*Master Thesis*
*Software Engineering*
*Thesis no: MSE-2006-02*
*January 2006*


# Model Driven Architecture
## - Test Methods and Tools

**Renas Reda**
**Yusuf Tözmal**

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 2x20 weeks of full time studies.

**Contact Information:**
Authors: Renas Reda, Yusuf Tözmal
E-mail: renasreda@gmail.com, yusuftozmal@hotmail.com

External advisor(s):
Lars-Ola Damm
Ericsson AB
Lars-Ola.Damm@Ericsson.com

University advisor(s):
Miroslaw Staron
Blekinge Institute of Technology (BTH)
Miroslaw.Staron@bth.se

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

# ABSTRACT

This thesis describes methods and tools available to test products developed with Model Driven Architecture (MDA) frameworks. The purpose of the research presented in this thesis is to find appropriate methods and tools available to test products developed in an MDA compatible way in an industrial setting. To find appropriate methods and tools a literature study as well as a case study were conducted at Ericsson. The results of the case study show that there exist important criteria both from an MDA perspective and Ericsson's own perspective. Based on the criteria a set of tools were evaluated and the results were that Pathfinder PathMATE in conjunction with Rational Software Architect were the most appropriate tools for Ericsson to use when testing their MDA application.

**Keywords:** Model Driven Architecture, Model Driven Development, test methods, test tools, case study, software engineering, testing.

# ACKNOWLEDGEMENTS

We would like to thank the following persons for their support in the thesis:

- Miroslaw Staron, our advisor at BTH who has shared his expertise in MDA with us. We are grateful for all the feedback, ideas and encouragement he has given us throughout the thesis.

- Lars-Ola Damm, our advisor at Ericsson who has answered numerous questions on Ericsson way of working. We are grateful for his continuous support throughout the thesis.

- Joakim Nilsson at Nohau, who has clarified several problems that we have encountered during the thesis. We are grateful for all the information he has provided us on MDD tools, primarily I-Logix Rhapsody.

- Magnus Persson at Telelogic, who has answered several questions necessary for us to understand MDD tools, primarily Telelogic TAU G2.

- John Hammel at Pathfinder, who has provided us with a great amount of information on Pathfinder PathMATE and model execution in general.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1 INTRODUCTION

## 1.1 Background

Model Driven Architecture (MDA) is a software development lifecycle that uses models as its core development artefacts [1]. The idea behind MDA is to raise the level of abstractions in software engineering to develop complex applications in simpler ways [2, 3]. An application is currently being developed at Ericsson in a MDA compatible way.

A literature study has been initiated to find which MDA test methods are available and what tools exist to implement them. The study is performed to better understand MDA since little emphasis has previously been placed on how to test products developed with the MDA frameworks as the focus so far has been on how to manage designs and coding.

At Ericsson, most of the applications that are being developed communicate with other applications through protocols. According to Varsamau et al. [4], protocols are used to describe the rules that systems in a telecommunication connection use when they communicate [5]. Therefore it is also of interest to the literature study how protocol testing can be performed on models.

The literature study is followed by a case study performed at Ericsson. The goal of the case study is to find the optimal tool for the project group to use when testing their MDA application that is currently being developed.

The evaluation of the case study has been done based on technical aspects, no financial aspects have been considered.

## 1.2 Aim

The aim of this thesis is to find appropriate methods and tools available for testing products developed in an MDA compatible way and to suggest the most suitable tool for testing the MDA application currently being developed at Ericsson.

## 1.3 Objectives

The objectives of this research are to:

1. Investigate methods for testing MDA products.
2. Determine the difference between the methods.
3. Investigate how protocol testing can be performed on models.
4. Investigate different tools available to test MDA products.
5. Suggest the tool that is most suitable for the project group at Ericsson when testing the MDA application that is currently under development.

Steps 1-4 are intended to be done based on a literature study, while step 5 is based on the case study.

## 1.4 Research context

An application is currently being developed at Ericsson in an MDA compatible way. The project group which is responsible for its development consists of 19 members. The application is being developed with Rational Software Architect on the J2EE platform which is a version of Java for developing web based applications [6]. The purpose of the application will be to manage services in mobile networks. The framework used to develop the application does not fulfill all the standards of MDA. The application is however being developed with MDA in mind and therefore it is called Ericsson's MDA application throughout the thesis.

## 1.5 Limitations

As this is a research conducted at Ericsson and specifically aimed at Ericsson's needs, it might not be feasible to generalize the results to other organizations. The case study was performed with Ericsson's current way of using tools in mind.

The adoption level of MDA might also be at another level in other organizations thus making it hard to generalize the results.

## 1.6 Research methodology

A literature study is conducted along with interviews with software developers that have extensive knowledge of MDA inside and outside of the studied organizations. There are three types and styles of interviews [7]: fully structured interview where predetermined questions with fixed wording are used, semi-structured interview where predetermined questions are also used but the order can be changed during the meeting, and unstructured interviews where the interviewer has a general area of concern but no predetermined questions are used [7]. The interview type that is used in the thesis is a semi-structured interview. This approach has been chosen as additional questions can be introduced during the interviews. Also the question wordings used might change during the course of the interview so that the questions used are clear to the respondents [7]. Interviews are performed to supplement the literature. Interviews are also performed to get better knowledge of MDA, to get recommendations of test tools and also to get opinions on which MDA test methods that are good.

The literature study is performed to get a clear picture of MDA. The literature study is also performed to learn of any existing advances regarding testing products developed in an MDA compatible way. The research is important in order to form an opinion on possible criteria.

After learning about MDA and aspects concerning it get clearer, a research on what tools exist to test products developed in an MDA compatible way is started. This research is performed by the use of the internet and also by contacting people that have experience of developing products with the MDA frameworks. This research is performed to see what different methods the test tools support so that criteria can be developed from that information.

The criteria that are developed from the literature study and the test tool research will be used in the case study. A case study is an empirical research method. According to

Wohlin et al. [8] case studies are used for monitoring projects, activities or assignments. The empirical research used in this study is divided into quantitative and qualitative research methods. A quantitative research method means that numeric data is collected and is analysed by the use of statistical methods [8]. Questionnaires are used as instrument to collect the data. According to Robson [9] the disadvantages of using questionnaires are that the respondents' memory, knowledge and experience may have a negative effect on their answers. The advantages of using questionnaires are that they provide a simple approach to collect data.

Criteria are also developed by qualitative research. Qualitative research in concerned with studying objects in their natural settings [7, 8]. We are located in the same room as the project group and consequently the discussions and issues surrounding us will be interpreted. According to Creswell [8], interpretations mean that the qualitative researchers filter the data through a personal lens. These interpretations will be used by us to develop criteria.

Developers are then interviewed in order to evaluate the found criteria and assess their suitability for Ericsson's purposes. The criteria are then prioritized by the subject group with the use of questionnaires. Depending on the criteria the test tools support, the optimal tool for testing the MDA application currently being developed at Ericsson is found.

## 1.7 Research questions

Our main research question is: what different methods and tools are available for testing and validating telecom focused MDA products?

The main research question is then divided into five detailed questions:

- Which methods can be used to test products developed in an MDA compatible way?
- Which tools are present that implement the methods found?
- How can protocol testing be performed on MDA models?
- What is the optimal tool for Ericsson to use when testing their MDA application?
- What are the requests for tools and methods imposed by Ericsson?

## 1.8 Expected outcomes

We expect to find important criteria when researching model testing tools. We also expect to find out a list of tools that are prioritised according to Ericsson's expectations and appropriate for testing MDA applications. We aim to find an optimal tool to test the MDA application currently being developed. We finally expect that one or several methods and tools will be more effective to use when testing products of this family.

## 1.9 Thesis outline

Chapter 2 describes MDA and the benefits of using it in relation to one process model and two process frameworks. Chapter 3 presents how testing at Ericsson is conducted. In chapter 4 different methods of testing a product developed in an MDA compatible are described. Chapter 5 presents tools that implement the test methods described in chapter 4. Chapter 6 presents the case study and the evaluation of test tools. Chapter 7 presents the thesis conclusion. Finally chapter 8 presents ideas of future work.

# 2 MODEL DRIVEN ARCHITECTURE

This chapter describes Model Driven Architecture (MDA) and the most important aspects related to it. Two process frameworks and one process model will also be explained and compared to MDA.

MDA is a software development lifecycle process that has been created by the Object Management Group (OMG) and was introduced in 2001 [1]. This process uses the Unified Modeling Language (UML) as the main development language. The main idea behind MDA is to use models as core development artefacts and thus be able to separate platform specific data from the software development process. Developing applications without platform specific terms makes it easier and less costly to port them to different platforms [1].

As the use of models are the core development artefact in MDA it helps deal with larger and more complex applications in simpler ways. It raises the level of abstraction in software development [2]. The ability to automate testing and transformation of models e.g. Platform Independent Models (PIM) to Platform Specific Models (PSM) is also something that propagates for MDA development [10]. MDA was introduced to reduce both time and cost of developing products as well as improve application quality [3].

As previously mentioned MDA uses UML models as the main development language, UML has a specification above 1000 pages and is a heavy language; it is not likely that one learns all of the UML language. Alex E. Bell [11] thinks that the software companies has UML fever and describes symptoms for it. The most dangerous symptom according to Bell is that UML models have turned into the product. He means that the software companies shift their focus from developing software to create superb models advertised as being only minor transformations shy of the deliverable source code.

In the following sections UML, executable UML (xUML) and UML profiles are explained, MDA and the kind of the models that are part of it are explained (Computation Independent Model (CIM), PIM and PSM). The transformations that can be done and the logic behind them are also explained. The UML metamodel is also explained. Finally some other software development processes available are outlined and the benefits of developing software with the MDA process are explained. In Figure 2.1, it is illustrated what the MDA lifecycle looks like.

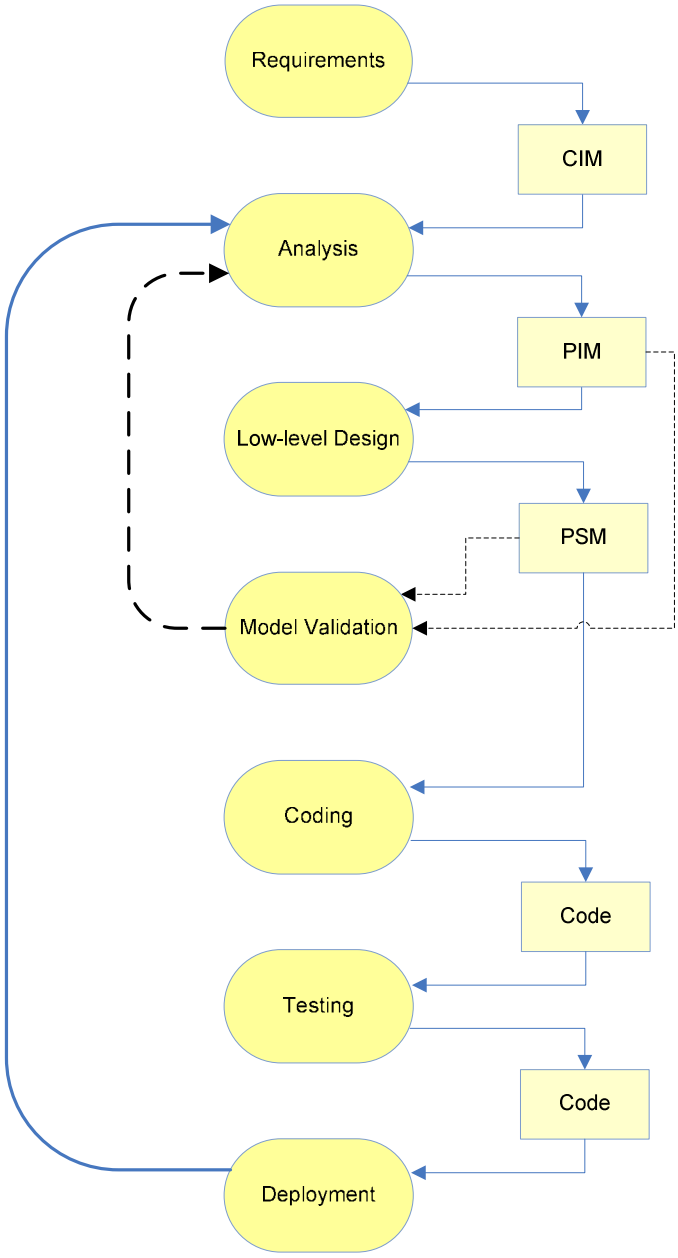



**Figure 2.1** An example MDA lifecycle model [12]

Figure 2.1 illustrates the MDA lifecycle model which is adapted from [12]. The requirements that are in text form are modelled as CIM models. The CIM models give input to the analysis phase, where the PIM models are then developed. The low-level design is done by transforming (see section 3.2) the PIM models into PSM models with a transformation tool (PIM models give input to the low-level design phase, with that input PSM can then be created by the use of transformations). The PIM and PSM models can be validated before they are transformed but that is not a mandatory requirement. If the PSM model is validated, iterations are performed back to the analysis level to update the PIM models (the dotted line illustrates those events). When the PSM models are completed they can be transformed into code. After the code is modified, testing and deployment is performed. If necessary, an additional iteration is made back to the analysis level.

## 2.1 Unified Modeling Language

The core modeling formalism in MDA is the UML. UML is a visual language for specifying, constructing and documenting the artefacts of a system [13]. According to Kleppe et al. [12] and Fuentes-Fernándes and Vallecillo-Moreno [2] UML is the most widely used and best known modeling language.

UML is a general purpose modeling language. This means that it can be used with all major object oriented and component based methods. UML can be applied to many application domains e.g. health, finance, telecom and aerospace. UML can also be used for the major implementation platforms, e.g. Common Object Request Broker Architecture (CORBA), Java 2 Enterprise Edition (J2EE) and Microsoft .NET [2].

### 2.1.1 Executable UML

xUML is a graphical specification language. It is a combination of UML, executable semantics (data on model execution) and timing rules [13]. The executable semantics are means of modifying the models so precisely that they can be executed without adding code. In xUML the execution semantics are fully defined and that is the reason why OMG has extended the UML standard to allow models to be executable. It is by using xUML that PIM (see section 2.2.2) can be executed and be subjected to testing through simulation.

### 2.1.2 UML profile

UML intends to cover as many domains as possible and when it is not able to totally support a specific domain the designers can extend UML to their particular domain or purpose. That is why UML provides extension mechanisms. The profiles mechanism included in UML 2.0 defines a set of UML constructs. This allows the specification of a Meta Object Facility (MOF) model (see chapter 2.6.2) to deal with the specific concepts and notations that are required, in particular application domains. UML profiles allow the customization of any MOF defined metamodel and not just UML [2]. There are UML profiles which are standardized by the OMG, those which are in process of being standardized [14,15], and a whole platform of non-standard company specific ones. An example of a standardised UML profile is the UML 2 Testing Profile (U2TP), which is used to define test specifications and test models. U2TP is described in detail in chapter 4.4 [12].

## 2.2 MDA models

The three types of models used in MDA are CIM, PIM and PSM. They are all explained in the following subsections.

### 2.2.1 Computation Independent Model

CIM is a model that describes what the business environment looks like, later when developing a PIM (described in section 2.2.2) the requirements are derived from CIM [1]. The technical details of the structure of the system are omitted in CIM, the model

is created to explain what the system is supposed to do. It is commonly targeted at the stakeholders of the system [16]. It is also a tool used for communication between domain experts and system architects. CIM usually consists of a few models focusing on different stakeholder concerns.

### 2.2.2 Platform Independent Model

PIM is a base model that captures all the business requirements but not the platform specific specification. PIM captures a viewpoint of the system [17], which is called a domain in MDA. PIM is a description of a software system at a high level of abstraction [12]. According to OMG, there are three major benefits of using PIM [10]:

1. **Validate correctness of the model.** OMG believes it is easier to validate a system that is based on models if it is not cluttered with platform specific details.

2. **Easier to produce new platform capability.** While maintaining the same base model it is easier to port the system to be used on different platforms.

3. **Better defined integration and interoperability.** Integration and interoperability between systems can be more clearly defined in platform-independent terms as platform specific details can be avoided.

### 2.2.3 Platform Specific Model

PSM is a view of the system from a platform specific viewpoint. The PSM contains what is expressed in the PIM with the added concern about the platform details it is intended for [17]. By adding platform specific information to PIM, PSM is developed. This is done with transformations and some manual work (see section 2.3). Because of the relationship between PIM and PSM, when a PIM gets changed the PSM can automatically be re-generated to reflect the changes made to the PIM. In this way the PSM never gets out-of-step with the PIM.

## 2.3 Transformations

One of the key ideas of MDA is the notion of transformations (also called mappings). Transformations work by generating a new model from an existing model according to the transformation definitions [10, 12].

There are 4 different kinds of transformations [10]:

1. **PIM refinements.** This transformation is used when models are specialized, filtered or enhanced without adding any platform specific information. This transformation is often used to refine the model.

2. **PIM to PSM.** The PIM together with the transformation definitions generates the initial version of the PSM.

3. **PSM refinements.** This refinement is needed for component realization and deployment.

4. **PSM to PIM.** This transformation is often used for abstracting models of existing implementation in a particular technology into a PIM. It is a procedure that often requires manual work.

The most common type of transformation is when the PIM gets transformed to an initial version of the PSM by using the transformation definitions that have been specified for a given platform. As there can be multiple transformations to different platforms, a certain PIM can be transformed to numerous PSMs. Code can later be generated from the PSM by a code generator (see Figure2.2).

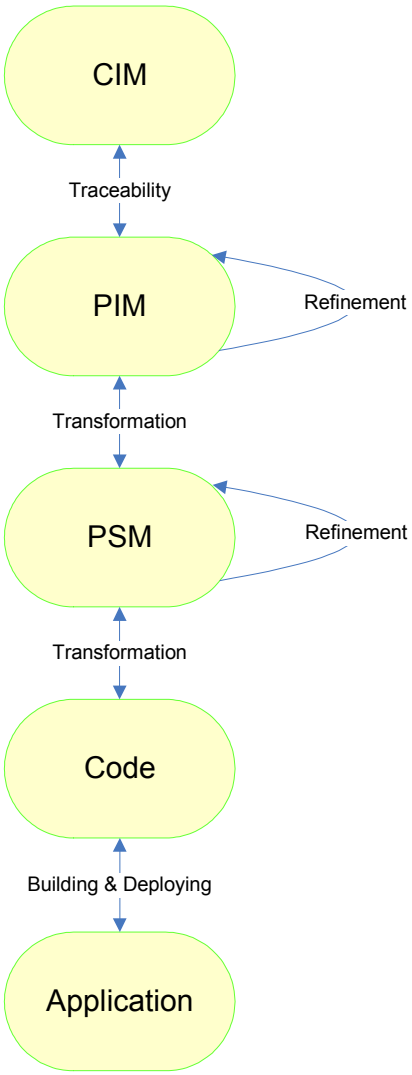



**Figure 2.2** Transformations

Figure 2.2 is adapted from [18]. It illustrates that it is through transformations from PIM to PSM that code can later be generated from the PSM.

## 2.4 Metamodel

Program languages are defined using a grammar in Backus Naur Form (BNF), which is a method suited for text-based languages. Graphical modeling languages like UML are not text based that is why there is a need for a different defining mechanism called metamodeling [12].

A metamodel is a model of a model. The metamodel is of a higher level of abstraction than the modeling language. It describes the abstract syntax of a modeling language. The metalanguage used in MDA is Meta Object Facility (MOF), more about MOF in section 2.6.2 [19]. The UML metamodel defines the language to be used when creating models. A metamodel is also a model and must therefore be defined by a language, in the case of MDA, MOF is used to define metamodels [20] as presented in Figure 2.3.

