment, in order to settle the status of the project. The ESI team had no intention to produce installation artifacts during this project. The reason was the characteristic of the final artifact, which was supposed to have only one installation point and the developers were responsible for that specific installation. Subsequently, the installation could be performed as easily using manual procedures. Registering the COM+ components used was considered as a small amount of work to do and, regardless of that part, it was not possible to create a completely automated installation process. Finally, the method engineer classified the workflow detail ‘Package Product’ and ‘Provide Access to Download Site’ as ‘Skip’. None of these workflow details were essential for deployment of a thin client, since there was no software package to deliver for installation on the user’s computer.

7.3.6 Experiences Gleaned

The application of MMC proved to be useful during the personnel project at ESI. Above we show a selection of five different configuration packages containing the situational systems engineering method used in that project. We cannot claim that our characteristics or configuration packages will be stable over time, since they are based on one single project. However, they are a starting point for further configuration work and can be reused during that work. Furthermore, the analytical tool and the classification system have proven useful during this project. The method engineer has used different levels of granularity when classifying prescribed actions, hence used the potential to adjust the balance between precision and cost. For example, the classification of the ‘Business Modeling’ discipline in RUP was straightforward.

The ESI environment does not offer a large project portfolio and the configuration work started from a blank sheet. Section 3 in MMC, the ‘Selection of Configuration Templates’, has not been tested since only one configuration template was created. As a natural consequence, the method engineer could combine the configuration packages without any conflicts; subsequently this part of MMC has not been exhaustively tested.

The method engineer used the ESI team’s experiences in tailoring the base method. Their earlier experiences gleaned from modeling web applications resulted in an enhancement of the base method. Consequently, it was possible to incorporate situational aspects based on experiences from previous projects, which is an important part of the MMC framework illustrated in Figure 25 (see Section 4.3). However, this enhancement, yet again, illustrates the need for harmonization between the base method and the added prescribed actions and their concepts. As

\textsuperscript{3} If we focus on our specific project, we had invested great effort in the requirements and modeling the user interface. Thus, we tried to make the user interface self-instructive.
with the large workshop, in Section 7.2, we can conclude that MMC offers weak support in this area. It enables the method engineer to argue for an enhancement but MMC does not contain prescribed actions for the analysis of concepts.

During this project, a potential new class, ‘Exchange’, was discussed. However, it was not incorporated in the classification system. This situation is possible to cover with a combination of ‘Skip’ and ‘Extend’ and we find a small classification system valuable. Using a wide range of classes increases the complexity of the classification system and does not make it easy-to-use. The introduction of an additional class still results in the same amount of rows in a configuration package; we have to declare the suppressed prescribed actions as exchanged and introduce the new one. The advantage of this might be found on the semantic level and could be important on the layer of granularity above the analyzed prescribed actions. For example, if an activity in RUP is classified as ‘Skip’ and a new activity is introduced, there is no natural classification for the workflow detail in the current classification system. This problem is illustrated in Table 10. However, we need to return to the intention for this classification. We did not find the base method sufficient, hence we extended it and that the need of enhancement should guide the classification.

7.4 Summary
The objective of this chapter was to present a selection of episodes as arguments for design decisions made during the development of MMC. Furthermore, this selection is an evaluation of the meta-method’s usefulness in configuration work. Hence, this chapter has acted as a validation of the development work.

The three episodes have illustrated different aspects of the development process. The first episode is a snapshot of the design decisions and tests made at an early phase of this meta-project. We described the process for a specific workshop, however this process reflects the principle procedures used during other workshops. The concepts, the analytical tool and the classification system in MMC proved useful during this workshop. As a natural consequence of an early test, the test resulted in design and development implications. These results have been discussed and are apparent in both Chapter 4 and the detailed description in Chapter 6. This workshop also resulted in practical experience of using the drafted concepts. Eliciting characteristics was not easy, and the meta-method project group used an analytical framework in order to compare different types of information systems. Finally, we let the example of a tentative configuration template illustrate the fact that the all combinations of configuration packages are not relevant for an organization.

The second workshop centered on the integration of the meta-method and the tool for configuring RUP, a tool delivered by Rational Software. Our intention was to evaluate how MMC could complement RPW, and if the benefits of a standard systems engineering method could be used. The analysis focused on the process, in which MMC could become a complement, and the common ground for concepts. The actual use of MMC during this workshop was limited, however we found that
the meta-method could become a valuable complement in this work. RPW is a technical tool and it is not clear either how to analyze the possible content of RPW Plug-Ins or what to base a relevant selection of Plug-Ins on when building a situational systems engineering method. In addition, the concepts used in RPW as well as MMC share some common ground but the Plug-In concept is more in line with Röstlinger and Goldkuhl’s (1996) method component concept than with the configuration package. The configuration template concept does not have a clear-cut counterpart in RPW.