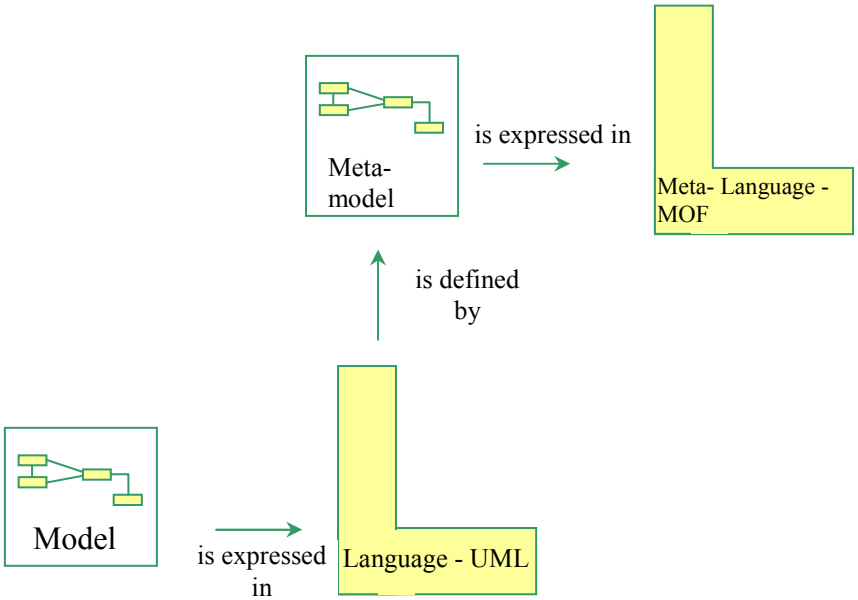



**Figure 2.3** Models, languages, metamodels and metalanguages [12]

Figure 2.3 illustrates that in MDA models are described by the UML language, the UML language is defined by UML metamodels, which are expressed in the metalanguage MOF.

### 2.4.1 OMG's 4-layer metamodeling Architecture

The "four-layered architecture" is an architectural framework for models, meta-models and meta-meta-models. The layers in the four-layered architecture are called M0, M1, M2, and M3. Every layer is an instance of the layer above except for layer M3 which is specified reflexively and therefore does not need layers above.

#### 2.4.1.1 Layer M0: Instances (User data)

The M0 layer is the running system in which the actual instances exist. This layer holds the user data, and the actual object that software is designed to manipulate [12, 21]. "Layer M0 specifies user objects that are instances of the UML user model classes" [19].

#### 2.4.1.2 Layer M1: The Model (User concepts)

The elements of the M1 layer are models. An example would be a UML model of a software system. M1 layer is a model of the M0 layer user data [12, 21].

#### 2.4.1.3 Layer M2: Metamodel (UML concepts)

Layer M2 holds a model of the information at M1. As it is a model of a model, it is often referred to as a metamodel [21].

#### 2.4.1.4 Layer M3: Meta-metamodel (Meta Object Facility)

Layer M3 defines a model of the information at layer M2, and therefore is often called the meta-metamodel. MOF is the standard for defining the layer M3 elements [21]. "Layer M3 specifies meta-metaclasses for the UML metamodel" [19].

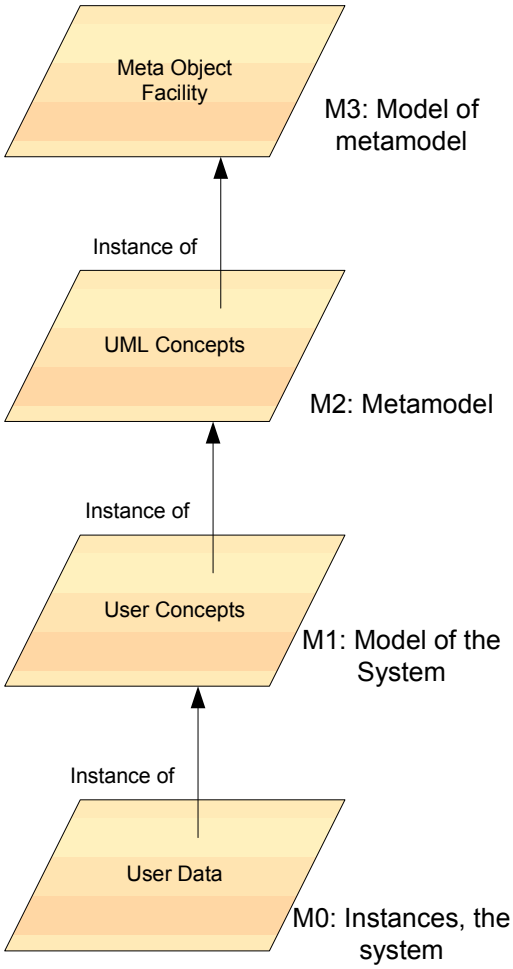



**Figure 2.4** Overview of layers M0 to M3 [12]

Figure 2.4 illustrates how the relationship is between the 4 different layers where M3 is at the top and M0 is in the bottom.

### 2.4.2 Meta Object Facility (MOF)

"The MOF is an OMG standard that defines the language to define modelling languages" [12]. The MOF exists on layer M3 and because there is no higher layer,

MOF is defined using MOF. When a metamodel is expressed in the same language as it self, it is called reflexive modelling [19, 22].

By the MOF definition of a modelling language (metamodel on layer M2) transformations can be defined between modeling languages.

MOF is the standard that UML and Common Warehouse Metamodeling (CWM) metamodels are written in. CWM is specially designed for data warehouse applications; CWM is based on a subset of UML.

## 2.5 Existing Software Development Processes Models

MDA is just a paradigm for software development. The current process models and frameworks can be compatible with it. In the next sections one process model and two process frameworks are described: The traditional waterfall process model, Rational Unified Process (RUP) and Extreme Programming (XP). The two process frameworks and the process model have been chosen due to their popularity and their diversity. They are explained so that it is clear how MDA can complement the shortcomings that they have.

### 2.5.1 The waterfall model

The traditional waterfall process model is one of the oldest software development processes available. It is based on a linear approach of the steps needed to define functions that must be completed before proceeding to the next step. The completion of each step follows with documentation that needs to be approved by the user for the next step of the model to begin.

The first step in the waterfall model is system analysis (see Figure 2.5) where the system requirements are specified and goals are determined from the business process. System design is the next step that comes when the overall architecture is established. Coding is the third step where requirements are transformed into code. Testing is the fourth step where the software is tested for efficiency and effectiveness. In the final implementation and maintenance steps of the system, software is developed and deployed [23]. The waterfall process can be represented as in Figure 2.5

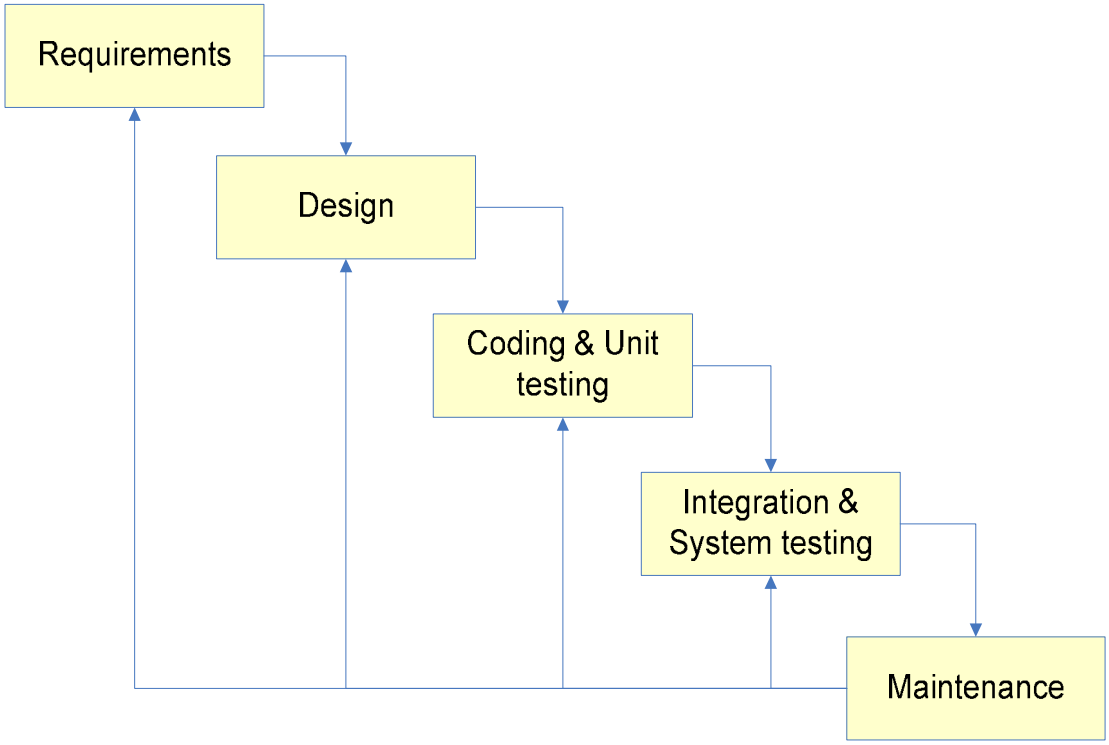



**Figure 2.5** The Waterfall Process

In Figure 2.5 it is illustrated that the requirements analysis is the first step required to develop software. The next steps are system and software design, coding and unit testing, integration and system testing, and maintenance.

The shortcomings of the waterfall process model are that it is inflexible when requirements are altered and that it requires commitment from the user early in the development stages. It is also a process that usually takes long time to complete [24].

### 2.5.2 Rational Unified Process

RUP is a process framework for developing high quality software. It is an adaptable process framework that constitutes a well defined process to allot and handle the work and responsibility that occur in software development organizations. RUP is also a product; it has been developed by Rational. It is included in Rational's suite of development tools.

RUP was created by the observations of the characteristics of failed software development processes. By analyzing them and pinpointing where they went wrong the creators developed some practices that they thought were best in order to develop high quality software within the time frames and budget that has been set.

In RUP, best practices are described. These practices show how to effectively deploy commercially proven approaches to software development. The practices are [26]:

- Developing software iteratively.
- Managing requirements.
- Use of component based architectures.
- Modeling software visually.

- Verifying software quality continually.
- Managing modifications of the software.

One of the major shortcomings of RUP is that it does not include much information on reuse. Reuse is important for companies which want to reduce time of delivery and the overall cost of the project [27].

### 2.5.3 Extreme Programming

XP is a software development process model that is one of the most popular agile software development methodologies. It is originally defined to address the specific needs of software development, conducted by smaller teams in the face of vague and changing requirements [25].

The basics of XP are [28]:

- Stories.
- Developers work in pairs.
- Releases are made frequently.
- Adjust stories before each iterative.
- All code must have unit tests.

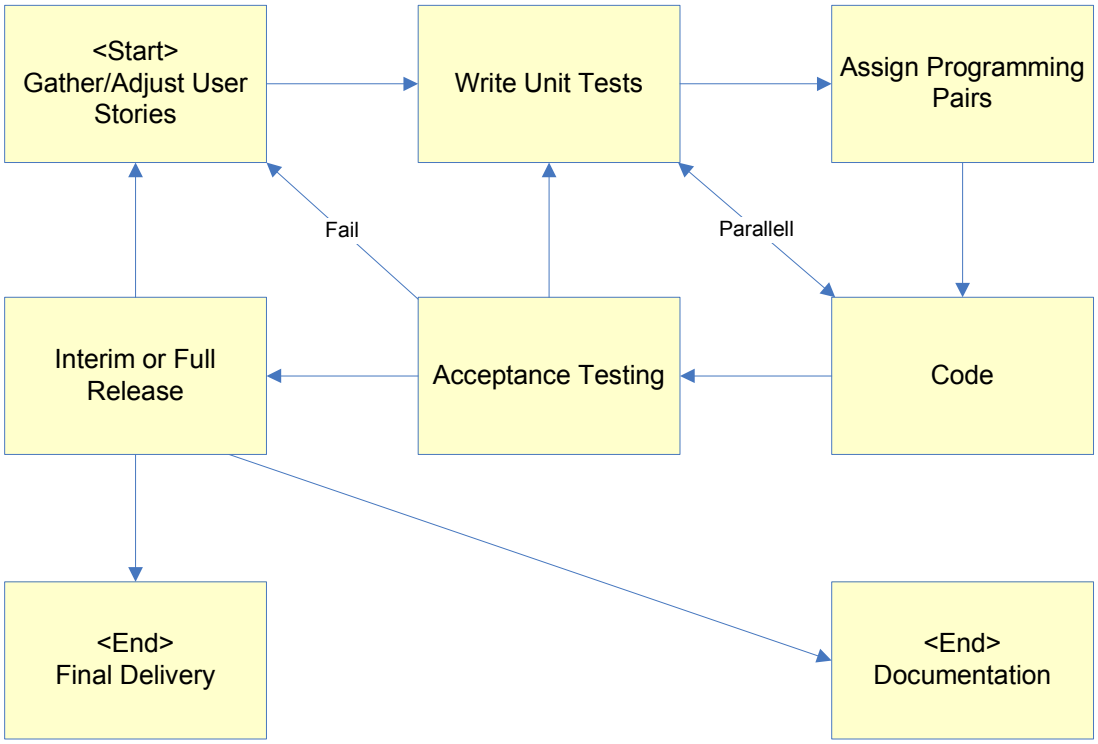



**Figure 2.6** Extreme Programming

Figure 2.6 illustrates that the first step in XP is to gather user stories which represent the system requirements. When the requirements are gathered the user and the developer together decide which requirements that shall be developed in the next iteration. In the next phase unit testing and coding are done in parallel. The specific tasks are assigned to programming pairs. When a unit is developed the user performs an acceptance test, if the test fails the user and developer have another meeting to

adjust the user stories and the process repeats it self. If the acceptance test passes, the next step is an interim release for individual units or a full release for a complete system. The last step is to complete the documentation, and deliver the product to the end user [23]. The difference from heavy-weight processes (like RUP) is that there is not much documentation.

The shortcomings of XP are that it is difficult to use it in large projects because of a lack of sufficient architecture planning, and over focusing on early results [29]. According to Muller and Padberg [30], XP does not guarantee that the customer's needs are actually satisfied because it does not maintain specific requirements from the start of the development.

## 2.6 The Benefits of MDA

The aim of MDA is to decrease the cost of development and increase the quality of the product. Cost is affected as time to market is reduced with the MDA process. The benefits are divided into two categories; cost and quality.

- **Cost.** The cost is measured in financial resources.
- **Quality.** The quality is measured with the number of faults in the system.

| Name | Type | Description |
|---|---|---|
| Effort | Cost | One of the aims with MDA is to reduce the effort for system engineers to make platform specific code. In MDA the importance is in the early stage of system design. Compared to the traditional development process there is more effort being done in design phase of MDA and less in the coding phase. |
| Faults | Quality | When developing according to the MDA process bugs introduced in the design and coding phases are reduced. |
| Re-deployment | Cost | It is common that systems are ported between platforms. Porting takes time and increases the possibility of reduced quality. MDA removes this problem to a one-time cost because the PIM transformations are defined once and transformations to other platforms are possible. |

| Name | Type | Description |
|---|---|---|
| Roundtrip Engineering | Quality | To develop using incremental prototypes is a good approach for increased quality of the system, but the size of the infrastructure often makes this approach difficult and time consuming. The MDA models support roundtrip engineering (see chapter 4.11). That is why prototyping can be done in a very early phase of the MDA process. |
| Application Level Focus | Cost | The MDA process gives the developers the benefit to focus on the application and not worry about the platform details. The application and the platform details are separated in MDA as the design is unaffected by the technology used to achieve system functionality. |
| Traceability | Quality | Often in traditional development processes it is frustrating for developers to use models (they use design language, natural language and programming language). Models could previously not express all the features that the application required. Developers often do not have the time required to refresh the models. With MDA there is good traceability. After the PIM models are refreshed it is much easier to apply the changes to the code in comparison to when using a traditional development process. |
| Skill | Cost | MDA reduces the amount of technical skill required to develop an application. Fewer engineers need to know about the technical details of the platform. |

**Table 2.1** The Benefits of MDA

Table 2.1 illustrates that MDA has several different advantages. The table contains the most important cost and quality benefits.

# 3 TESTING AT ERICSSON

At Ericsson testing is a subject that gets a lot of attention due to the complexity of mobile networks. There is a strong urge to improve the testing phase while optimizing it and making it shorter. Ericsson uses four different types of testing phases:

1. **Basic test.** This type of test includes unit and component testing. It is used to make sure that the units of the system function properly.

2. **Function test.** This type of test is intended to test if the functional requirements posed on the system work in a simulated environment.

3. **System test.** This type of test is intended for the whole system. This testing phase is used to make sure that the system operates as specified on a target platform.

4. **Regression test.** This test is used when an error has been corrected.

At present time the smallest phase of them all is the basic test phase. This phase gets the smallest amount of work, approximately 5% of the total effort. Function and system testing are however two phases that Ericsson spends a lot of time on. If a test case generates an error, the error report is sent to the responsible developer. When the error has been corrected and verified, a regression test is also needed to ensure that all the other parts that could possibly be affected by the change still function properly.

## 3.1 Test tools

There are numerous test tools being used at Ericsson at the moment. Ericsson used to develop their own testing tools before, but the trend is now to buy commercial test tools instead. The reason for this is that it costs more to develop and maintain in-house tools, and it is not part of Ericsson's core business.

In every new application that is being developed at least one tool is used for basic, function and system testing. The most common tools and frameworks used are:

1 A basic test framework built on C++ for component level testing. Works with Extensible Markup Language (XML). Used to test against an in-house developed component server i.e. Framework for Flexible Distributed Systems (FDS).

2 XUnit for basic testing. Mostly JUnit [31] is used but also other types of JUnit like frameworks.

3 TITAN is developed in-house at Ericsson. It uses The Testing and Test Control Notation (TTCN-3) [32] which is a black-box test language. It is used approximately 90% of the time in function test. Not as established in system test but its usage rate is increasing at a rapid rate.

When function and system tests are performed at Ericsson, it is very important that the tests are based on the proper protocol specifications. XML schemas and Abstract Syntax Notation number One (ASN.1) documents are used to specify the types and

constraints that are valid when sending and receiving messages from the different protocols connected to their systems.

When developers at Ericsson have finished developing their system from the XML schema they at least need to perform a boundary value testing, to test what happens if the value entered is less then allowed, more than allowed and also what happens when the value is entered correctly, to see that the program really works.

As Ericsson develops telecom systems that communicate with other applications, GUI testing has never been a big issue. Most of the time manual testing has been sufficient. The MDA application that is currently being developed will however require more GUI testing as it is an application that will communicate a great deal with humans and not only with other applications through protocols.

## 3.2 Integrated testing platform

The ambition at Ericsson is to be able to integrate all their tools by using a common platform. The Eclipse platform is seen as the best suited platform to do this (see Figure 3.1). One of the sub projects of Eclipse is the Test and Performance Tools Platform (TPTP). This platform has been created to provide an open and common development platform for test and performance tools.

According to a modeling tool manager at Ericsson, Eclipse TPTP provides a foundation that could make it possible to integrate tools from different vendors. The Eclipse Modeling Framework (EMF) and UML2 aspects of Eclipse TPTP are of particular interest to Ericsson. Ericsson is pushing vendors to comply with these standards so that the different tools can have a common metamodel and a common Application Programming Interface (API). This would make it possible to add important features to any of the tools.

By using an integrated testing platform both the test preparation and test execution on the System Under Test (SUT) will be easier to perform. There should be a much more organized way of working where all the tools are centralized to one location (see Figure 3.1). An integrated test environment will also make it easier to develop and maintain the test tools that exist. Dependencies between elements of the platform are shown in Figure 3.1.

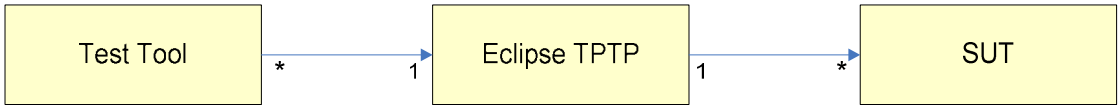



**Figure 3.1** Integrated testing platform

## 3.3 Importance of TTCN-3 in Ericsson

According to a test designer at Ericsson the major advantage with TITAN is that the user can import an XML schema converted to an ASN.1 document and then write templates from that document. A third party program is used to do the translation from XML schema to ASN.1. If a tester uses a test template with information that does not comply with the ASN.1 document, TITAN lets the tester know that the test will not be successful as the information does not comply with the ASN.1 document. If the ASN.1 document is changed then TITAN lets the tester know that the template written, which is affected by the change will not work anymore and needs to be rewritten. Changes in the specification are very common and happen often during the development cycle.

TITAN is also appreciated by Ericsson because numerous protocols have been implemented as test ports in its TTCN-3 language for future reuse. Every time a new protocol has to be defined to test the system it can be implemented and reused in the future.

TITAN also has the advantage of allowing the testers to send messages and listen to numerous different ports at the same time. A system that operates by working with many different ports is something that is very common in the telecom business.

# 4 TESTING METHODS

This chapter describes different test methods available for testing Ericsson's MDA application, and other products developed in an MDA compatible way are described.

In Table 4.1 each method described in this chapter is summarized. The table illustrates the name of the methods; their descriptions; what they test; and which tools implement them.

| Method | Description | Testing is performed on | Tools implementing method |
|---|---|---|---|
| Test case generation from statechart diagrams | Generates test cases from a UML statechart diagram created for testing | Code | Conformiq |
| Test case generation from requirements | Generates test cases from models that specify the requirements posed on the system | Code | T-Vec RAVE |
| Model syntax verification | Validates that the developed models comply to proper UML syntax | Models | Telelogic TAU G2, I-Logix Rhapsody, Compuware OptimalJ |
| Compare generated scenario with expected | Compares the manually defined scenario with the computer generate one | Models | I-Logix Rhapsody, Telelogic TAU G2 |
| Test cases defined with sequence diagrams | Test cases are defined by sequence diagrams that describe how the system is supposed to behave | Models | I-Logix Rhapsody |
| Test cases defined with UML 2 testing profile | Test cases are defined by the UML testing profile | Models | Telelogic TAU G2 |

| Method | Description | Testing is performed on | Tools implementing method |
|---|---|---|---|
| Test cases generated from UML models and C++ code | Test cases are generated from the UML models describing the system as well as the C++ code | Models and Code | I-Logix Rhapsody |
| Test cases generated from use case and sequence diagrams | Test cases are generated from the use case and sequence diagrams that describe the system | Code | Cow Suite |
| Test cases generated from activity diagrams | Test cases are generated from activity diagrams describing the system | Code | No tool supporting this method at the moment |
| Visual runtime analysis | Executes the system to show what happens inside | Models | I-Logix Rhapsody, Pathfinder PathMATE, Rational Rose RealTime, Telelogic TAU G2 |
| Code based test method | Test cases are written in code | Code | Eclipse TPTP, Rational Software Architect, Telelogic TAU G2, TITAN TTCN-3 |

**Table 4.1** Methods of testing products developed in an MDA compatible way.

## 4.1 Test case generation from statechart diagrams

This method generates test cases from a statechart diagram developed for testing. By using UML statecharts a wide range of aspects of a system can be specified. UML statecharts are used to represent the behaviour of an object of a class. Therefore statechart diagrams provide a natural basis for test case generation. A statechart contains of three primary things: states, actions, and transitions. It is through transitions among states that the dynamics of objects are modelled. Offut and Abdurazuk [33] use the label "event-name" for describing transition along with parameters to describe which variables are associated with the event that must be triggered for the transition to be taken place. There are four events specified in UML; call events, signal events, time events, and change events.

One way of generating test cases is by using change events. A change event models an event that occurs when an explicit Boolean expression becomes true as a result of a change in value of one or more attributes or associates. Offut and Abdurazuk claim that these can be used as basis for generating tests as they can be expressed as predicates. Change event is raised implicitly when the associated predicate becomes true. A guard is a precondition that controls whether the transition is taken or not [33]. The difference between using a change event, and a guard is that a guard is only evaluated when an event is dispatched while a change event is evaluated continually until it becomes true.

Offut and Abdurazuk explain that a way of generating a minimum sufficient amount of test cases is by triggering events and attributes so that every transition in every statechart is taken once (see Figure 4.1). To generate the maximum amount of test cases each clause in each predicate should be tested. A clause is a Boolean expression that contains negated Boolean expression; for example relational expressions and Boolean variables. A tester can now decide how much coverage they want the test cases to have of the system depending on the cost and benefit trade-off. An example test process is presented in Figure 4.1

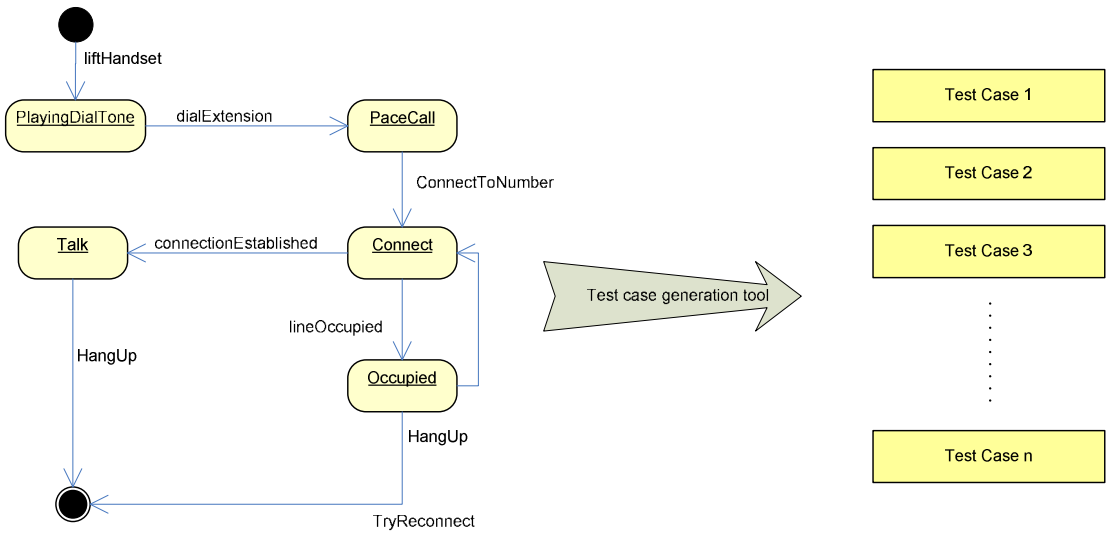



**Figure 4.1** Generate test cases from extended statechart diagram

As illustrated in Figure 4.1 test cases are generated from a statechart diagram that describes a situation, a phone call in this example. Test cases can be generated for each transition from the first state PlayingDialTone, indicating that the line is open, to the last state, when the call is ended (the black dot).

### 4.1.1 Usage of the method

The following steps are required to use the method:

1. The test designer creates a statechart diagram for the system
2. The tester presses test case generation button, and the tool that implements the method generates, and executes the test cases.
3. The tool that implements the method returns the result of test case execution.

## 4.2 Test case generation from requirements

This method works by generating test cases from a proprietary language "T-Vec linear form." The T-Vec linear form is used to model the requirements posed on the system. Modeling of software requirements is different from modelling of software design as it describes what the system is supposed to do, and not how the system is supposed to function [34]. What is very important is that the requirements are consistent and unambiguous. So the tools that implement this method checks to see if the requirements are valid by checking contradictions, or feature interaction defects.

When models have been developed by the test designer the tool that implements the method traverses the logical paths, and determines the locations of boundaries while also identifying reachability problems. It uses a test selection criteria based on domain testing theory. When the models have been checked, test vectors are generated (see Figure 4.2). Test vectors include inputs, expected outputs, and identification information that traces to the associated requirement [34]. It is from the test vectors that test cases are generated.

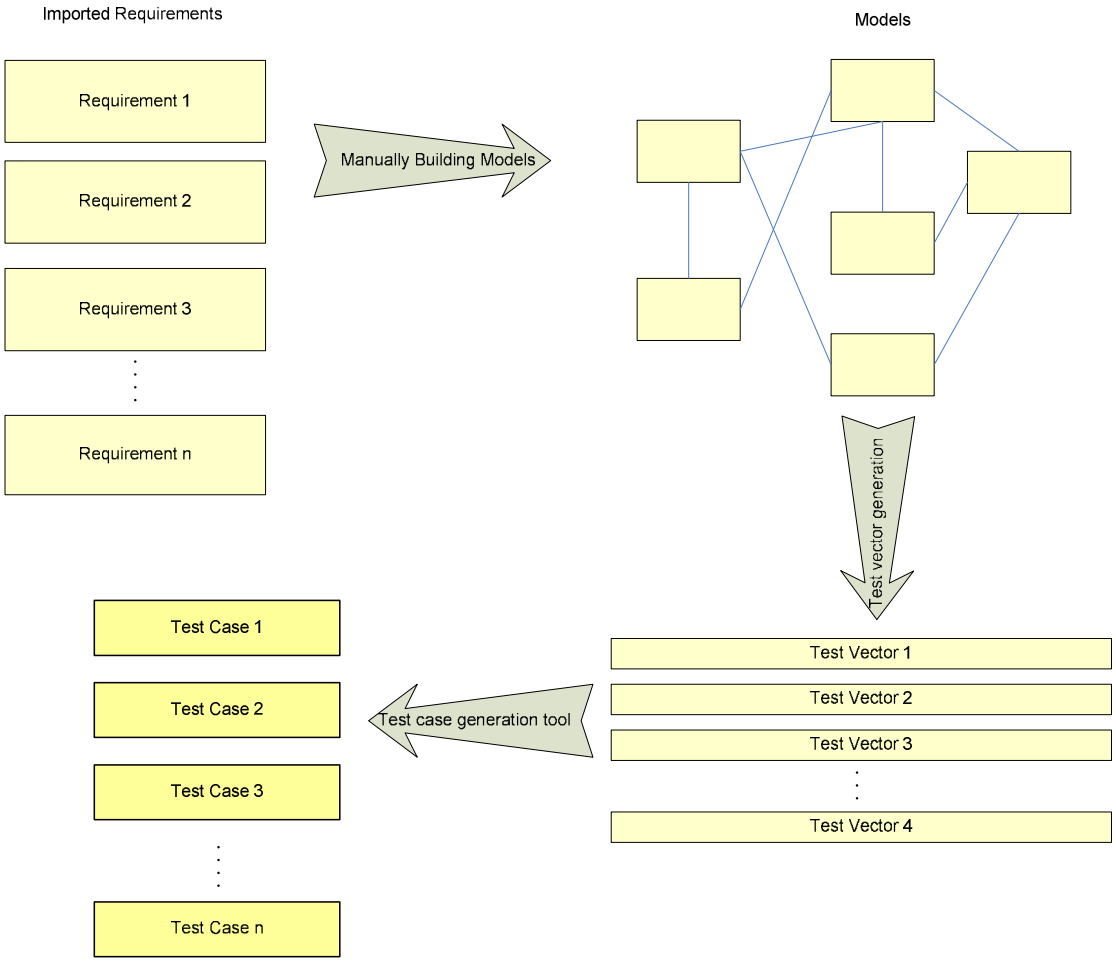



**Figure 4.2** Generate test cases from requirement

Figure 4.2 illustrates how test cases are generated from the requirements. The requirements are imported to the tool. The tester builds models, which reflects the requirements. The test vectors are generated from the model. Each test vector generated by the tool that implements this method.

### 4.2.1 Usage of the method

The following steps are required to use the method:

1. The test designer imports the requirements posed on the system.
2. The test designer builds models reflecting the requirements posed on the SUT.
3. The tester presses the test case generation button and the tool that implements the method generates test vectors.
4. The tool that implements the method generates test cases from the test vectors.

## 4.3 Model syntax verification

According to Shen et al. [35] static validation is used to check whether a model is syntactically valid, i.e. whether the model is an instance of the UML meta-model including the well-formalness rules given by OCL. OCL is a query and expression language for UML that is an integral part of the UML standard [36].

This method works by the use of static validation i.e. by checking that the models conform to accurate syntax rules. It checks that the models do not violate the rules. This method can be run both automatically and manually depending on the tool that implements it. The goal with this method is to identify problems before they impact the software [37].

### 4.3.1 Usage of the method

The following steps are required to use the method:

1. The designer develops the models.
2. The tester presses the syntax verification button and the tool that implements the method shows which syntax rules have been broken.

## 4.4 Compare generated scenario with expected

This method works by comparing an expected scenario with a generated one (see Figure 4.3). The SUT is executed, and sequence diagrams from the chosen parts of the animated system are generated. These sequence diagrams are then compared to the ones that have been defined as an expected scenario. The tester can also choose which parts of the sequence diagrams to compare and which parts to ignore. If the manually created sequence diagram does not match the generated sequence diagram, faults are encountered. The testers can then see exactly were the diagrams differ and use that information to repair the system [38].

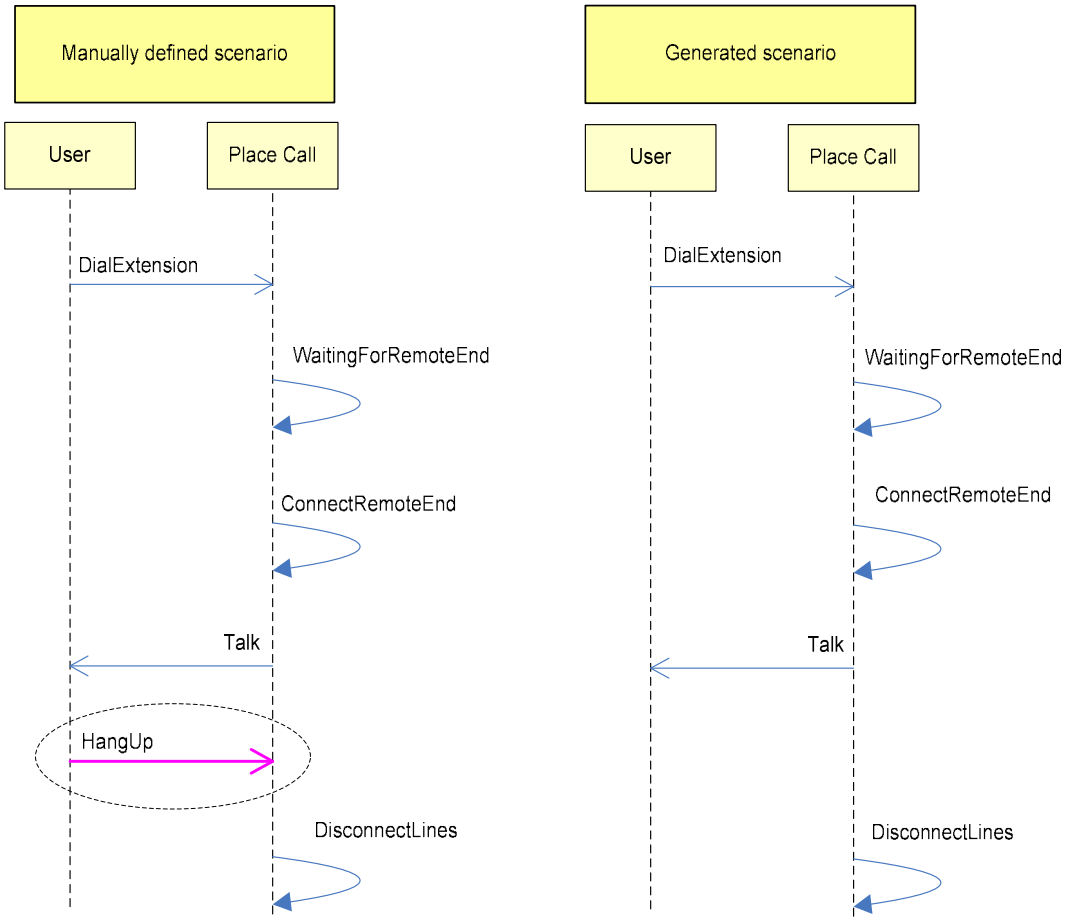



**Figure 4.3** Comparison between the manually defined scenario and the generated one

In Figure 4.3 it is illustrated that the difference between the manually defined scenario and the generated one is that there is a HangUp message missing in the generated scenario. This is clearly shown by the program that implements the method by highlighting (which is indicated by an oval in Figure 4.3) the HangUp message.

### 4.4.1 Usage of the method

The following steps are required to use the method:

1. The test designer defines a sequence diagram of how the system is expected to behave.
2. The tester executes the system by pressing a button.
3. The program that implements the method generates a sequence diagram.
4. The program that implements the method compares the generated sequence diagram with the expected one.
5. If the sequence diagrams match the test case passes.

## 4.5 Test cases defined with sequence diagrams

This method works by letting the test designer create sequence diagrams that express how the SUT is supposed to behave. The test designer declares what messages and signals to send in to the SUT, and then describes what messages or signals to expect back [39]. When the tool that implements the method runs the test case it sends in the messages that are stated in the sequence diagram and waits for the response of the SUT.

Both black-box tests and white-box tests can be defined by programming the tool that implements the method what messages and signals to look for. This is possible by setting the program to observe the messages that go back from the SUT to the test component and ignore the messages that go from the SUT to other components. If the SUT does what is expected of it the test cases pass. If it however does something that the test designer has not defined either a fail or warning arises. The severity of the problem is what decides if it is a fail or warning. If the testing for any reason cannot be completed, an error is reported.

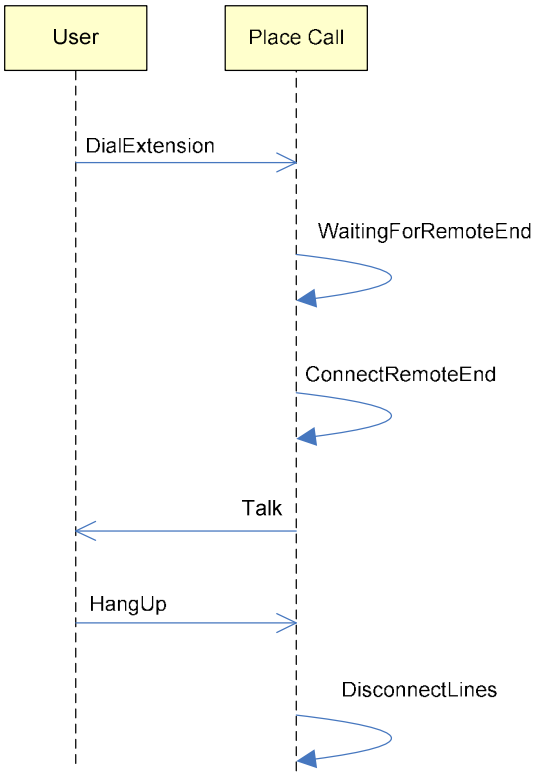



**Figure 4.4** Define test cases with sequence diagram

Figure 4.4 illustrates how test cases developed by the use of sequence diagrams might look like. The messages being sent and expected to be returned are clearly defined by the use of arrows.

### 4.5.1 Usage of the method

The following steps are required to use the method:

1. The test designer creates a sequence diagram of how the system is expected to behave.
2. The tool that implements the method sends the messages that have been stated by the test designer to the SUT and records what is received.
3. If the messages sent back from the SUT match the ones in the test case the test case pass.

## 4.6 Test cases defined with UML 2 Testing Profile

In 2004 the UML 2 Testing Profile (U2TP) became a standard for testing the UML. U2TP provides concepts to develop test specifications, and test models mainly for black-box testing. U2TP was developed because the OMG belief was that also the testing part of MDA should be specified with UML. U2TP was developed with TTCN-3 and JUnit in mind [31, 32]. The UML testing profile was introduced to bridge the gap between designers and test designers by providing a means to test with UML. The profile allows for reuse of UML design documents and test development in an early system development phase [40]. U2TP introduces four logical concept groups covering different aspects [41]:

1. **Test Architecture.** Concepts introduced to specify structural aspects of the test system.
2. **Test Behaviour.** Specifies the behaviours and the objectives that are necessary to evaluate the test objective.
3. **Test Data.** Defines the test data and templates used in test procedures.
4. **Test Time.** Time quantified definition of test procedures.

To specify tests with U2TP activity, sequence, state machines, and interaction diagrams can be used. The most common way to specify a test case with U2TP is through sequence diagrams [42]. The sequence diagrams are used to specify the interactions between the test component and the SUT. The outcome of the test cases can be pass (SUT behaves as expected), fail (SUT does not behave as expected), inconclusive, and error (the test could not be concluded).