The third and last episode was also the most comprehensive. It was the only system engineering project in which MMC has been used so far. The application of the analytical tool and the classification system proved useful during this project. Experience gleaned from other projects was incorporated into the configuration packages. This was an important part of the original idea in Chapter 1 and the MMC framework in Chapter 4. As a result of this project, the ‘Exchange’-classification, a possible design implication on MMC, was discussed. This classification was described as a combined ‘Skip’ and ‘Extend’ classification. However, this classification has not been incorporated into MMC and its classification system, since we believe that the existing classes can capture this situation as well.

The method engineer used an analytical framework based on actability to characterize the final artifact. Hence, we reused experiences from the first workshop about difficulties in capturing characteristics without using a screen. However, a reflection could be made on this framework. Its main focus is on the final artifact, its requirements and surrounding environment. Thus, using this framework as the only screen, we risk missing aspects of the project team and the project environment. Therefore, a more comprehensive screen or a complement would be valuable.

The configuration work in the first and third episodes was performed without proper tool support. During the second episode, we examined the potential to use the existing tool RPW when working with RUP. The principles of MMC are possible to map to RPW, however the mapping has to be done manually. Consequently, the method engineer lacks proper tool support when working with MMC and this is an obstacle. Thus, an integrated tool would be desirable in such a case. We do not know how the situation is for other possible base methods, since that is beyond the scope of this study.
8 CONCLUSIONS AND FURTHER RESEARCH

The chapter is devoted to summing up the theoretical discussions and the empirical grounding made in this thesis, thus providing some concluding remarks. In addition, we give some comments on further research based on the experience gleaned in this thesis. Section 8.1 is a general discussion about achieving the thesis’ aim. This achievement is made concrete in Sections 8.2 and 8.3 where we recapture the main contributions of Method for Method Configuration (MMC). The limitations of the proposed meta-method and its perspective are discussed in Section 8.4. In Section 8.5 we discuss future research and improvement areas of MMC. Finally, in Section 8.6 we return one last time to our example about oranges.

8.1 Achieving the Aim

In this thesis, we address the problem with rigorous standard systems engineering methods and how to turn them into situational systems engineering methods. We have experienced that the current popularity of such methods, where Rational Unified Process (RUP) is one example, has both its upsides as well as it downsides. On the positive side, we find the completeness with regard to the phases in the development process, these methods have been sorted out through use in several organizations, training courses are widely available and advanced tool support can often be found. However, the existence of these methods causes new problems or variants of old ones. These methods are impressive both in the amount of prescribed actions to perform and the artifacts produced during a project and we believe that such methods should not be applied ‘as is’. Consequently, a situational approach is necessary, which was the foundation for the formulation of the thesis’ aim:

*The aim of this thesis is to present and elaborate on the design of a meta-method emphasizing reusable configurations based on standard systems engineering methods’ rationality and context.*
Conclusions and Further Research

We elaborated on this aim in order to become more precise about the meaning of ‘present’ and how the elaboration could be validated. In Chapter 1 we referred to ‘present’ as the principles and structure of the meta-method. Through the discussion of methods and method engineering in Chapter 3 we became even more precise stating that we needed to present basic values, concepts, prescribed actions, notation and a framework. Furthermore, elaboration of design has to be validated as valuable knowledge. The elaborated knowledge has been validated using a combination of the three grounding processes, internal, empirical, and external theoretical, as discussed in Goldkuhl (1999).

We consider both the presentation as well as the elaboration aspect of the aim accomplished. In Chapters 4 and 6, we present the meta-method’s concepts, prescribed actions and framework. Chapter 4 was devoted to a theoretical discussion of MMC and included the conceptual framework, XML-based notation, an analytical tool for addressing a systems engineering method’s rationality and finally a framework for prescribed actions. Subsequently, we cover the main constituents of a method discussed in Chapter 3 as well as in our initial discussion in Section 1.1.2. We used RUP as the unit of analysis with regard to rigorous systems engineering methods, and in Chapter 6 we presented a detailed description using RUP as a shell. Using RUP terminology, MMC is described as a discipline having workflow details, activities, and steps. Chapter 5 contains a discussion of how the theoretical version of MMC was mapped into the RUP specific version.