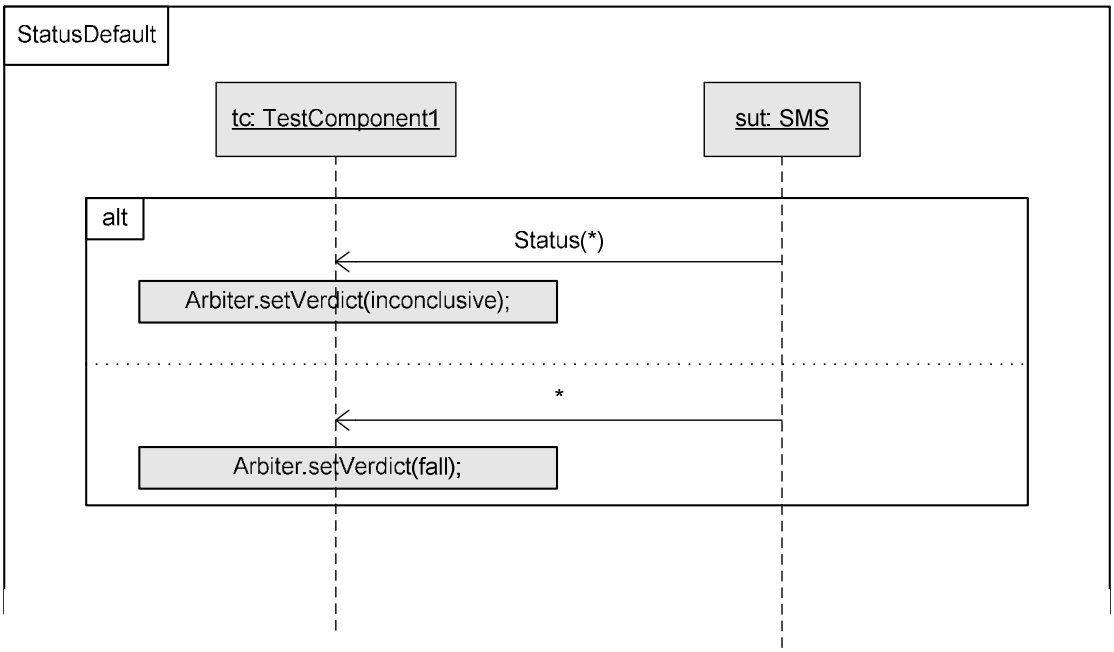



**Figure 4.5** Define test cases with UML 2 Testing Profile

Figure 4.5 illustrates how a test case with U2TP might look like when verdicts have been set.

### 4.6.1 Usage of the method

The following steps are required to use the method:

1. The test designer creates a sequence diagram of how the system is expected to behave with a test component and the SUT.
2. The test designer adds verdicts to the sequence diagram.
3. The program that implements the method runs the test case.
4. The program that implements the method shows if the test case fail or pass.

## 4.7 Test cases generated from UML models and C++ code

Test cases are generated from the UML models and the generated C++ code. This method works by analyzing both the models and the code. From the models the tool that implements the method knows what the input and the output should be. By analyzing the code the tool knows how much of the code it has developed test cases for. It also uses the code to execute the system through a virtual machine. With virtual model execution this method explores the reachable state space of the SUT. By working with different strategies, test cases are generated in an incremental manner.

The test case generation goals are expressed by high-level model elements such as transitions and states [38]. The test cases are saved as sequence diagrams [44].

### 4.7.1 Usage of the method

The following steps are required to use the method:

1. The tester presses the test case generation button.
2. The tool that implements the method generates test cases from the design models and the generated C++ code.
3. The tool that implements the method shows how much code it has generated test cases for.
4. The test cases are stored as sequence diagrams.

## 4.8 Test cases generated from use case and sequence diagrams

This method works by generating test cases from the systems use case and sequence diagrams. The test designer creates a diagram with a hierarchical organization of the UML diagrams together with a number that describes the importance of each diagram. By knowing this the tools implementing this method can choose the most important test cases. The tester can also choose to generate all test cases possible [44].

This method combines two original components, a method that derives the test cases and a strategy for test prioritization and selection which helps to decide which and how many test cases from the derived test cases should be launched [45, 46].

With the use case diagrams the system functionalities are described, and with each use case diagram a relative sequence diagram describes the object interactions and exchanged messages. With the information from those two diagrams test cases can be generated [45, 46].

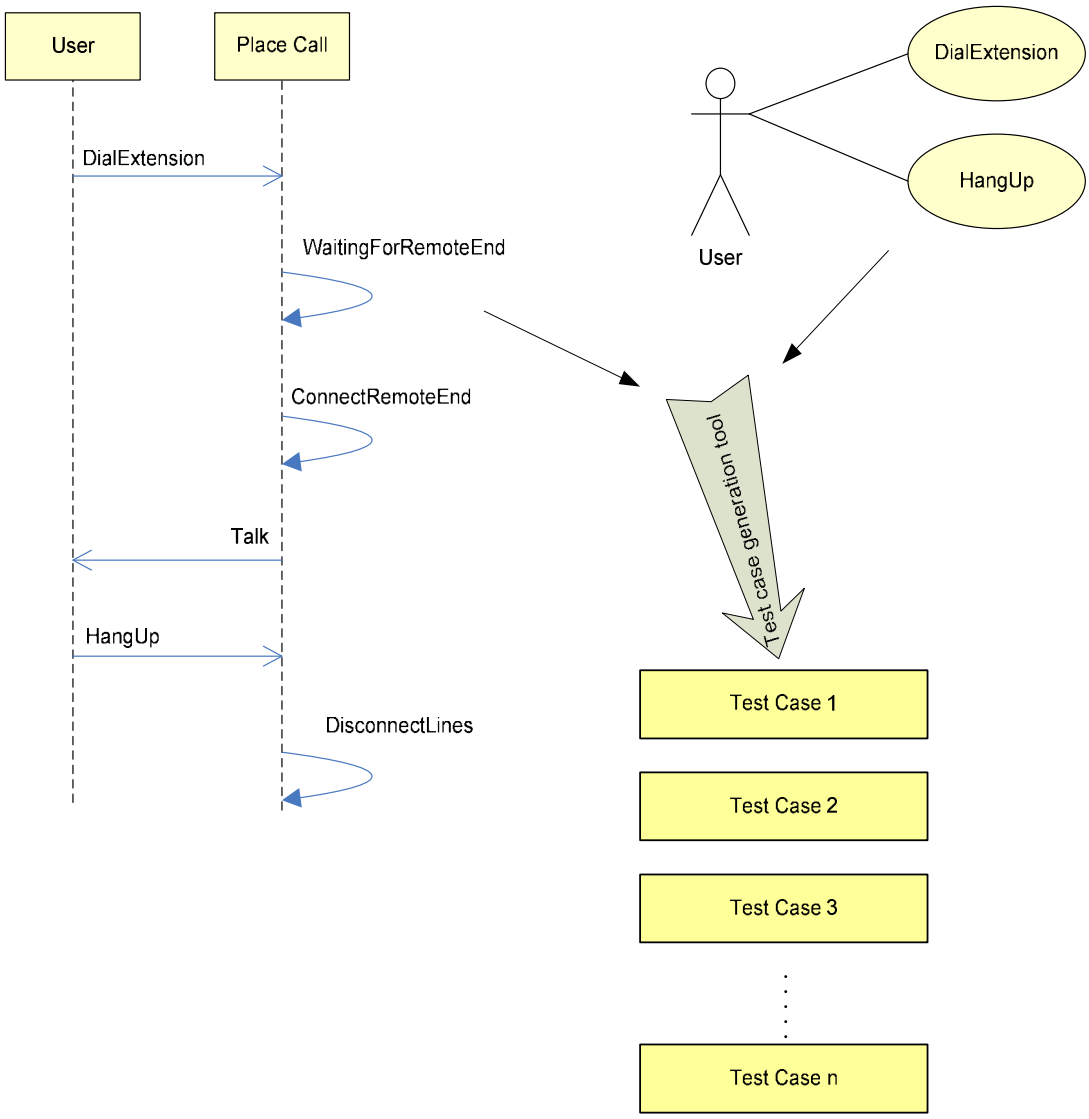



**Figure 4.6** Generate test cases from use case and sequence diagram

Figure 4.6 illustrates how the test cases are generated from the use case scenarios DialExtension, HangUP and the sequence diagrams that describe them.

### 4.8.1 Usage of the method

The following steps are required to use the method:

1. The tester imports the use case and sequence diagrams.
2. The tester presses the test case generation button and the tool that implements the method generates test cases from the models.
3. The test cases are stored in a custom format.

## 4.9 Test cases generated from activity diagrams

Activity diagrams describe the sequential or concurrent flow between activities. They can be used to model the control flow of an operation or to model the dynamic aspects of a group of objects. By emphasizing on the activities of the object it can be used to describe the realization of the operation in the design phase and to describe the sequence of the activities among the objects involved in the control flow.

The activity diagram elements represent aspects of system information, which are essential information of the system. Activity diagrams consist of nodes and edges. The edges represent the occurring sequence of activities, objects involving the activity such as control flows, signal flows, and message flows. The nodes represent processes or process control, including action states, activity states, signal senders, and receivers etc [47].

With this method each possible paths in the activity diagram is executed at least one time. When all of the states are traversed the tool will create test scenarios. When all the test scenarios are developed the generation of test cases can be started. At least one test case can be generated from each test scenario. The constraint conditions are extracted from the guard transitions in each transition of the test scenario sequence. From activity states and corresponding objects the object method sequence, which represents the internal behaviour of the software in runtime, is extracted. Finally proper combination of values and input and output parameters are generated to satisfy the condition constraints. A test case is then formed from the input sequence, expected object method call sequence and expected output form [47].

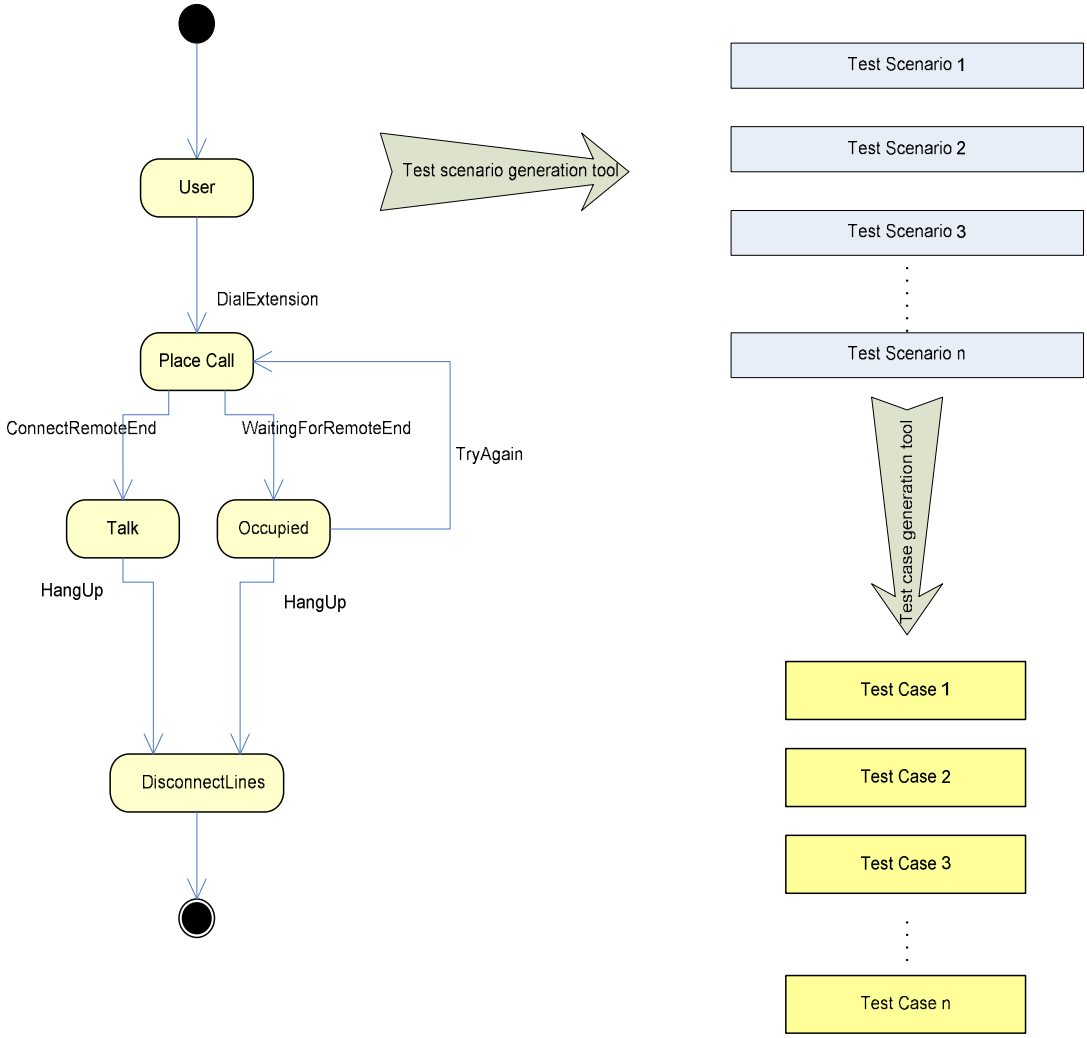



**Figure 4.7** Generate test cases from activity diagram

In Figure 4.7 it is illustrated how test cases are generated from the input sequences and the output sequences. One test case is generated for the input sequence DialExtension and the output sequence WaitingForRemoteEnd.

### 4.9.1 Usage of the method

The following steps are required to use the method:

1. The designer develops an activity diagram that describes an operation.
2. The tester presses the test case generation button.
3. The tool that implements the method generates test cases from the activity diagram.

## 4.10 Visual runtime analysis

Visual runtime analysis is a method that allows the tester to see what happens inside the models while the system is executing. By the use of this method it is possible to encounter errors while developing the system. The tester can see possible errors, and also see errors that are not evident before the execution of the system [38, 48].

What is so good with this method is that the tester can often execute systems that are far from finished. Many times even a PIM (see chapter 2.2.2) can be run and observed. The tester can also see were inside the system the execution pointer is (see Figure 4.8). This means that errors can be found early in the development process thus making it easier and less expensive to correct them [38, 48].

Some tools that implement this method also let the tester do protocol testing by sending different messages and signals into the system to see what happens. Stepping through the system bit by bit is also possible with some tools.

Figure 4.8 illustrates how a tool that implements this method shows exactly were in the statechart the execution pointer is. In this case the execution pointer is in the Connect state.

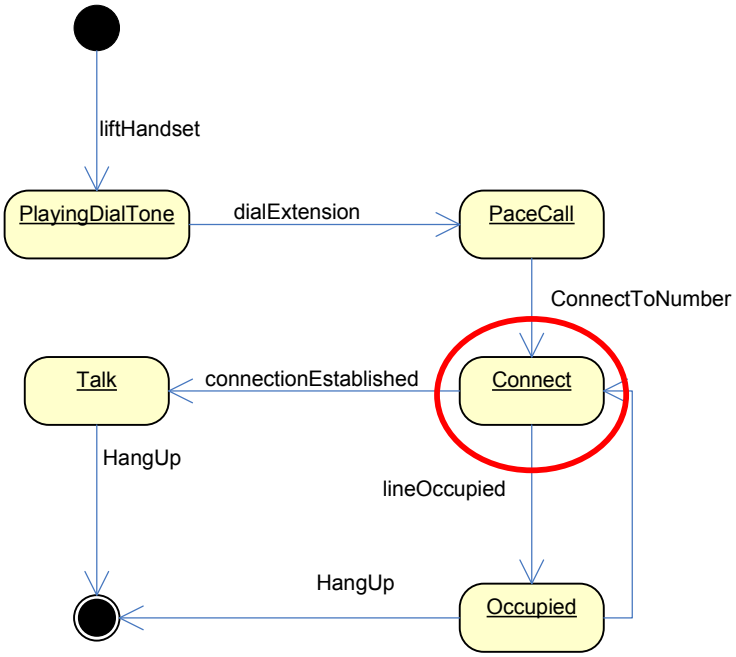



**Figure 4.8** Visual runtime analysis

In Figure 4.8 it is illustrated that the execution pointer is at the Connect state. If the tester sends the message lineOccupied the execution pointer will go to the state Occupied and the red circle will be moved there.

### 4.10.1 Usage of the method

The following steps are required to use the method:

1. The tester presses the model execution button.
2. The program that implements the method executes the design models.
3. The tester sends messages and signals to the system while it is executed.
4. The tester can see possible errors that are generated inside the system.

## 4.11 Code based test method

This method works by testing the code that has been generated from the models. Testing is therefore done in a traditional way in comparison to the MDA oriented

methods described in this chapter [49]. It is possible to test small parts of the system that by using stubs. Stubs are used to simulate behaviour of components that have not been implemented yet. By using stubs components that are finished but surrounded by unfinished components can be tested.

If errors are found when testing the code and the code therefore is changed, many tools offer to roundtrip engineer the code. Roundtrip engineering means that the altered code gets re-generated into the models. So if the code is altered the models get updated to reflect the changes that have been made to it.

Test cases to test the generated code can be unit tests, function tests, regression tests, and system tests. The test developer defines them manually and run them automatically through an engine.

### 4.11.1 Usage of the method

The following steps are required to use the method:

1. The tester develops test cases by code.
2. The tool that implements the method runs the test cases automatically.
3. The tool that implements the method shows which test cases passed and failed.

# 5 TOOLS

This chapter defines which tools that support the different methods from chapter 4. Overall information of the different tools is described in this chapter. More detailed information will be presented when evaluating the tools in chapter 6.

In Table 5.1 it is illustrated which methods each tool implements, and what kind of outputs are generated from them.

| Tool | Methods implemented | Output |
|---|---|---|
| Conformiq | 4.1 | HTML/XML result of the executed test cases |
| Compuware OptimalJ | 4.3 | Result of test completion |
| Cow Suite | 4.8 | Test cases in text format |
| Eclipse TPTP | 4.11 | Result of test completion |
| I-Logix Rhapsody | 4.3, 4.4, 4.5, 4.10 | Result of test completion |
| Pathfinder PathMATE | 4.10 | Analysis of SUT |
| Rational Rose Realtime | 4.10 | Analysis of SUT |
| Rational Software Architect | 4.11 | Result of test completion |
| Telelogic TAU G2 | 4.3, 4.6, 4.8, 4.10 | Result of test completion |
| TITAN | 4.11 | Result of test completion |
| T-Vec RAVE | 4.2 | Test cases in TTCN-3 format |

**Table 5.1** Test tools, the methods they have implemented, and their outputs.

## 5.1 Conformiq

Conformiq is a test tool that generates test cases from statechart diagrams, and executes them against the SUT.

It works by generating test cases from a statechart model that defines what needs to be tested [50]. In the statechart diagram the test designer describes which different directions there are in which the functions inside the program can be run along with the expected result. Conformiq then analyzes the model, and creates a lot of tests that covers different combinations and aspects of the model. Conformiq can also generate test cases from non-deterministic behaviours which mean that test cases can be generated from state chart diagrams that do not have parameters explicitly set. The tester can specify how big coverage rate there should be. The statechart diagram is in standard UML structure and therefore easy for the test designer to develop [51]. By working with this standard, Conformiq also allows for defining parallelism and concurrency.

To be able to run the test cases, the test developer have to define system adapters. A system adapter is a client that translates the test cases to a format that the SUT

understands. The test developer can virtually define any interface. The test developer can also define different system adapters so that Conformiq can send messages to one interface and listen for responses from another.

When the tests are created they are automatically run against the SUT. With Conformiq it is possible to run numerous test cases in a short time. Conformiq works by sending messages, and then comparing the response to the information that the test designer has stated in the model. If the response is the same as what is expected Conformiq tells the tester that the test case has passed. In addition to functional testing Conformiq can also be used for load, scalability, and stress testing.

## 5.2 Compuware OptimalJ

Compuware OptimalJ is a MDA based development environment. It is used to develop J2EE applications.

In Compuware OptimalJ it is possible to check that models follow accurate syntax. Syntax checks are performed automatically each time code is being generated but can also be done manually by the tester. The model checker gives very detailed constraints on PIM models; most information is about delete rules i.e. what happens if certain object should be deleted. Some model checking messages can be very informative whilst other found can be very hard to understand, which is likely when the models are of high complexity [52, 53].

Compuware OptimalJ includes a JUnit module for testing. With this module it is possible to open, create, and execute tests. The testing is done for the code that has been generated from the models [53].

## 5.3 Cow Suite

Cow Suite it a test tool that works by generating test cases from the use case and sequence diagrams that describe the system.

It works in combination with Rational Rose to derive test cases [54]. It generates test cases from UML use cases and sequence diagrams. Their test generation algorithm works by exhaustive enumeration of all possible use cases and message sequences. The idea behind Cow Suite is to generate test cases from the use case and sequence diagrams that describe the system.

Cow Suite requires that the use cases be organized in a hierarchy to explicitly define associations between use cases and among actors and use cases. Cow suite does not require that the models are fully developed. It can develop a test plan that is as abstract as the models it is analyzing [55].

Cow Suite is designed to keep the amount of test cases low while still keeping the coverage of the functional areas as wide as possible. It is also designed to allow the tester to interact with it and decide which elements of the UML model that should be tested.

The test cases developed by Cow Suite can be adapted to become the input format of a particular test driver. Cow Suite does not execute the test cases but interacts with the test driver that is chosen.

## 5.4 Eclipse TPTP

Eclipse TPTP is a platform that is used for the whole test and performance lifecycle, from testing in the early stage of the development to application monitoring at a later stage. It includes test edition and execution, monitoring, tracing and profiling, and log analysis capabilities [56]. Eclipse TPTP is not a MDA environment. However MDA tools that support Eclipse can be used inside of it [57].

TPTP includes test tools such as JUnit (see appendix A) based component testing tool, web application testing tool and a manual testing tool. The web application testing tool records user interactions with a browser-based application and makes it possible to edit a test, generate an executable test, and execute the test, and then analyses the test results [58]. Following things can be done with JUnit in Eclipse TPTP [58]:

- Creating a JUnit test.
- Editing a JUnit test.
- Generating Java code for a JUnit test.
- Running a JUnit test.
- Analysis of JUnit test results.

## 5.5 I-Logix Rhapsody

I-Logix Rhapsody is a Model Driven Development (MDD) environment (does not fulfil all the standards set by OMG). It is not only used for developing systems but also for testing them. By the use of I-Logix Rhapsody users can create models, validate them, and generate code from them.

In I-Logix Rhapsody test cases are defined as sequence diagrams. A sequence diagram is developed to express how the SUT should behave. In the Sequence diagram the test designer defines what messages and signals the SUT should receive, and what messages and signals the SUT should send. The test case can then be executed to see if the system really behaves as described. The test case can either generate pass, fail, not active, and active. If a test case fails, I-Logix Rhapsody tells how many percent of it that passed, and also offers to show graphically which messages and signals that failed.

In I-Logix Rhapsody it is possible to generate sequence diagrams when executing the design model. These sequence diagrams can then be compared to the expected ones [43]. If the generated sequence diagram differs from the manually defined Rhapsody tells the tester that an error has occurred and points out were the differences occur.
Test cases can be generated automatically by I-Logix Rhapsody Automatic Test Generator (ATG) to cover the different states in the models**.** It generates test cases to perform structural testing. ATG analyzes UML models and the generated C++ code to automatically generate sets of model-level test cases with high coverage of the design [38]. ATG can also like Conformiq generate test cases from non-deterministic

behaviour. The test cases are saved as sequence diagrams. These sequence diagrams can later be used in Test Conductor. Test Conductor is a tool inside I-Logix Rhapsody that allows testers to test the product developed against its requirements throughout the development cycle.

In I-Logix Rhapsody syntax checks are performed automatically. The user is alerted immediately as soon as a syntax error is performed. It is also possible to run the syntax check manually for selected parts or the entire models. Syntax checks are also done before generating code to see that the models follow accurate syntax and that the models contain everything needed to generate code.

In I-Logix Rhapsody the tester can also easily follow the system behaviour as the model is executed. When the system is being executed it is possible to see with state chart and activity diagrams what state the system is in. It is also possible to run a scenario to check if the system behaves as specified. If a particular model execution run does not conform to the sequence diagram specification (scenario), Test Conductor will report an error. Injecting event stimulus is also possible to change transition between one use case and another [43]. The execution can be done both on the host computer and at a real target environment through TCP/IP.

In I-Logix Rhapsody protocol testing can be performed by the use of sequence diagrams and by execution of the design model.

## 5.6 Pathfinder PathMATE

Pathfinder PathMATE makes it possible for developers to execute their design model at an early stage before coding or code generation begins. Pathfinder PathMATE integrates with Rational Rose and Rational Software Architect. It also has Eclipse 3.0 plugin architecture.

It works by transforming the PIM to C, C++ or Java software. PathMATE then executes the models, and lets the developers see what happens inside the system while running. In Pathfinder PathMATE small elements as well as whole systems can be tested [48].

By the use of Pathfinder PathMATE a tester can execute, and debug the PIM model in both the host and target environment. It has no editing capabilities or UML display but makes use of the editing environment used for development.

When using Pathfinder PathMATE the tester can insert break and trace points so that the program stops executing and collects information. These can later be saved in a driver file. The tester can also step through different events manually and stop the execution whenever it is appropriate.

When using Pathfinder PathMATE the tester can turn animation on to see were in the active state the system is while it is executing. The tester can choose both the appropriate speed of the animation as well as which classes to animate. In Pathfinder PathMATE the tester can assert/inject signals, messages, and interrogate and modify system parameters and variables to see what happens. Pathfinder PathMATE can therefore be used for protocol testing. It can also run different test scenarios created by

the tester. Finally Pathfinder PathMATE can also generate sequence diagrams when the system is executed.

## 5.7 Rational Rose RealTime

Rational Rose RealTime is a UML development environment that lets users execute their models to detect errors.

When the tester executes the system in Rational Rose RealTime, triggered transitions, active states in the state diagram monitors, and dynamic structures animated in the structure monitor can be seen. In addition, probes can be used to trace the messages being passed inside the system [59].

## 5.8 Rational Software Architect

Rational Software Architect (RSA) is an MDD environment that is integrated with Eclipse (see section 5.4). It allows users to develop models, generate code from the models, and then test the code.

With the included JUnit, RSA makes it possible for the test designer to write Java test cases. This is done easily by creating a new test suite from inside of RSA. Test suites are then run to unit and component test the generated code. RSA includes a convenient wizard that assists the tester to create, and edit test suites. The test suite can only be created if the Java project is set to test perspective [60]. To run the test cases Java code must be generated from the models.

RSA includes a function to validate the model for potential transformation problems that can cause compilation errors in generated Java code. Examples of errors like this are: multiple inheritance, circular generalizations, and naming conflicts.

RSA also includes a debugger that lets the developer run the application, and observe were in the code the execution pointer is. Setting break points and suspending launched programs is also possible.

## 5.9 Telelogic TAU G2

Telelogic TAU G2 is a MDD environment that is not only used for developing systems but also for testing them [61]. It has functions to both validate the models, and test the generated code.

In Telelogic TAU G2 a test designer defines a test case by U2TP (see chapter 4.6), and then gets notified if the system fails the test case [62]. The test case can be described by defining in the sequence diagram what the test case looks like. An example of this is when the test designer has a test component and the SUT. The test designer describes what message the test component sends to the SUT, and what response he/she expects back. If the response does not comply with what is expected an error is generated. It is also possible to include a variety of symbols like e.g. timers and verdicts. This method of testing can be done throughout the development cycle.

Syntax checks are performed automatically in Telelogic TAU G2. As soon as a user performs a syntax error, Telelogic TAU G2 alerts him/her of encountered problems. The user can do a more in depth check manually at any time by pressing the check button. Finally syntax checks are also done before generating code. These checks are different though as they look at the limitations the code generator might have in respect to the UML model.

In Telelogic TAU G2 it is possible to execute the system, and watch through a visual debugger what happens inside. Messages and signals can be sent to the system so that the tester can observe what happens inside. It is also possible to go through the system bit by bit as well as insert brake and trace points to observe a special location inside. Scenarios can be generated from a trace point in the form of sequence diagrams and can then be compared to an expected scenario by the compare/merge feature. According to a product specialist at Telelogic the sequence diagrams generated can also be used for regression tests and to specify larger tests**.**

To test the generated code TTCN-3 test cases can be executed. When using the TTCN-3 test environment test cases are written in code and run against the SUT (function and system tests).

In Telelogic TAU G2, protocol tests can be performed by the use of U2TP, execution of the design model, and by writing and executing TTCN-3 test cases.

## 5.10 TITAN

TITAN is an in-house developed TTCN-3 environment that Ericsson uses to program test cases, compile them, and execute them [70]. It is commonly used by Ericsson to function and system test software.

In TITAN the designers build TTCN-3 test suites. These are then compiled to C++ code before they are executed. The test executor works both in single and parallel mode. TITAN lets designers import ASN.1 documents, which then can be used to build skeletons from so the designer only needs to fill in parts of the test cases reflecting the document. It also has a function to syntax check the test cases.

The main part of TTCN-3 and TITAN is the use of test ports. Test ports provide connection between the test suites and the SUT. As writing test ports requires strong programming skills, TITAN can generate test port skeleton for each TTCN-3 module that contains a reference to a test port. TITAN also allows for easy connection to ports already defined.

The executable test suite consists of many components but the main ones are [70]:

- Compiler. The compiler translates TTCN-3 and ASN.1 modules to C++ code.
- Base library, is written in C++ code and contains supplementary functions for the generated code.
- Test Ports, facilitate the communication between the SUT and the test system.

## 5.11 T-Vec Requirements-based Automated Verification (RAVE)

T-Vec RAVE is a test tool that generates test cases from models that specify the requirements of the SUT.

The models are analyzed and used to generate the optimal set of test cases needed to fully test the system. The test cases are then transformed into test drivers for any programming language or test scripts i.e. TTCN-3. These test cases can then be executed in the preferred test execution tool i.e. TITAN (see previous section).

The T-Vec RAVE tool consists of four components:

1. T-VEC Tabular Modeler (TTM). A modelling tool used for requirement capture, management, and defect analysis.
2. Test Vector Generator. Used for functional test case design.
3. Test Driver Generator. Used to transforming generic test vectors into test drivers or test scripts for execution.
4. Report generator. That indicates test coverage, model defects, test cases etc.

## 5.12 Summary

There are different ways to test products developed in an MDA compatible way. Possible ways are to validate the design models, execute the design models, test the code that has been generated from the design models, and generate test cases from either test models or the design models.

The tools that have been presented in this chapter belong to 4 different categories:

- **Tools that validate models**
  - I-Logix Rhapsody
  - Telelogic TAU G2.

- **Tools that test the code that has been generated from the design models**
  - OptimalJ
  - Eclipse TPTP
  - RSA
  - Titan
  - Telelogic TAU G2.

- **Tools that execute design models and display what happens inside them**
  - Pathfinder PathMATE
  - I-Logix Rhapsody
  - Telelogic TAU G2.

- **Tools that generate test cases from models**
  - Conformiq
  - Cow SUITE
  - I-Logix Rhapsody
  - T-Vec RAVE.

# 6 THE EVALUATION OF TEST TOOLS

This chapter describes the whole process from developing criteria to evaluating the tools on how many criteria they support. In the following sections the case study design, case study operation, selection of subjects, questionnaires, execution of case study, threats to validity and the analysis of results of the case study are explained.

## 6.1 Case study design

A case study is usually performed by collecting data for the specific purpose. Statistical analyses can be carried out with the collected data. A case study normally aims to track an attribute or relationship between several attributes [8].

In the following sub-sections the case study operation, the selection of subjects and the questionnaires are described.

### 6.1.1 Case study operation

Criteria were developed by identifying important aspects of MDA. Criteria were also developed based on how software at Ericsson is developed. After the criteria had been developed, it was important to get them prioritized. Questionnaires were designed to serve as instruments for criteria prioritization.

### 6.1.2 Selection of subjects

The selection of subjects connects to the generalisation of the result. It is important that the selection of subject is representative for the population [8]. The subjects in this case were designers, technical coordinators and testers.

As the goal for this study is to find appropriate methods and tools to test an MDA application currently being developed at Ericsson we thought it would be very important to give questionnaires to all the project members. The reason for this is that the opinions of designers, testers and technical coordinators might differ greatly. It is important to capture all the different views and use that data to find an optimal test tool.

Two additional subjects that are not included in the project have also been chosen to take part in the study. They are working as testers and have sufficient knowledge of the MDA application being developed.

### 6.1.3 Questionnaires

Questionnaires in general are the most common instruments to collect data; they can be sent either in paper form, or as in our case, electronically. The most common way to collect data from questionnaires is to have an introduction page which illustrates how to fill it out [8]

The questionnaires have been used to serve as instruments to get criteria prioritized by the manually respondents. The questionnaire used in this study consists of 26 criteria grouped into 6 groups (see section 6.3.1). Each questionnaire comes with an information page (see appendix B) that explains to the respondents how the questionnaire is built and what method they should use to evaluate the criteria. There are also 4 examples in the information page about how a valid measurement could look like. The last example is a picture of the groups with points that add up to 100 points and criteria inside the group that add up to 100 points.

The questionnaires are electronic Excel documents that include a counter that shows how many points that are left to distribute among the groups. Two additional counters are included in the Excel document that show how many points that have been distributed inside the group and how many points that are left.

At the end of the questionnaire there are two background questions that the respondents were asked to fill in. The questions concerned the respondents' roles in the project and also how much knowledge of MDA that they possess (see appendix B).

The reason for developing and distributing the questionnaire electronically is that in our pilot study (see section 6.1.4.1) it was found that it was hard to keep track of points left to distribute. It was also easier to email it to the respondents than locating them physically. Finally we believed that the respondents would be more willing to respond and send it back if everything could be done from their computers.

#### 6.1.3.1 Prioritization of criteria

There are different techniques available to prioritize criteria. The technique that has been used in this study is the Hundred-Dollar test [64]. The Hundred-Dollar test technique was slightly modified to be used in this case study. Instead of dollars, points were used. This modification was done so that the method would be more suitable to the questionnaire that was distributed to the respondents.

The reason why the Hundred-Dollar test technique was used in this case study is because it is well suited when there are many criteria. It also does not require too much time and effort from the participants. The Hundred-Dollar test can also be used to get more views on the criteria by, for example, also letting the participants prioritize the different groups that the criteria belong to.

Other techniques also possible to use for prioritization beside the Hundred-Dollar test, are the Analytical Hierarchy Process and the Planning game. They are described in the following sections.

##### 6.1.3.1.1 The Hundred-Dollar Test

The Hundred-Dollar test works by giving each participant $100 (or 100 points in our case) of imaginary money which they are free to spend on the criteria [64]. The biggest amount of money is spent on the criterion which is most valuable to the participant; the second biggest amount is used on the second most critical criterion and so on. After all of the participants have given their votes, the votes can be summarized from the least important to the most important.

There are a couple of different ways to implement the hundred-dollar test. The usual $100 test technique (the one described above) is good to use when the criteria are similar.

| Criteria | $100 |
|---|---|
| A | $5 |
| B | $12 |
| C | $8 |
| D | $21 |
| E | $7 |
| F | $11 |
| G | $2 |
| H | $25 |
| I | $5 |
| J | $4 |

**Table 6.1** Example of a Hundred-Dollar Test table

In the case of criteria that are better suited when grouped into different categories, it can be interesting to know which category is perceived by the participants as the most valued one. This can be done by first letting them give their $100 to the categories and then to the criteria in each category. An example is shown in Table 6.2.

| Group<br>Criteria | Points | Total Points |
|---|---|---|
| **Group 1** | **20** | |
| Criteria A | 30 | 500 |
| Criteria B | 70 | 1400 |
| **Group 2** | **35** | |
| Criteria C | 20 | 700 |
| Criteria D | 35 | 1225 |
| Criteria E | 45 | 1575 |
| **Group 3** | **45** | |
| Criteria F | 10 | 450 |
| Criteria G | 20 | 900 |
| Criteria H | 5 | 225 |
| Criteria I | 25 | 1125 |
| Criteria J | 40 | 1800 |

**Table 6.2** Hundred-Dollar Test table where criteria are grouped into categories

In Table 6.2 it is illustrated what the total point of each criterion is. The total point is derived from the group point multiplied with the criterion point.

To find the most critical criterion, the criteria points are multiplied with the category point thus obtaining the total points for each criterion. Although criterion B got the highest point it will not be the most valued because its category point is not high

enough, totally 70*20 = 1400. Criterion J will be most valued because of the high criterion and category point, totally 40*45 = 1800.

#### 6.1.3.1.2 Analytical Hierarchy Process

Analytical Hierarchy Process (AHP) is a multi-criteria decision technique where qualitative factors are the most important ones [65]. The first step in AHP is to decide the relative importance between the criteria. This is done by comparing each pair of objectives and ranking them on the following scale: Comparing objective *i* and objective *j* (where *i* is assumed to be at least as important as *j*), give a value $a_{ij}$ as Table 5.3:

| Numerical Value | Verbal Scale |
|---|---|
| 1 | Objectives i and j are of equal importance |
| 3 | Objective i is weakly more important than j |
| 5 | Objective i is strongly more important than j |
| 7 | Objective i is very strongly more important than j |
| 9 | Objective i is absolutely more important than j |
| 2,4,6,8 | Intermediate values |
| *1/3* | *Objective j is weakly more important than i* |

**Table 5.3** Pair-wise comparison scale [65]

Next step is to show in a table the grade of importance, $a_{ii} = 1$ ($a_{ii}$ is always set to 1), if $a_{ij} = k$ then $a_{ji} = 1/k$ and this gives us following table.

| Criteria | A | B | C | D |
|---|---|---|---|---|
| **A** | **1** | 1/7 | 1/3 | 1/4 |
| **B** | 7 | **1** | 4 | 3 |
| **C** | 3 | 1/4 | **1** | 1/2 |
| **D** | 4 | 1/3 | 2 | **1** |
| **Column sum** | 15 | 1,72 | 7,33 | 4,75 |

**Table 6.3** Pair-wise rating of selection criteria

The final step in AHP is to calculate an overall weight on each objective and then calculate an average sum of the criteria. That is done by taking each entry and dividing them by the sum of the column it appears in. For example the A,A entry would be 1 / 1+7+3+4 = 0.067

| Criterion | A | B | C | D | Row average |
|---|---|---|---|---|---|
| **A** | 0.067 | 0.083 | 0,046 | 0,053 | 0,062 |
| **B** | 0,466 | 0.580 | 0,546 | 0,632 | 0,556 |
| **C** | 0,200 | 0.144 | 0,136 | 0,105 | 0,146 |
| **D** | 0,266 | 0.193 | 0,272 | 0,210 | 0,235 |
| **Column sum** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

**Table 6.4** Normalized pair-wise rating of selection criteria

Table 6.4 shows that 56% of the objective weight is on objective B, 23,5 % on objective D, 14,5 % on objective C, and 6 % on objective A

A restriction with AHP is that it is difficult to use with many criteria. It will be very time consuming as in a case where there are 25 criteria, there will be $25^2$ comparisons. The AHP technique is best suited in 4 criteria researches. With AHP it is also hard to get the full picture of what one wants. AHP has a consistency index which is possible to use, if consistency occurs (like A>B, B>C, and A<C),

#### 6.1.3.1.3 Planning Game

The Planing Game (PG) technique is used when different groups of people vote on 2 different types of criteria. The stakeholder votes on the criteria while the developer estimates how long time it will take to implement the criteria [66].

The stakeholders votes critical, important and useful for each criteria.

1. **Critical** means that the product is useless without the criteria fulfilled. It will not satisfy the stakeholders' requirements.
2. **Important** means that without the criteria the stakeholders probably will not be satisfied with the product.
3. **Useful** means that the criteria is nice to have, it makes the product more fun.

Several participants voting can end up with all of the criteria being critical. This can be avoided by making each vote have points like: critical 9 points, important 3 points and useful 1 point.
The developers estimate the time to develop a requirement in 3 categories:

1. The estimation can be done precisely.
2. The estimation can be done reasonably.
3. The estimation can not be done at all.