MMC has been validated through an iterative research pattern described in Chapter 2. However, below, we address each validation process in sequence. An internal grounding has been made and is an important part of our empirical contribution with regard to our research strategy. With our interaction research strategy, we deliberately conducted the method engineering efforts in both an academic research environment as well as in an empirical one. In the latter, the author and other researchers from the VITS research group worked together with qualified practitioners, mainly from Volvo IT. Several workshops were devoted to developing the MMC concepts and their completeness and consistency. The result can be found in Sections 4.2.1 to 4.2.5. Second, the meta-method was empirically grounded in these workshops and a separate system engineering project at the Department of Informatics (ESI) at Örebro University. Furthermore, MMC was continuously tested through several stages of the development process. A selection of these tests or episodes of use are presented in Chapter 7. That chapter is used as one part of the argumentation behind our design decisions. The third and final grounding process, theoretical grounding, is presented both in Chapter 3 as well as in Chapter 4. The former is devoted to alternative perspectives on systems engineering methods and method engineering while the latter is a comparison with MMC, intertwined with the presentation of our conceptual model and framework.

8.2 Prescribed Actions for a Focal Area
The theoretical contribution of MMC is mainly found in our conceptual model and how it has been turned into prescribed actions with an analytical tool and
classification system. The complete conceptual model is found at the end of Chapter 4 (see Figure 26). We have introduced the concepts of configuration package and configuration template, which aim to ease the burden of configuring standard systems engineering methods given certain project characteristics. A configuration package has been described as a pre-made method configuration designed to fit a specific development characteristic. Furthermore, configuration packages can be combined into configuration templates representing recurring development patterns in a particular development organization. Both configuration packages and configuration templates are classifications of the prescribed actions in the standard systems engineering method, referred to as the base method.

![Importance Complexity Matrix](image)

**Figure 40.** Suggested ways for performing different prescribed actions of the base method derived from the importance and complexity of the areas with which the prescribed actions are concerned.

The base method’s rationality is used as the focal area for the classification of prescribed action. Prescribed actions are performed for reasons, reasons founded in the method engineer’s perspective. Thus, each prescribed action has a purpose, which is used, as the focal area for method configuration in MMC. Depending on the importance and complexity to achieve that purpose with regard to the projects overall goal, we use the classification matrix in Table 14, below. However, the analysis is performed using the analytical tool shown in Figure 40, above. It
Conclusions and Further Research

embodies heuristics for classifying prescribed actions according to Table 14. On the horizontal axis, we find importance of fulfilling the purpose of a prescribed action, ranging from low to high. Vertically, we have the complexity of the area of concern that the particular prescribed action addresses, and the complexity ranges from low to high.

Table 14. Performance of prescribed actions in the base method deduced from the categories of attention and enhancement.

<table>
<thead>
<tr>
<th>Attention given to prescribed action</th>
<th>Reduced</th>
<th>As is</th>
<th>Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Skip</td>
<td>Skip</td>
<td>Skip</td>
</tr>
<tr>
<td>Normal</td>
<td>Perform reduced</td>
<td>Perform as is</td>
<td>Perform extended</td>
</tr>
<tr>
<td>A lot</td>
<td>Emphasize reduced</td>
<td>Emphasize as is</td>
<td>Emphasize extended</td>
</tr>
</tbody>
</table>

Table 14 is based on the two dimensions attention and enhancement. Attention corresponds to the importance, in Figure 40, of achieving the prescribed action’s purpose. Depending on our analysis, we could give none, normal, or a lot attention to a prescribed action. For example, if the purpose of a prescribed action is unimportant we would not give it any attention. On the horizontal axis, we map complexity to the need for enhancing the prescribed action in order to fulfill its purpose. We work with three options, to reduce the content of a prescribed action, to perform it as described in the original method, or to extend it with content from additional systems engineering methods. For example, a complex task might not be solved through the prescribed action, as it is found in the base method, and hence additional prescribed actions on a lower layer of granularity have to be added to the prescribed action in question. However, when discussing reduction we have to exclude prescribed actions on the lowest layer of granularity from this line of reasoning. They are atomically prescribed action and thus cannot be reduced or extended. Suppressed or added prescribed actions are implemented using a lower layer of granularity than the prescribed action in question and there is no lower layer in this case. These restrictions are illustrated in Figure 40 and Table 14 by the shaded area. All in all, we end up with a classification system of at most seven possible categories.