With the points and the estimated time of each criterion the stakeholder can now decide which criteria they want to have planned for the next product release [67].

### 6.1.3.2 Background information

The first question that was asked of the subjects is what role they have in the project. It is important to see how different roles can affect their feelings on what criteria they find important. It is also interesting to see if subjects with similar roles have similar views.

The knowledge level that the subjects consider themselves having on MDA was of great interest to this study. By analyzing how the level of knowledge can influence their choices when evaluating the criteria, patterns can be discovered. Both the similarities subjects with better knowledge of MDA might have and what differences subjects with good and bad knowledge of MDA might have is also of valuable information.

### 6.1.4 Execution of case study

The pilot study was done to validate our design, in particular our questionnaires. Getting feedback on how the questionnaire was constructed was important so that changes for the better could be made before it was sent to all the respondents.

The whole case study took approximately 2 months to conclude. It started at the beginning of October and ended at the beginning of December. The most amount of time was spent on identifying important aspects of MDA and how work is conducted at Ericsson.

The questionnaires were sent to a total of 14 respondents of which 9 responded. The remaining 5 did not consider themselves knowledgeable enough in the area and therefore they asked to be omitted from the case study. The questionnaires were sent to the respondents electronically on the 16$^{th}$ of November 2005. A few days later they were returned. The analysis of the questionnaires took 2 weeks.

#### 6.1.4.1 Pilot study

A copy of the questionnaire was given to a project member whose role is Designer & System manager. When the questionnaire was returned, questions were asked about his opinion of it.

His first feeling was that it was very hard to keep track on how many points that were left. He said that a couple of times the total was more then 100 points and he had to redo the prioritization again. He recommended the questionnaire to be electronically in Excel so that the subjects would know how many points are left to distribute. He also thought that the groups and criteria could be presented in two different windows. He thought it was hard to decide how the points should be distributed as the groups were spread on two pages. Changes to the questionnaire were made based on the information provided by the project member.

#### 6.1.4.2 Main study

When the pilot study was done and the main study was planned the questionnaire was carried out to collect the data that should be analyzed [8]. It is in this part where the criteria get prioritized by the selection of subjects.

The main study was initiated by sending each respondent a copy of the questionnaire and an information page that explained how the measurement should be done. The questionnaire was sent by email that also described who the senders were, the reason why they had received the questionnaire, and also a time limit that they were asked to keep.

The first two questionnaires were returned the day before the time limit ran out. The following 6 questionnaires were returned on the due date. During the time limit, three members of the project explained that they would not be returning the questionnaire as they did not think it would be fair to the result. They did not know enough of either MDA or the project to be able to contribute to the study.

The overall response of the questionnaire was very good. The respondents indicated that they did not have any problems of understanding how to fill it out. There were no misunderstandings of how to grade the different groups or the criteria inside the groups. All questionnaires were returned with all the information that was required.

## 6.2 Threats to validity

It is important that a case study must be valid, the threats to validity must be considered already in the planning phase. Wohlin et al. [8] describes four types of threats to validity, conclusion, internal, construct and external.

### 6.2.1 Conclusion validity

Conclusion validity is about the relationship between the treatment and the outcome. How can we be sure that the conclusions we have made are correct [8]?

> *Threat:* Low statistical power due to low sample size
> *Minimize threat:* There are about 9 respondents in the project that we had the advantage of working near. The tests used in this study rely on sufficient number of criteria and not the size of the subject group. That is why 9 respondents are enough.

> *Threat:* Hard to compare the test tools on equal grounds because they provide different functionality and work in various manners.
> *Minimize threat:* The test tools are grouped together based on their similarities and functions which are derived from appropriate methods.

### 6.2.2 Internal validity

Internal validity exists in a case study if there is a relationship between the treatment and the outcome. The result should not be of a factor which we have no control over or have not measured. The treatment should cause the outcome [8].

> *Threat:* The possible threat is to concentrate on MDA testing in general without having Ericsson's specific issues in mind.

> *Minimize threat:* This threat is minimized by doing the case study at Ericsson and by participating in the meetings of the project group. It is also minimized by having close contact with our supervisor at Ericsson.

### 6.2.3 Construct validity

Construct validity is about relationship between theory and observation [8], in our case it is whether the criteria were correctly developed.

> *Threat:* The criteria are not clear enough for the group in the case study, so they will not be able to correctly prioritize them.
> *Minimize threat:* A pilot study minimizes this threat; feedback is received by giving the questionnaire to one person first.
>
> *Threat:* Criteria are missing or too many criteria exist.
> *Minimize threat:* Unfortunately this threat can not be avoided, but it is minimized by letting our supervisors review the criteria. This threat is also minimized by letting the respondents add criteria they might find important that are not already there.

### 6.2.4 External validity

The external validity is about the generalization. Can the result of this case study be generalised outside the scope of the study [8, 68]?

> *Threat:* The knowledge on MDA at the studied Ericsson organization is relatively low so the case study result can not be compared with another company that has a higher knowledge on MDA.
> *Minimize threat:* The knowledge on MDA at Ericsson can not be affected however a conclusion can be made on which type of companies this case study should not be generalized to.
>
> *Threat:* Miss the up-to-date tools in the research and do the evaluation on old versions of the tools.
> *Minimize threat:* Interviews are performed with people at Ericsson working with methods and tools. We also participated in a test conference to get knowledge about the new tools available in the market. Finally we also contacted vendors and performed a literature study.
>
> *Threat:* The selection of subjects will be too similar.
> *Minimize threat:* We also gave the questionnaire to people outside the project, which are the testers.

## 6.3 Analysis of results

To be able to draw valid conclusions of the collected data, interpretation of the data is needed [8]. In this section we discuss the criteria development, list of criteria, what our expected outcome was and what the actual outcome is.

### 6.3.1 Criteria development

To get an understanding of which criteria are important for Ericsson when evaluating test tools, observations were first done. These observations answered a lot of questions

on their thoughts of the project. The observations kept on until the study was finished. This way of collecting information was very good as new criteria were developed up to the moment that questionnaires were sent. To get more information and clearer opinions, one designer and one tester were interviewed. They both have great experience in testing. The interviews gave clearer information on how they operate today and what they find important.

Criteria were also developed by listening to the questions that a couple of the project members had when a company called Telelogic came to present their MDD tool. The needs of the project members became very clear when questions were asked to Telelogic about their tool

To develop criteria that concerned MDA a literature study was made. The outcome of this study was that there are four MDA specific features that are important to include as criteria.

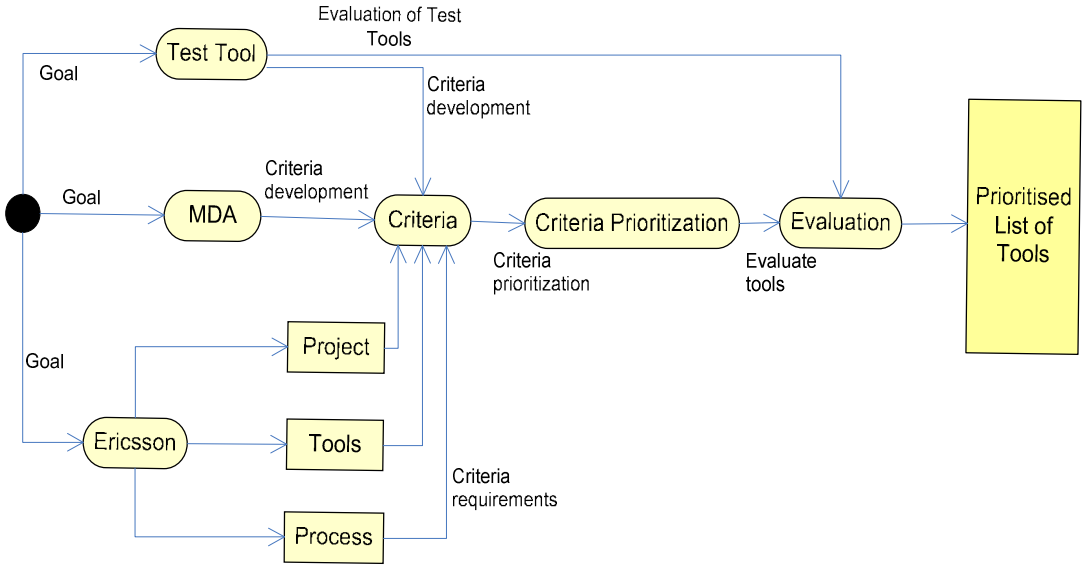



**Figure 6.1** Criteria development

In Figure 6.1 it is illustrated that criteria were developed by studying test tools, MDA and Ericsson. When all the criteria were put together and the prioritization of the criteria was done by the subject group the evaluation of the tools based on which criteria they support started. That resulted in a prioritised list of tools.

#### 6.3.1.1 List of criteria

The criteria were grouped into six groups. They have been grouped together mainly on basis of their similarities. They have also been grouped together depending on which categories they belong to. For example all criteria that are MDA specific are grouped together in the MDA group. Finally criteria have also been grouped on basis of what functions they serve. The criteria and their groupings are illustrated in Table 6.5.

| Criteria No. | Criteria |
|---|---|
| | **Test Case Generation** |
| 1 | Generate TTCN-3 test cases |
| 2 | Automatic generation of test cases from design models |
| 3 | Automatic generation of test cases from test models |
| | **Integration** |
| 4 | Integration with Rational Software Architect (RSA) |
| 5 | Import of models from Rational Rose |
| 6 | Integration with Requisite Pro |
| 7 | Integration with Rational ClearCase UCM for configuration management |
| 8 | Integration with Rational ClearCase base for version control |
| 9 | Integration with the Eclipse platform |
| | **Usage** |
| 10 | Good and easy to understand report of test execution/generation result |
| 11 | Generating report of test execution/generation in a custom format |
| 12 | Good documentation |
| 13 | Good support |
| 14 | Support ports already defined |
| 15 | Support the current process |
| | **Models** |
| 16 | Use of models to test the system |
| 17 | Syntax testing of the design models |
| 18 | Software inspections |

| Criteria No. | Criteria |
|---|---|
| | **MDA** |
| 19 | Platform test on PIM |
| 20 | Analysis of PIM to PSM |
| 21 | Architecture analysis of Models |
| 22 | PSM assumptions of the platform |
| | **Testing** |
| 23 | Visual runtime analysis of the system |
| 24 | Test according to XML schema message and boundary |
| 25 | Use of stubs to basic test |
| 26 | Use of code to test the system |

**Table 6.5** Table of criteria

### 6.3.2 Overall result of criteria prioritization

Six designers as well as one technical coordinator in the project group have responded to the questionnaire. From outside of the project group two testers have answered. The results of the background questions are that the member of the project group's overall knowledge of MDA is 2.71. The overall knowledge of the testers (respondents outside the project group) is 2 out of 5. The number 5 means that the respondents believe they have very good knowledge of MDA while 1 means they have insufficient knowledge of MDA.

In Table 6.6, it is illustrated how many points each group (Test Case Generation, Integration, Usage, Models, MDA, and Testing) obtained. It also illustrates how many total points each criterion obtained. Total points are calculated by multiplying the average point of the criterion and the average point of the group it belongs to. One example is the first criterion inside the Test Case Generation group. The total points that the Generate TTCN-3 criterion obtained is by multiplication of its own points (24,4) and the average points the Test Case Generation group obtained (15,6). So the total points of the generate TTCN-3 test cases criterion is: 24,4 x 15,6 = 380,2.

| Criteria No. | Criteria | Points | Total Points |
|---|---|---|---|
| | **Test Case Generation** | **15,6** | |
| 1 | Generate TTCN-3 test cases | 24,4 | 380 |
| 2 | Automatic generation of test cases from design models | 35,6 | 553 |
| 3 | Automatic generation of test cases from test models | 35,6 | 553 |

| Criteria No. | Criteria | Points | Total Points |
|---|---|---|---|
| | **Integration** | **19,4** | |
| 4 | Integration with Rational Software Architect (RSA) | 16,1 | 313 |
| 5 | Import of models from Rational Rose | 4,4 | 86 |
| 6 | Integration with Requisite pro | 18,7 | 363 |
| 7 | Integration with Rational ClearCase UCM for configuration management | 18,9 | 367 |
| 8 | Integration with Rational ClearCase base for version control | 18,6 | 361 |
| 9 | Integration with the Eclipse platform | 19,2 | 372 |
| | **Usage** | **18,5** | |
| 10 | Good and easy to understand report of test execution/generation result | 21,7 | 401 |
| 11 | Generating report of test execution/generation in a custom format | 13,3 | 247 |
| 12 | Good documentation | 21,1 | 391 |
| 13 | Good support | 18,3 | 339 |
| 14 | Support ports already defined | 15,0 | 278 |
| 15 | Support the current process | 9,4 | 175 |
| | **Models** | **14,4** | |
| 16 | Use of models to test the system | 39,4 | 570 |
| 17 | Syntax testing of the design models | 30,6 | 441 |
| 18 | Software inspections | 26,7 | 385 |
| | **MDA** | **10,6** | |
| 19 | Platform test on PIM | 24,4 | 258 |
| 20 | Analysis of PIM to PSM | 23,3 | 246 |
| 21 | Architecture analysis of Models | 17,8 | 188 |
| 22 | PSM assumptions of the platform | 21 | 223 |
| | **Testing** | **19,4** | |
| 23 | Visual runtime analysis of the system | 35,6 | 691 |
| 24 | Test according to XML schema message and boundary | 19,5 | 379 |
| 25 | Use of stubs to basic test | 21,0 | 408 |
| 26 | Use of code to test the system | 17,2 | 335 |

**Table 6.6** Table of criteria points

Table 6.6 illustrates that the most important and valued criterion is the visual runtime analysis of the system. This criterion obtained 691 points. Second most valued criterion is the use of models to test the system criterion. The least valued criterion is the import of models from Rational Rose which received 86 points, while the second least important value is that the test tool should support the current process.

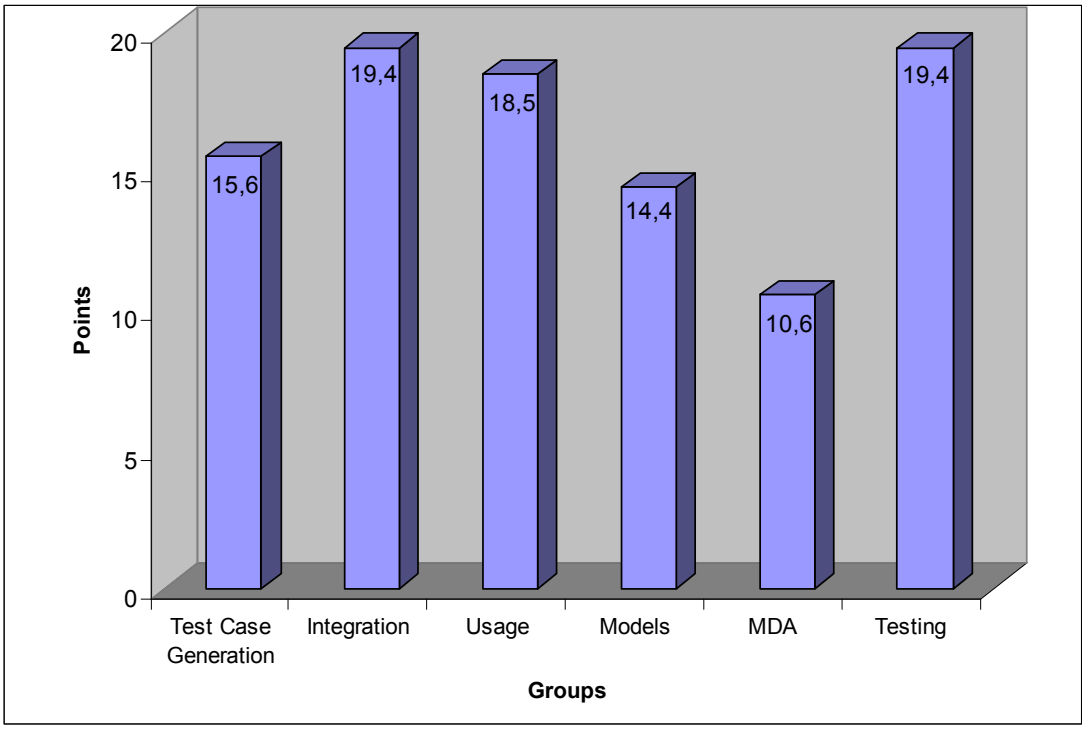



**Figure 6.2** Actual outcome of the criteria groups' prioritization

As illustrated in Table 6.6 and Figure 6.2 the most important groups are the Integration and the Testing group closely followed by the Usage group. It is also very evident that MDA is the group that has obtained the least amount of points. This is probably connected to the overall low amount of knowledge the respondents think they have of MDA.

#### 6.3.2.1 The respondents criteria group prioritization

The subject group consists of 7 project members and two testers outside the project group. The differences in how the two groups (project members and testers) prioritized the criteria can be seen in Figure 6.3

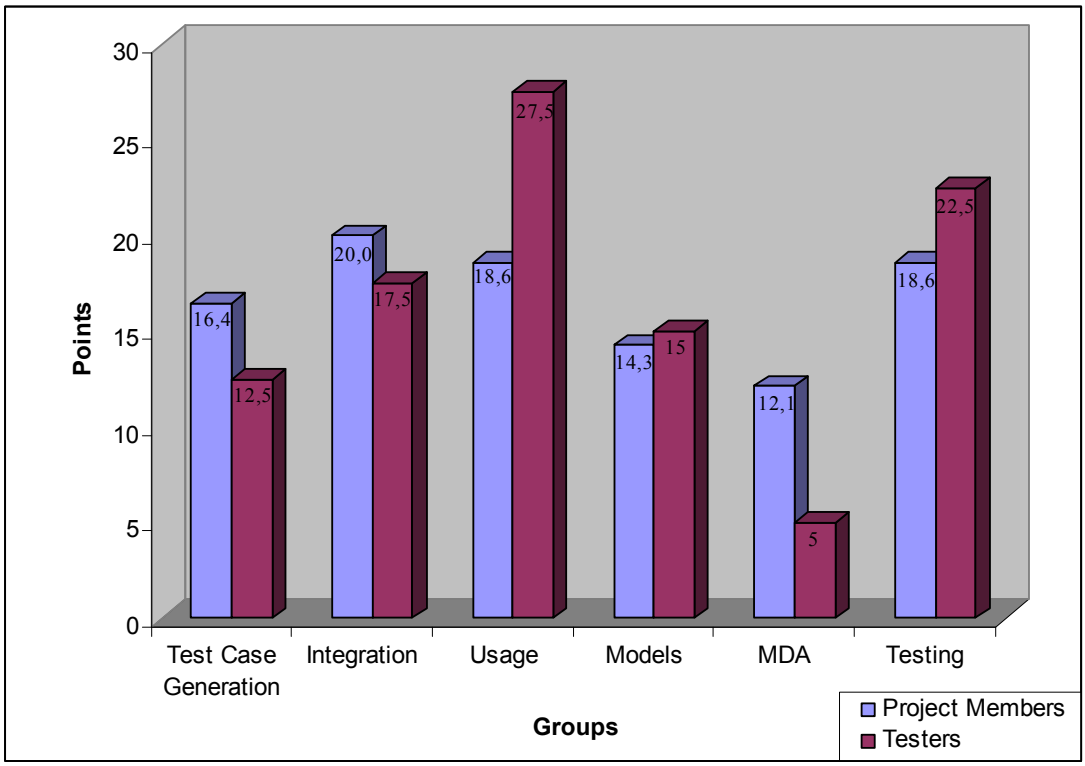



**Figure 6.3** Project members and testers

Figure 6.3 illustrates that the testers valued the criteria that belong to the usage and the testing group very high. It is also very clear that they are not as interested in the criteria that belong to the MDA group as the project members are. The amount of testers included in this case study was very low and therefore not that reliable. It should however be noted that their overall opinions were significantly similar**.**

### 6.3.3 Result of the prioritization of the significant respondents

To find which respondents are significant the Chi-square test has been used. A significance level 0.05 (5%) has been chosen. By knowing who the significant respondents are, the criteria prioritizations of the deviant respondents can be removed from further analysis. This will result in a more even prioritization of criteria thus raising the credibility of the research 9 respondents responded to the questionnaire which means that there are 36 different pairs to compare. The chi-square calculation of the pairs resulted in 27 significant pairs and 9 non-significant pairs. A significant pair in this thesis is two respondents that have less than or equal to 5% difference in their criteria prioritization. There were 14 significant respondents that were mixed (a mixed pair can be a significant project member and tester), 12 significant project member pairs and 1 significant tester pair (there are only two testers in this case study) see appendix E. One problem that had to be solved was when person 1 was significant with person 2 and person 2 was significant with person 3 while person 1 and person 3

were not significant. To solve this problem the average values from pair 1 (person 1 and 2) and pair 2 (person 2 and 3) were calculated. Then the average value for each groups were calculated for both pair 1 and 2, and at last an overall average were calculated for all the significant pairs (see Table c.1 and c.2 in appendix C).

Chi square is a family of a non-parametric test of statistical significance for tabular analysis. Non-parametric tests do not make the assumption about distributions and parameters, only general assumptions are made to derive non-parametric tests. The significance level depends on how much that is willing to accept that the two samples are wrong in generalization from the population it represents [66, 69].

The calculation of the chi-square and significance level of one pair was done in a web based calculator [69], the calculator creates a table based on the dimensions that have been given by the user. Next step is to fill out the cells with the data. The calculator then calculates the chi-square for the table and the significance level between the two respondents in the pair. The significance level was set to 0.05. The non-significant pairs are the respondents whose points totally differ more than 5 % in significance.

Then the total value for each row and column and the degree of freedom was calculated (which is used to find the significance level). The chi-square and significance level was then calculated. The significance level is represented as the p-value. All the information about the significant pairs and the significance level between them can be found in appendix E.

#### 6.3.3.1 Comparison between the significant respondents and all respondents

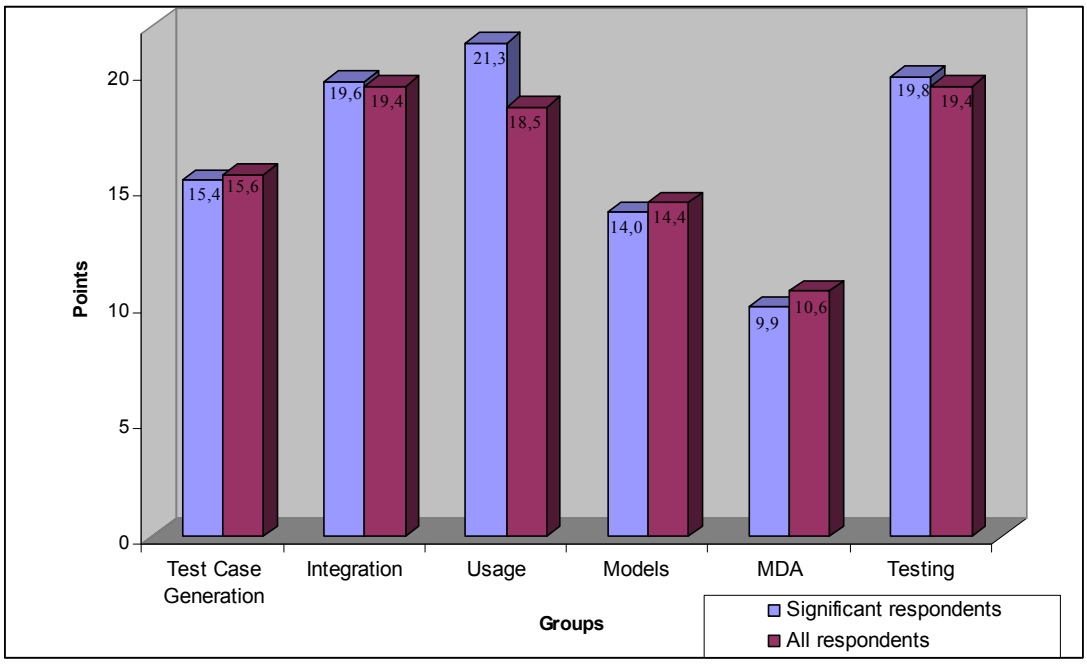



**Figure 6.4** Comparison between the significant respondents and all respondents

Figure 6.4 illustrates that the usage group was the highest prioritized group by the significant respondents. The test case generation group has been prioritized almost the same by both respondent groups. The significant respondents' model and MDA groups were less prioritized compared to all the respondents (see Figure 6.4) while MDA is still the least important group.

#### 6.3.3.2 Comparison between the significant project members and all project member

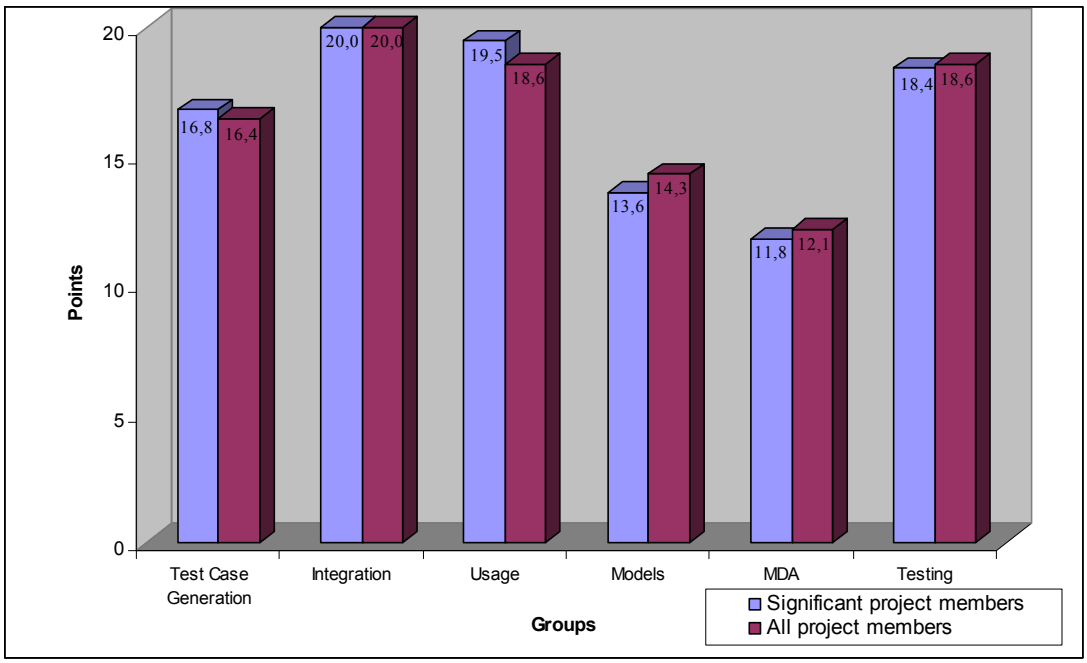



**Figure 6.5** Comparison between the significant project members and all designers

In Figure 6.5 the differences between the significant project members and all of the project member are illustrated. Note that integration is exactly the same prioritized by the significant project members. The usage is 2nd most valued group in relation to the significant project members and all project members. Testing is the 3rd most important group for the significant project members. The significant project members prioritized the model group quite equal to all the project members. The significant project members have also prioritized MDA almost as low as all of the designers.

#### 6.3.3.3 Comparison between significant designers and significant testers

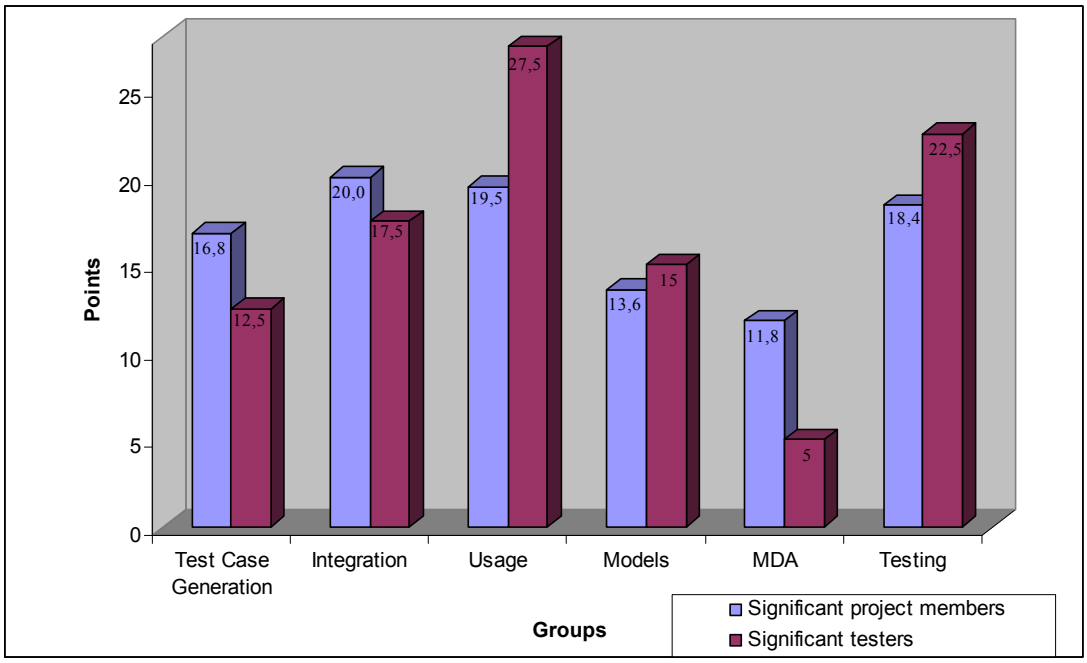



**Figure 6.6** Comparison between significant project members and significant testers

Figure 6.6 illustrates the comparison between the significant project members and testers. It is shown that the prioritization has been performed differently by the significant project members and the significant testers. The usage group is the most important group for the testers while it is 2nd most important for significant project members. It is also illustrated how big differences there are between the significant designers and testers when prioritizing the test case generation, MDA, and testing groups.

### 6.3.4 Discussion on the criteria prioritization

The fact that the visual runtime analysis of the system criterion obtained the most amount of points shows that this is one function that the project group would really like to be able to use in a test tool. The possibility to execute design models and be able to debug it by sending messages and signals to it is of great interest to the project group. It also shows that a great deal of testing can be made by executing the design model.

The use of models to test the system criterion was also very high valued by the project group. It shows the project group's intention of not only designing the system by models but also testing it by them.

An interesting fact is that Rational Rose is not a modeling tool that the project group expects to use in the future, thus, the criterion that a test tool should be able to import models from Rational Rose received the lowest points. Also the fact that the criterion concerning that the test tool should support the current process received a very low point shows that the project group is more then willing to change the current process if it results in better software. It also shows that they rather adapt the process to the tools and not the opposite.

It is clear that test case generation is something that the project group would be very interested in achieving. Both the test case generation from design models and the test case generation from test models obtained the same amount of points. The thought is clearly that if test case generation is possible then the system will be tested more thoroughly which will result in more problems being found and corrected.

Finally it is also very clear that the amount of points that the Integration group obtained shows that it is very important that the test tool can integrate with the most important and crucial tools currently being used at Ericsson especially Rational ClearCase.

### 6.3.5 Comparison of test tools

In this section each test tool has been prioritized in relation to which criteria they support. That is, they have been assigned points depending on which criteria they support. In the following subsections the test tools have been divided into groups according to their similarities and functions. Comparisons have also been made between the tools inside of each group.

The tools that have obtained the most points are the MDD tools. I-Logix Rhapsody is the tool inside of this group that has obtained the most amounts of points. I-Logix Rhapsody is closely followed by Pathfinder PathMATE which is a program that integrates with RSA. The third highest points went to Telelogic TAU G2 which is also an MDD tool (see Figure 6.7).

The test tools are listed in Table 6.7, the tool with most points first and the tool with the least amount of points last. At the bottom are the test case generation tools Cow Suite with only 814 points and Conformiq with 1375 points (see Figure 6.7).

| **Criteria No.** | Criteria |
|---|---|
| I-Logix Rhapsody | 2, 5, 6, 8, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25 |
| Pathfinder PathMATE | 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19, 22, 23, 25, 26 |
| Telelogic TAU G2 | 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 23, 24, 25, 26 |
| Rational Software Architect | 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17 , 18, 19, 22, 25, 26 |
| Compuware OptimalJ | 5, 8, 9, 10, 12, 13, 15, 17, 18, 19, 20, 21, 22, 25, 26 |

| Criteria No. | Criteria |
|---|---|
| Rational Rose Realtime | 5, 6, 7, 8, 10, 12, 13, 17, 18, 23 |
| Eclipse TPTP | 4, 6, 7, 8, 9, 10, 12, 13, 15, 25, 26 |
| T-Vec RAVE | 1, 3, 8, 10, 11, 12, 13, 15, 16 |
| TITAN | 8, 10, 12, 13, 14, 15, 24, 25, 26 |
| Conformiq | 3, 10, 11, 15 |
| Cow Suite | 2, 5, 15 |

**Table 6.7** Comparison of test tools, criteria supported

Table 6.7 illustrates how large difference there is in criteria support for the different tools.

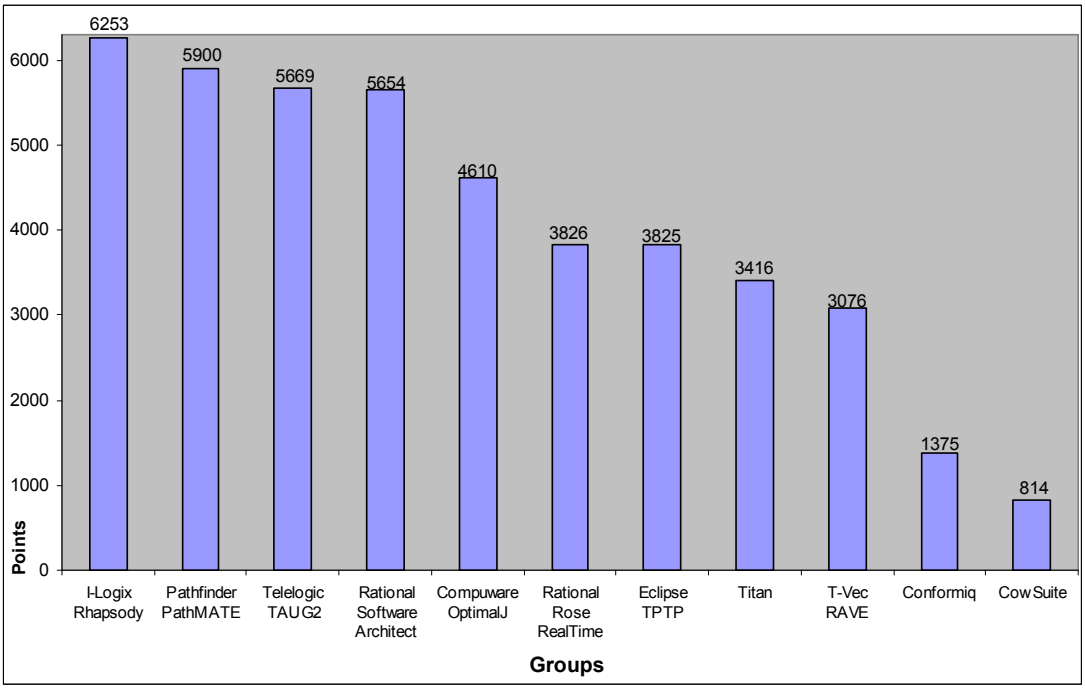



**Figure 6.7** Comparison of test tools, total points

In Figure 6.7 it is illustrated how many points the test tools obtained. It is very evident that the 5 MDD tools included in the evaluation are the test tools that have obtained the most amounts of points. The reason for this is that they offer more MDA related features and thus support more criteria. It is also evident that the test case generation tools are the tools that obtained the least amount of points as their main use is to generate test cases and therefore do not offer the same amount of features as the others.

#### 6.3.5.1 Comparison of test tools that validate models

The test tools that belong to this group are I-Logix Rhapsody and Telelogic TAU G2. The amount of points and their relation to each other is seen in Figure 6.8

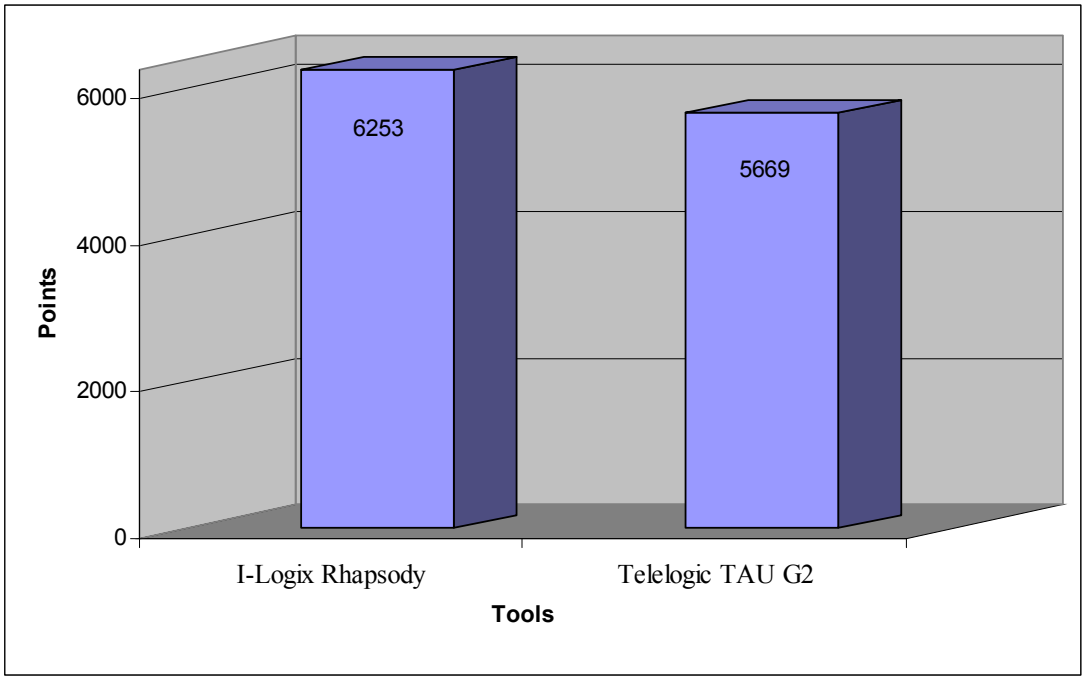



**Figure 6.8** Test tools that validate models

As illustrated in Figure 6.8, I-Logix Rhapsody is the MDD tool that has obtained the most amounts of points in this group mainly due to its extra features, like the ability to generate test cases from design models. It is evident that the two different model validation tools obtained similar mount of points.