Summing up this brief recapitulation, we have contributed with one possible way to focus the base method’s rationality and subsequently a possibility to elicit reusable method configurations. The benefits of the MMC concepts are that method configuration can be performed efficiently since pre-made configurations are possible to use over and over again. Furthermore, one interesting aspect of this classification schema is the attention dimension and the potential to support method adaptation in addition to method configuration. Method adaptation is the final
tailoring performed by the method user during method utilization, and an attention
classification is a recommendation for the amount of effort to be put into a
prescribed action.

The conceptual model has affected the design of the MMC framework, which is
found in Section 4.3. The framework is divided into three sections to support the
reusability aspect of MMC, which is that each section should not be performed with
the same frequency. MMC section 1 is about constructing Configuration Packages
for specific characteristics in the organization’s development situations and should
be the least frequent section. In MMC section 2, Configuration Packages are
combined into Configuration Templates which are stored in a repository. The most
frequent section is MMC section 3 in which a Configuration Template is selected
and fine-tuned into a situational systems engineering method for a specific project.
Consequently, there is no need to perform a complete method assembly for each
new project. Furthermore, it seems that experiences can be gathered and reused
more efficiently since they can be attributed to coherent sets of prescribed actions
common in the organization, rather than to context-free atomic actions.

8.3 Empirical Experiences
We do not claim that MMC has been exhaustively tested. Thus, our empirical
experiences of using MMC are limited. However, an exhaustively testing was
beyond the scope of this thesis and the possible as well as the chosen selection of
tests were justified in Section 2.2.3.3. The testing conducted has mainly been used
as a first evaluation of the design made. We have experiences gleaned from two
empirical environments with quite different characteristics. Volvo IT is a large
multinational company with a large portfolio of projects of different sizes, and has
an advanced implementation of RUP. At ESI, on the other hand, system engineering
projects are not frequent, and they do not use a specific standard systems
engineering method. However, during this specific project, the choice fell on RUP.
Considering these different characteristics, we have found MMC useful in both
organizations and we have moved MMC outside the organization where it was
developed. Thus, we do not find any reason why the MMC should be bound to a
specific organization. Nevertheless, we are aware that the tests performed within
each environment are not comparable with each other. They differ both in
complexity, realism and MMC version tested. Furthermore, the author has
participated actively in all tests, thus the externalized version of the meta-method
has not been tested by other method engineers alone.

None of the performed tests have been complete with regard to the content of
the MMC framework. Especially the prescribed actions in MMC section 3 have not
been tested during real configuration efforts. The main reason has been the supply of
configuration tasks in proportion to the time limits of this study. In order to test
the selection of configuration templates, we need a range of configuration templates,
which have not been possible to produce in this context. Such a test would involve a
large project portfolio. Furthermore, we have not tested the use of change request on
configuration package or configuration templates. Since we have begun building the
Conclusions and Further Research

initial set of both of these types of artifacts, no real change requests have been found. Of course, one could argue that a request for this initial set is a change request, but that is hairsplitting.

The first and second sections have been tested, which is presented in Chapter 7. Consequently, the heart of MMC, the analytical tool and the classification system, has been tested. The most complete test was performed on a system engineering project at ESI. It resulted in five different configuration packages and one configuration template. The latter was a combination of the former, which has not explicitly been reproduced in this thesis. The main reason was that the combination did not differ from its parts.

This project showed that MMC contributed with a valuable focus during configuration work. MMC was an efficient way to decide on the prescribed actions. One ‘common-sense’ experience during these tests was the order in which we addressed the importance of achieving a purpose and the complexity in achieving it. We propose that the importance aspect is considered before the complexity aspect is discussed, since the latter is unimportant to consider if the purpose of a prescribed action is not found relevant to fulfill at all.

The classification of prescribed actions in a configuration package and a configuration template could be found on different layers of granularity. This has become valuable since we can find an efficient combination of cost and precision for each configuration. For example, classification of a complete discipline in RUP could be combined with the classification of a step within another discipline since this would be considered as coherent, and everything could be contained in a configuration package.

However, we should also be self-critical. In the ESI case study, we gave the characteristics of the final artifact too much attention, while the characteristics of the project team come to rest in the background. One possible explanation is vaguely prescribed actions about how to elicit characteristics. We have built much of the characteristics using the analytical framework based on actability presented in Ågerfalk et al. (2002), both when working at Volvo IT as well as at ESI. This framework is not an explicit part of MMC rather it was our choice of perspective on information systems during characterization.

We have only built one complete configuration template based on the final version of MMC and it was a straightforward task to combine the configuration packages. All the chosen configuration packages were designed for this specific configuration template. Thus, it was quite natural that we did not experience any problems with overlapping configuration packages.

All tests have been performed on one base method, RUP. This test set-up was justified through our choice of unit of analysis, discussed in Section 2.2.1. Thus, we do not know how applicable MMC is on other standard systems engineering methods. This depends on several factors but we find the following central:

- The chosen systems engineering method’s ability to cover different types of projects as an ideal type of method. Small methods will require several supplementary parts from other systems engineering methods, and that is not
within MMC’s primary focus (see Section 8.4). One solution could be found in constructing a new larger systems engineering method, which is then used as the base method in MMC.

- How explicitly the systems engineering method’s rationality is described. Reconstruction efforts might be needed if it is vaguely described, thus reducing MMC’s direct value.

8.4 Limitations of Method for Method Configuration

We gleaned experience from the empirical grounding of MMC and can address the limitations of the perspective used in MMC. In Section 7.2 we discuss an episode of use where MMC falls short for several reasons and where conceptual modeling could be a complement to configuration based on a systems engineering method’s rationality. Nevertheless, the method’s goals should be the base when performing the conceptual modeling, focusing on the roles each concept has for different goals.

In line with these experiences we can identify the following type of situations during method configuration:

1. Choose among the standard systems engineering method’s prescribed actions based on their goals.
2. Add prescribed actions from additional systems engineering methods when relevant options are missing in the standard systems engineering method.
3. Develop prescribed actions in those cases in which the method engineer’s purpose cannot be achieved through the standard systems engineering method or additional systems engineering methods (or parts thereof).

MMC addresses the first situation and provides guidelines for how to perform the choices based on the standard system method’s rationality (the procedures are discussed in Section 4.2.4.1 and Chapter 6). MMC can be used for choosing prescribe actions from additional systems engineering methods using their rationality as the base for decision-making. In a situation in which the proposed prescribed actions complement the overall goal through these actions’ rationality the configuration should be extended, at least from MMC’s point of view. However, MMC does not support the method engineer in the integration issues that are needed, mapping different concepts and notations to each other. The third situation has basically no support at all in MMC when the method engineer has concluded that there is no prescribed action either in the standard systems engineering method or the additional systems engineering methods that has been evaluated. In this case, new prescribed actions have to be developed, which is method engineering. However, this development should be focused on adding the necessary rationality to the standard systems engineering method.
Conclusions and Further Research

8.5 Future Research

During the work with this thesis a couple of issues have been put aside since they have been beyond the scope of the thesis. Summing up, we find four major tracks for further research, which are discussed below.

Test and Evaluation

The first and most apparent need is more elaborated tests for MMC. As discussed above, testing has not been the primary focus in our study and we are completely aware of the fact that MMC has not been completely sorted out. Thus, it would be a welcome contribution to elicit both the weaknesses as well as the strengths of MMC. More testing in different organizations is needed as well as on different base methods. Depending on the chosen base method, such an evaluation would also involve operationalization of MMC to the specific method.

An Improved Operationalization of The Characteristic Concept

From our delimited test we have found the need for a further development of the characteristic concept. We need an improved way to describe projects and make abstractions into development situations. In the present design of MMC, this is a weak point. A development situation needs to capture both the characteristics of the final product as well as the project team and its conditions. One option might be to use more widely used characteristics such as those found in van Slooten and Hodes (1996) and Harmsen (1997).

Tool Support

Another important future research topic is the tool support for the proposed meta-method and for method configuration based on systems engineering method’s rationality. Today, it is time-consuming to keep track of the prescribed actions and their impact on each other. A tool would improve the efficiency as well as the quality of the configurations. Reading Kaidnl et al. (2002) we find that the lack of tools is one reason why industrial projects do not adapt new methods. Considering the RUP case such a tool probably needs to be integrated with the Rational Process Workbench (RPW) and Rational Rose (RPW is a plug-in to Rose), which today is the tool for configuration of RUP. However, tool support requires an elaboration of the configuration package’s interface, a standardized way to declare prerequisites for a reusable pattern as well as results from it. Today, the putty between different configuration packages is the produced or modified artifacts, however they are used implicitly and that is not enough when discussing a tool.

General Knowledge About Standard Systems Engineering Methods

We have used the concept standard systems engineering method a systems engineering method that has its origins outside the target organization and is used by more than one organization, and where the target organization does not control the development. In Section 1.2 we discussed dependencies to the method-supplier, which become a central issue during implementation and configuration of such
Meta-Method for Method Configuration

methods. We believe that an improved understanding of this phenomenon is needed. With regard to method configuration issues, such knowledge would be valuable when deciding when in-house complements should be developed and with what focus. One concrete example is the implementation of tool support for RUP. We find it an interesting challenge to build a tool using the foundation from MMC, which complements Rational Software’s standard tools.

8.6 Rational Unified Process for Gathering Oranges?

Let us return one last time to our oranges. Would the initial fictitious project team have performed better with a situational method for gathering oranges? The answer is of course not straightforward and depends on a wide range of factors of which we will consider two. First, the answer depends on the project team’s belief and competence in methods. The team members need to share some basic values, and we cannot disregard the need for competent method users. Second, the answer depends on the suitability of the configuration we can achieve. The first issue is not within the scope of this thesis, therefore we turn our attention to the second issue. Is it possible, through the use of MMC, to turn RUP into a method for gathering oranges? Let us briefly address a delimited part of this problem.

The team defined its goal as fulfilled when they had fourteen gathered oranges. Subsequently, the team is not supposed to produce a final artifact, rather to produce a service efficiently. An important part of MMC is the use of experiences, and looking back on the discussion in Chapter 1 we find that the environment issue is crucial for this service. Based on this superficial analysis we have found the characteristic ‘How crucial is knowledge about the environment?’, which is defined below. The characteristic’s dimension has two values Crucial and Unimportant (N/A is not considered as an actual value). These values reference two configuration packages of which we focus on the former.

<characteristic name = "How crucial is knowledge about the environment?">
  <value valueid = "Crucial" cpid = "Change The Environment"/>
  <value valueid = "Important" cpid = "Do Not Prepare Existing Environment"/></value>
  <value valueid = "N/A" cpid = "N/A"></value>
</characteristic>

The project team had limited knowledge about the environment. In order to efficiently gather the oranges, the team needs to use its knowledge in an intelligent way. Hence, we demarcate the configuration package to the ‘Environment’ discipline in RUP. The discipline’s purpose is to define and manage the environment in which the project is to be conducted. In the workflow detail ‘Prepare Environment for Project’ we find an activity of interest, the ‘Assess Current Organization’. In this activity, we described the current status of people's competencies, people's attitudes, problems and so forth. This is a highly important activity in order to make
Conclusions and Further Research

knowledge about the environment inter-subjective. One possible way to use this knowledge is to deliberately change the environment. Recapturing the project environment, it consisted of a room with furniture. Consequently, based on this knowledge these could be rearranged in order to fit the team and the task better. The suggestion from Chapter 1 was to divide the room into four sections, making the demarcation with furniture. Each section is then allocated to a sub-team in order to facilitate a more focused search.

However, in RUP there is no prescribed action with the purpose of rearranging the environment. We can find prescribed actions about selection and set-up of tools, however not the kind of change we have in mind. The former activities are of no importance and can be suppressed, and we need to introduce a new activity. In the broad outline, below, we have introduced this activity in configuration package.

```xml
<configurationpackage id = "Degree Of New Functionality:
Existing Functionality With New GUI">
  <prescribedaction name = "Prepare Environment for Project"
layer = "Workflow Detail">
    <classification status = "Emphasized_extended">It is
    crucial to achieve a suitable environment
  </classification>
  <prescribedaction name = "Assess Current Organization"
layer = "Activity">
    <classification status = "Emphasize_as_is">Important to
    have a inter-subjective knowledge about the environ-
    ment</classification>
  </prescribedaction>
  <prescribedaction name = "Select and Acquire Tools" layer = "Activity">
    <classification status = "Skip">No tools available
  </classification>
  <prescribedaction name = "Rearrange Environment" layer = "Activity">
    <classification status = "New">Need to achieve a
    suitable environment</classification>
  </prescribedaction>
  ...
</configurationpackage>

We have left out some blanks, i.e., the ellipses in this configuration package. Thus, there remains some configuration work to do in order to complete this configuration package as well as tailoring the remaining parts of the base method. Subsequently, this is a fifth area of future research, one in which our fictitious project team will be a major stakeholder.
REFERENCES


References


Conallen, J (2000). *Building Web Applications with UML*. Addison-Wesley, Reading, MA, USA.


References


The world of systems engineering methods is changing as rigorous ‘off-the-shelf’ systems engineering methods become more popular. One example of such a systems engineering method is Rational Unified Process. In order to cover all phases in a software development process, and a wide range of project-types, such methods need to be of an impressive size. Thus, the need for configuring such methods in a structured way is increasing accordingly. In this thesis, method configuration is considered as a particular kind of method engineering that focuses on tailoring a standard systems engineering method. We propose a meta-method for method configuration based on two fundamental values: standard systems engineering method’s rationality and reuse. A conceptual framework is designed, introducing the concepts Configuration Package and Configuration Template. A Configuration Package is a pre-made ideal method configuration suitable for a delimited characteristic of a (type of) software artifact, or a (type of) software development project, or a combination thereof. Configuration Templates with different characteristics are built combining a selection of Configuration Packages and used as a base for a situational method. The aim of the proposed meta-method is to ease the burden of configuring the standard systems engineering method in order to reach an appropriate situational method.

**Nyckelord**


Compiler Generation for Parallel Languages from Denotational Specifications, 1993.


Design of a Multiple View Presentation and Interaction Manager, 1989.

A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.


Johan Ringström: Compiler Generation for Parallel Languages from Denotational Specifications, 1993.


No 488   Eva Toller: Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
FHS 8/95 Dan Fristedt: Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
No 538   Staffan Flodin: Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 546   Magnus Werner: Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96 Mikael Lind: Affärspanlinjert kultur i förändringsprocess - etnografi och tillämpning av synsätt och metod, 1996.
No 558   Patrik Nordling: The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 563   Niclas Andersson: Compilation of Mathematical Models to Parallel Code, 1996.
No 587   Jörgen Lindström: Cheferns användning av kommunikationsteknik, 1996.
No 589   Esa Falkenroth: Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591   Niclas Wahllöf: A Default Extension to Description Logics and its Applications, 1996.
<table>
<thead>
<tr>
<th>No</th>
<th>Author</th>
<th>Title</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>No 599</td>
<td>Peter Ingels:</td>
<td>A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997</td>
<td></td>
</tr>
<tr>
<td>No 607</td>
<td>Per-Arne Persson:</td>
<td>Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997</td>
<td></td>
</tr>
<tr>
<td>No 609</td>
<td>Jonas S Karlsson:</td>
<td>A Scalable Data Structure for a Parallel Data Server, 1997</td>
<td></td>
</tr>
<tr>
<td>FiF-a 4</td>
<td>Carita Åbom:</td>
<td>Videomötesteknik i olika affärsituationer - möjligheter och hinder, 1997</td>
<td></td>
</tr>
<tr>
<td>FiF-a 6</td>
<td>Tommy Wedlund:</td>
<td>Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T30-bolag inom ABB, 1997</td>
<td></td>
</tr>
<tr>
<td>No 615</td>
<td>Silvia Coradeschi:</td>
<td>A Decision-Mechanism for Reactive and Coordinated Agents, 1997</td>
<td></td>
</tr>
<tr>
<td>No 627</td>
<td>Fredrik Eklund:</td>
<td>Declarative Error Diagnosis of GAPLog Programs, 1997</td>
<td></td>
</tr>
<tr>
<td>No 629</td>
<td>Gunilla Idefors:</td>
<td>KriSSpel coH Informationsteknik inför en oförrutsigbar framtid, 1997</td>
<td></td>
</tr>
<tr>
<td>No 639</td>
<td>Jukka Mäki-Turja:</td>
<td>Smalltalk - a suitable Real-Time Language, 1997</td>
<td></td>
</tr>
<tr>
<td>No 640</td>
<td>Juha Takkinen:</td>
<td>CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997</td>
<td></td>
</tr>
<tr>
<td>No 643</td>
<td>Man Lin:</td>
<td>Formal Analysis of Reactive Rule-based Programs, 1997</td>
<td></td>
</tr>
<tr>
<td>No 653</td>
<td>Mats Gustafsson:</td>
<td>Bringing Role-Based Access Control to Distributed Systems, 1997</td>
<td></td>
</tr>
<tr>
<td>FiF-a 13</td>
<td>Boris Karlsson:</td>
<td>Methodanalysis för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997</td>
<td></td>
</tr>
<tr>
<td>No 674</td>
<td>Marcus Bjäreland:</td>
<td>Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998</td>
<td></td>
</tr>
<tr>
<td>No 676</td>
<td>Jan Håkegård:</td>
<td>Hierarchical Test Architecture and Board-Level Test Controller Synthesis, 1998</td>
<td></td>
</tr>
<tr>
<td>No 688</td>
<td>Per-Ove Zetterlund:</td>
<td>Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998</td>
<td></td>
</tr>
<tr>
<td>No 677</td>
<td>Jimmy Tjäder:</td>
<td>Projektledaren &amp; planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998</td>
<td></td>
</tr>
<tr>
<td>FiF-a 14</td>
<td>Ulf Melin:</td>
<td>Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998</td>
<td></td>
</tr>
<tr>
<td>No 700</td>
<td>Patrik Häglund:</td>
<td>Programming Languages for Computer Algebra, 1998</td>
<td></td>
</tr>
<tr>
<td>FiF-a 16</td>
<td>Marie-Therese Christiansson:</td>
<td>Inter-organisatorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998</td>
<td></td>
</tr>
<tr>
<td>No 712</td>
<td>Christina Wennemans:</td>
<td>Information om immateriella resurser - Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998</td>
<td></td>
</tr>
<tr>
<td>No 723</td>
<td>Henrik André-Jönsson:</td>
<td>Indexing time-series data using text indexing methods, 1999</td>
<td></td>
</tr>
<tr>
<td>No 725</td>
<td>Erik Larsson:</td>
<td>High-Level Testability Analysis and Enhancement Techniques, 1998</td>
<td></td>
</tr>
<tr>
<td>No 731</td>
<td>Åsa Jansson:</td>
<td>Miljöhänsyn - en del i företags styrning, 1998</td>
<td></td>
</tr>
<tr>
<td>No 733</td>
<td>Thomas Padron-McCarthy:</td>
<td>Performance-Polymorphic Declarative Queries, 1998</td>
<td></td>
</tr>
<tr>
<td>No 734</td>
<td>Anders Bäckström:</td>
<td>Värdebygande och underintäkter - möjligheter och hinder, 1998</td>
<td></td>
</tr>
<tr>
<td>FiF-a 21</td>
<td>Ulf Seigerroth:</td>
<td>Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999</td>
<td></td>
</tr>
<tr>
<td>FiF-a 22</td>
<td>Fredrik Öberg:</td>
<td>Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998</td>
<td></td>
</tr>
<tr>
<td>FiF-a 25</td>
<td>Bengt E W Andersson:</td>
<td>Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsscenario i samverkan om utbyte av information, 1998</td>
<td></td>
</tr>
<tr>
<td>No 742</td>
<td>Pawel Pietrzak:</td>
<td>Static Incorrectness Diagnosis of CLP (FD), 1999</td>
<td></td>
</tr>
<tr>
<td>No 748</td>
<td>Tobias Ritzau:</td>
<td>Real-Time Reference Counting in RT-Java, 1999</td>
<td></td>
</tr>
<tr>
<td>No 751</td>
<td>Anders Ferntoft:</td>
<td>Elektronisk affärscommunication - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader, 1999</td>
<td></td>
</tr>
<tr>
<td>No 752</td>
<td>Jo Skåmedal:</td>
<td>Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999</td>
<td></td>
</tr>
<tr>
<td>No 753</td>
<td>Johan Alvehus:</td>
<td>Mötets metafor - en studie av berättelser om möten, 1999</td>
<td></td>
</tr>
<tr>
<td>No 754</td>
<td>Magnus Lindahl:</td>
<td>Bankens villkor i låneavtal vid kredittävling till högt belånade företagsförvärv: En studie ur en agenteoretiskt perspektiv, 2000</td>
<td></td>
</tr>
<tr>
<td>No 766</td>
<td>Martin V. Howard:</td>
<td>Designing dynamic visualizations of temporal data, 1999</td>
<td></td>
</tr>
<tr>
<td>No 769</td>
<td>Jesper Andersson:</td>
<td>Towards Reactive Software Architectures, 1999</td>
<td></td>
</tr>
<tr>
<td>No 775</td>
<td>Anders Henriksson:</td>
<td>Unique Kernel Diagnosis, 1999</td>
<td></td>
</tr>
<tr>
<td>FiF-a 30</td>
<td>Pär J. Ågerfalk:</td>
<td>Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999</td>
<td></td>
</tr>
<tr>
<td>No 787</td>
<td>Charlotte Björkregren:</td>
<td>Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999</td>
<td></td>
</tr>
<tr>
<td>No 788</td>
<td>Håkan Nilsson:</td>
<td>Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000</td>
<td></td>
</tr>
<tr>
<td>No 790</td>
<td>Erik Berglund:</td>
<td>USE-Oriented Documentation in Software Development, 1999</td>
<td></td>
</tr>
<tr>
<td>No 791</td>
<td>Klas Gäre:</td>
<td>Verksamhetsförändringar i samband med IS-införande, 1999</td>
<td></td>
</tr>
<tr>
<td>No 800</td>
<td>Anders Subotic:</td>
<td>Software Quality Inspection, 1999</td>
<td></td>
</tr>
<tr>
<td>No 807</td>
<td>Svein Bergum:</td>
<td>Managerial communication in telework, 2000</td>
<td></td>
</tr>
</tbody>
</table>