#### 6.3.5.2 Comparison of test tools that test code

The test tools that belong to this group are: Telelogic TAU G2, RSA, Compuware OptimalJ, Eclipse TPTP, and Titan. The amount of points that the test tools have obtained and their relation to each other can be seen in Figure 6.9

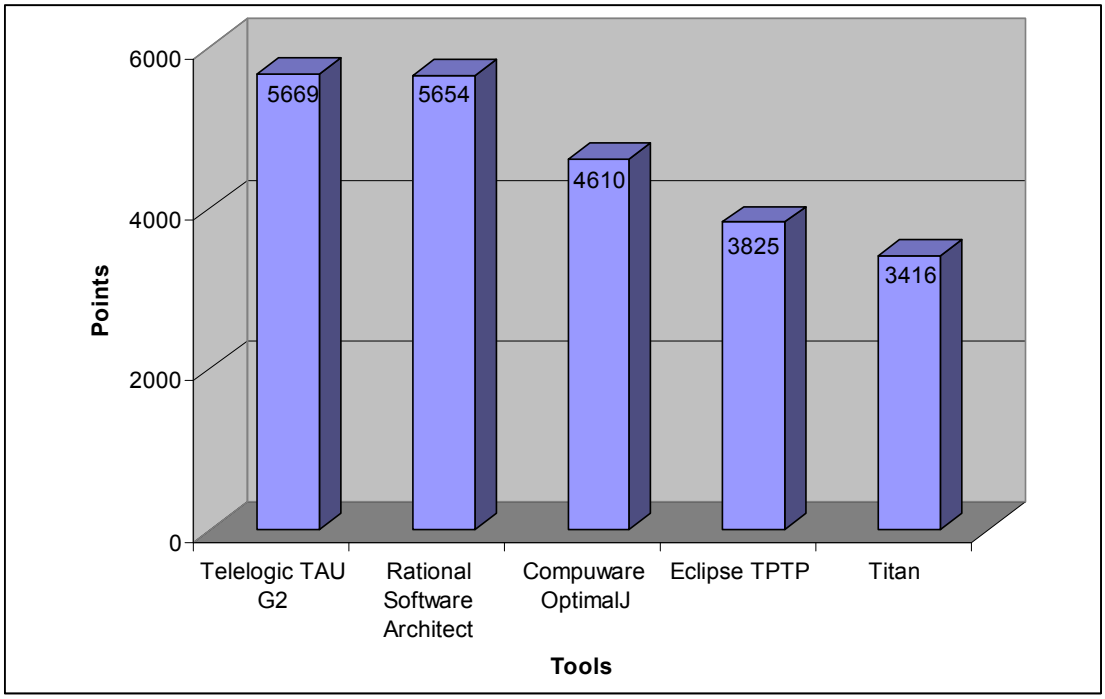



**Figure 6.9** Test tools that test the code that has been generated from design models

As illustrated in Figure 6.9, Telelogic TAU G2 is the MDD tool that has obtained most points in this group. It has a TTCN-3 test module integrated that can be used to test the code that has been generated from the models. TITAN and Eclipse TPTP are tools that are not MDD based and therefore they have obtained fewer amounts of points. They are only used for testing the code that have been generated from models and have obtained equal amounts of points.

#### 6.3.5.3 Comparison of test tools that execute design models

The four tools that are able to execute design models are: I-Logix Rhapsody, Pathfinder PathMATE, Telelogic TAU G2, and Rational Rose RealTime. The amount of points they have obtained and their relation to each other can be seen in Figure 6.10.

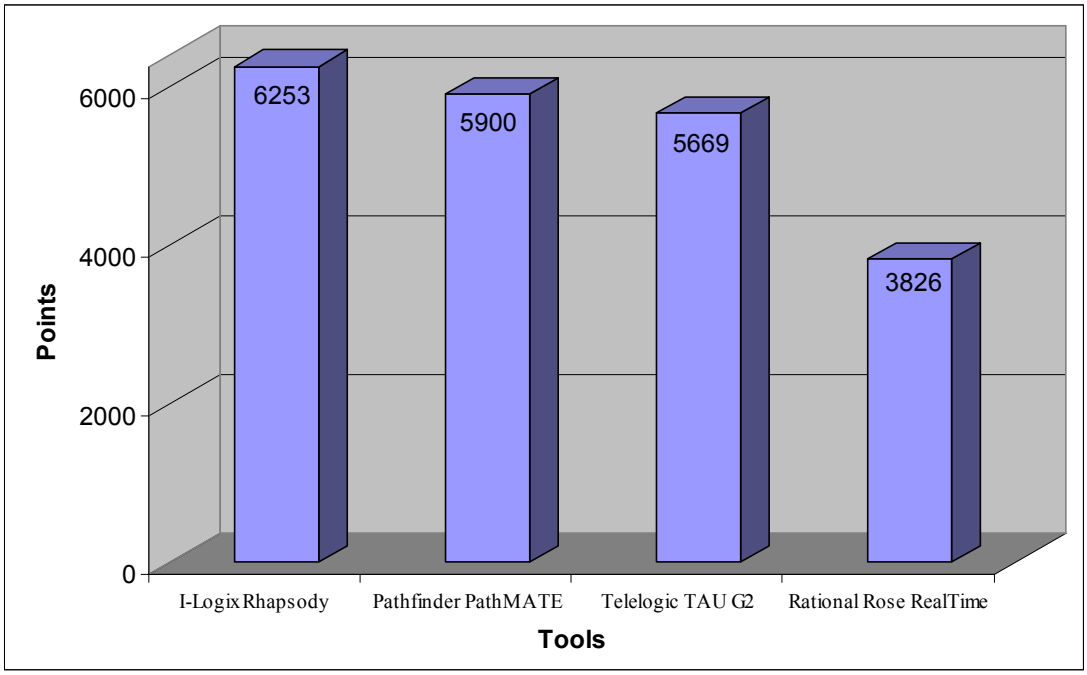



**Figure 6.10** Test tools that execute design models and show what happens inside them

As illustrated in Figure 6.10, there are four tools able to execute the design models and of those I-Logix Rhapsody has obtained the most amounts of points. Pathfinder PathMATE is second, Telelogic TAU G2 is third while Rose RealTime is last. The differences in points that the 4 tools have obtained is mainly due to their integration support.

#### 6.3.5.4 Comparison of test tools that generate test cases from models

There are three tools included in this evaluation whose overall function is to generate test cases from models. As they are tools that are only used to generate test cases their overall points are less then the others. T-Vec RAVE is the tool that has obtained the most amounts of points in this group.

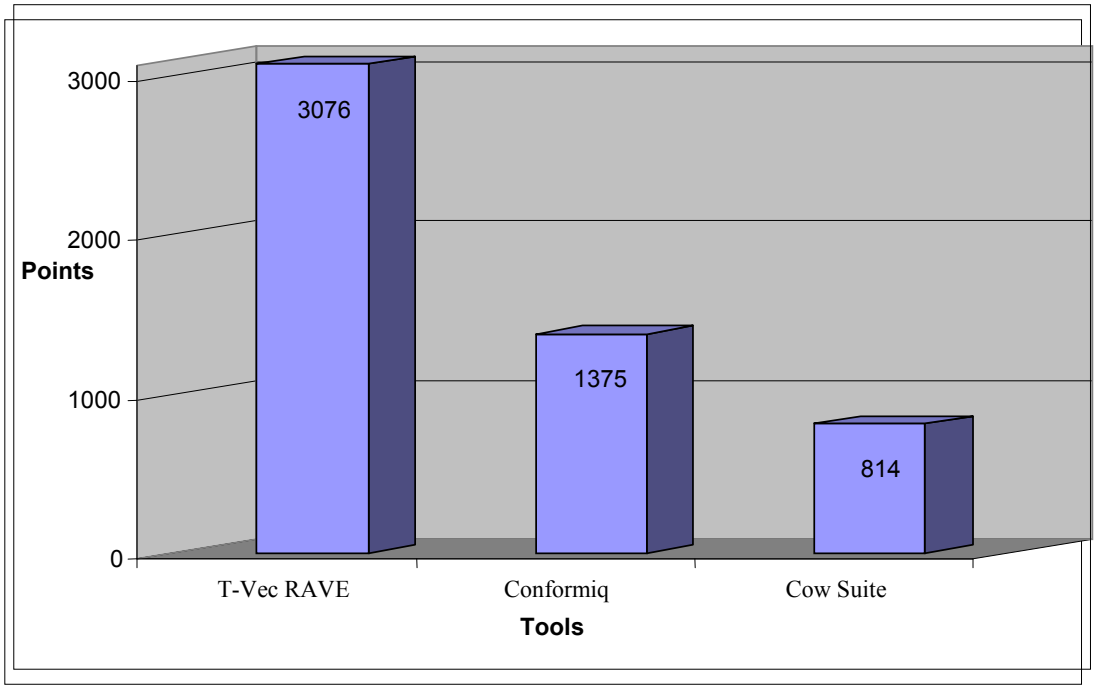



**Figure 6.11** Test tools that generate test cases from models

As illustrated in Figure 6.11, T-Vec RAVE which generates test cases in any custom programming language from models that specify the requirements posed on the system, obtained most amount of points. On second place is Conformiq which generates test cases from a statechart diagram that has been created specifically for testing. Cow Suite is last because it is experimental. It also does not offer any support or documentation.

### 6.3.6 Tool comparison between significant and non-significant respondents

The amount of points that the tools obtained changes when based on the criteria that the significant respondents prioritized (see Figure 6.12) In Table 6.4 it is illustrated how many points each tool obtained when only the significant respondents prioritized the criteria and when the whole group prioritized them. In Table 6.4 it is illustrated that there are differences in how many points each tool obtained. The only difference thus in how many points each tool has obtained in comparison to each other is that Eclipse TPTP now has more points then Rational Rose RealTime. So there is no change in the top 5 test tools in how they compare to each other when based on the significant respondents and all of the respondents.

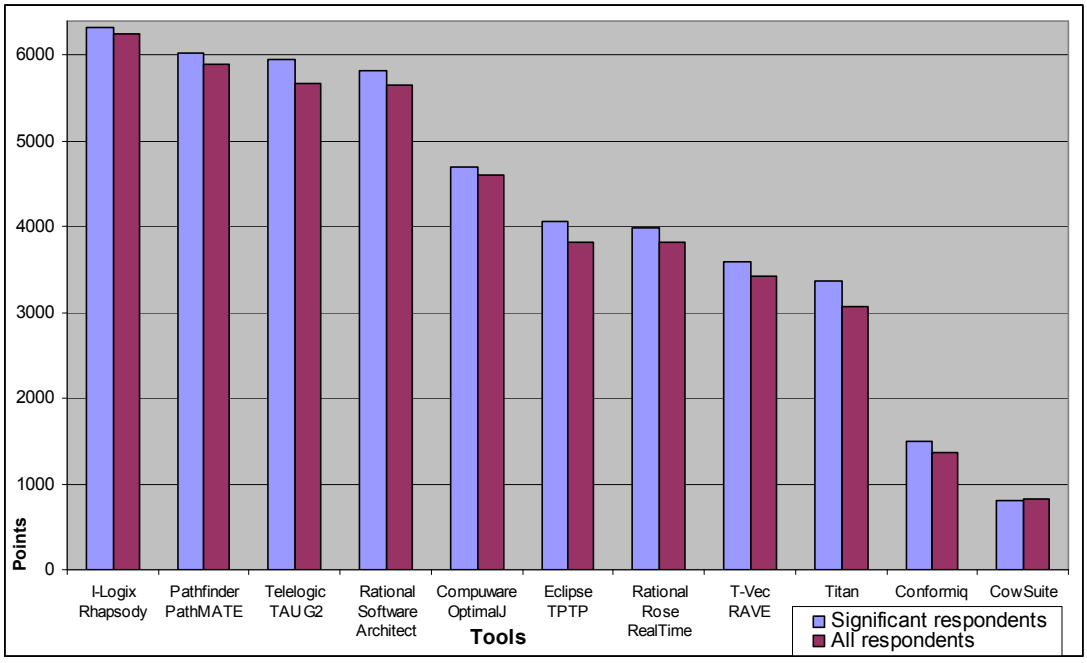



**Figure 6.12** Comparison of test tools with significant respondents

In Figure 6.12 it is illustrated that the only difference in how the tools compare to each other when only using the criteria prioritizations of the significant respondents is that Eclipse TPTP now has more points then Rational Rose RealTime. Except for Eclipse TPTP and Rational Rose RealTime there is no difference in how the tools compare to each other when only based on the criteria prioritizations of the significant respondents. This shows that the respondents have a very uniform view of what is most appropriate for them.

### 6.3.7 Comparison of test tools with future modifications

In some cases developers of the different test tools have offered extra functions to be implemented in the next version. The future modification column in Table 6.8 shows which additional criteria that will be supported in the coming version of the test tools (Future modification column).

| Tool | Criteria supported | Future modification | Total amount of points |
|---|---|---|---|
| Pathfinder PathMATE Spotlight | 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19, 22, 23, 25, 26 | **2** | **6453** |
| I-Logix Rhapsody | 2, 5, 6, 8, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25 | | **6253** |
| Telelogic TAU G2 | 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 23, 24, 25, 26 | **5** | **5755** |
| Rational Software Architect | 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17 , 18, 19, 22, 25, 26 | | **5654** |
| Compuware OptimalJ | 5, 8, 9, 10, 12, 13, 15, 17, 18, 19, 20, 21, 22, 25, 26 | | **4610** |
| Rational Rose Realtime | 5, 6, 7, 8, 10, 12, 13, 17, 18, 23 | | **3826** |
| Eclipse TPTP | 4, 6, 7, 8, 9, 10, 12, 13, 15, 25, 26 | | **3825** |
| T-Vec RAVE | 1, 3, 8, 10, 11, 12, 13, 15, 16 | **6** | **3779** |
| TITAN | 8, 10, 12, 13, 14, 15, 24, 25, 26 | | **3076** |
| Conformiq | 3, 10, 11, 15 | **14, 24** | **2042** |
| Cow Suite | 2, 5, 15 | **12** | **1205** |

**Table 6.8** Comparison of test tools with future modifications

As illustrated in Table 6.8 Pathfinder PathMATE now has taken the lead with 6453 points. The Pathfinder PathMATE developers have offered an extra function to generate test cases from the design models which has significantly raised its overall points. With this function Pathfinder PathMATE and RSA which are used together obtain more points then any of the other MDD development tools. It is also illustrated that both test case generation tools: T-Vec RAVE and Conformiq have greatly raised their overall points with the added usage and integration support.

### 6.3.8 Discussion

When it comes to testing MDD applications, I-Logix Rhapsody is a clear winner. It is the MDD tool that has most functions. The fact that it is possible to work in two views, one for model and one for code is something that is appreciated a great deal. Whenever a change is made in one view the other one is automatically updated. With I-Logix Rhapsody it is also possible to execute the design model on the target location with a TCP/IP connection. We feel that I-Logix Rhapsody is the MDD tool that has the best overall test features. However as it does not support J2EE we do not find it to be the best test tool for Ericsson to use when testing their MDA application.

Pathfinder PathMATE together with RSA are the best selections of tools to use when developing and testing the MDA application currently being developed at Ericsson. The reason for this is because these two tools together is the solution with J2EE support that obtained the largest amount of points. As described in chapter 5.6, Pathfinder PathMATE is designed to work in conjunction with RSA. Pathfinder PathMATE adds features to RSA, like the possibility to execute the design model for protocol testing, to see how it behaves and what happens inside of it. The disadvantage of RSA and Pathfinder PathMATE is that it is not possible to validate models when using those tools. It is also not possible to develop test cases from models i.e. sequence diagrams. There are also no possibilities of generating test cases from the design model.

Telelogic TAU G2 also appears to be a good MDD tool. It is the first tool that officially supports U2TP (see 4.6). One negative aspect with Telelogic TAU G2 is that it is only possible to execute the design model on the host computer and not the target location as with I-Logix Rhapsody and Pathfinder PathMATE. It is also only possible to transform the design model to C code before execution and debugging can be performed. That is trivial compared to Pathfinder PathMATE that can transform the design model to C, C++ and Java before executing it.

Of the tools that can only be used to test the code that has been generated from the design model we find Eclipse TPTP to be the best. We like Eclipse TPTP because of its UML support and integration possibilities. It is possible to integrate MDA tools into Eclipse TPTP and then use them from there.

The tool that we find best in the test case generation group is Conformiq. Unlike T-Vec it does not require the tester to learn a new modelling language. It is also good as it works in a non-deterministic way which means that it can generate parameter values by it self. We find this group of tools very interesting and expect a future where tools like this will be used more often.

### 6.3.9 Conclusions from the case study

In this chapter the case study and the evaluation of test tools is described. The case study was needed to develop criteria and getting them prioritized by the respondents. In total there were 9 respondents of which 7 were members of the project group and 2 were testers outside it.

To prioritize the requirements the Hundred-Dollar Test method was used. This prioritization method was chosen as it is a very efficient prioritization technique. It is also an easy method for the respondents to understand. Questionnaires were sent electronically to 7 project members as well as two testers outside of the project. 26 criteria that were grouped into six groups were asked to be prioritized by the respondents. The highest prioritized criterion was the visual runtime analysis of the system. The second highest prioritized criterion was the use of models to test the system. The least valued criterion was integration of Rational Rose.

In overall there is no difference in how the test tools compared to each other when only using the criteria prioritizations of the significant respondents. This shows that the prioritized list of test tools is accurate and general for the whole project group at Ericsson. It also shows that all the respondents have similar likings in which criteria they think are important for test tools to support.

As I-Logix Rhapsody does not support J2EE, the optimal tool for Ericsson to use when testing their MDA application is Pathfinder PathMATE (see chapter 5.6) in conjunction with RSA.

The tools that obtained the most amounts of points are the MDD development tools. These tools offer both development and test features. I-Logix Rhapsody is the MDD tool that obtained most points. It supports 18 out of the total 26 criteria.

The group of tools that are in the middle of the points table are the non-MDD tools where test cases are written in code. As they have all obtained quite few points it is evident that these tools are highly appreciated by the project members. They can unfortunately not be used to validate models but are very useful to test if the generated code is working properly or not.

The type of tools that obtained the least amount of points are the test case generation tools. The reason why they have such low points is that they do not offer all of the features that the MDD tools do. They are only focused on developing test cases and not validation and execution of models. They also have less integration features that are very important for the members of the project group. In this group T-Vec RAVE obtained most points mainly due to its versatility.

# 7 THESIS CONCLUSION

The purpose of this thesis was to investigate which methods and tools that can be used to test products developed in an MDA compatible way. This was done by performing a literature study and a case study at Ericsson.

There are different ways in how products developed in an MDA compatible way can be tested. Possible ways are to validate the design model, execute the design model, test the code that has been generated from the design model, and generate test cases from either test models or design models.

It is not possible to validate models with the use of test cases written in code. Test cases written in code are meant to test the code that has been generated from the design models. A way of validating models was therefore needed. U2TP is a test profile that has been standardized by OMG. U2TP was developed because the OMG belief was that also the testing part of MDA should be specified with UML. With the UML 2 standard it is possible to develop profiles for specific needs. Therefore many tool vendors develop their own test profiles to be used when validating the design model.

One objective of the thesis was to investigate how protocol testing could be performed on products developed in an MDA compatible way. Protocol testing of applications can be performed in many different ways. Protocol testing of UML models can however mainly be performed in two different ways. The first approach is to execute the design model. Executing the design model lets the tester see what happens inside the application when different messages and signals are sent. The second is by the use of U2TP or custom profiles that use sequence diagrams to send messages and signals to the SUT (System Under Test). Observations can then be done to see if the SUT behaves in a way that is specified in the sequence diagram.

To find the most appropriate tool for the project group at Ericsson to use when testing their MDA application, we developed a set of criteria. The criteria were then prioritized by the project group. Based on which criteria the test tools supported the most suitable tool was found.

The tool that to the largest extent supports the prioritized criteria is I-Logix Rhapsody. It is the MDD environment that has the best test features. However it does not support J2EE and can therefore not be used by the project group.

The study determined that the most appropriate tools for the project group at Ericsson to use when testing their MDA application is Pathfinder PathMATE together with Rational Software Architect. Pathfinder PathMATE is designed to be used in conjunction with Rational Software Architect. It adds the function to execute the design model and by that allow the testers to perform protocol testing early in the development stages. The reasons why they are considered the most appropriate tools are mainly because of their support for both J2EE and the tools used at Ericsson.

# 8 FUTURE WORK

Additional work can be done to improve and evolve the result of this thesis:

- More thorough exploration of test tools. By spending more time exploring test tool, additional tools and features can be found and new opinions of each tool can be formed.

- Determine if the tools and methods presented in this thesis are feasible when applying them in larger scale on larger systems

- Determine which methods and tools that fit which test phase best. The optimal tools for unit testing, function testing, and system testing can then be established.

- Reverse engineering of legacy code. Determine which MDA tool supports reverse engineering best.

- Determine what a complete and integrated MDA test environment should look like.

# 9 ABBREVIATIONS

| | |
|---|---|
| ASN.1 | Abstract Syntax Notation One |
| CIM | Computational Independent Model |
| CORBA | Common Object Request Broker Architecture |
| EMF | Eclipse Modeling Framework |
| MDA | Model Driven Architecture |
| MDD | Model Driven Development |
| OMG | Object Management Group |
| PIM | Platform Independent Model |
| PSM | Platform Specific Model |
| SUT | System Under Test |
| TTCN-3 | Testing and Test Control Notation version 3 |
| TPTP | Eclipse Test & Performance Tools Platform Project |
| U2TP | UML 2 Test Profile |
| UML | Unified Modeling Language |
| XMI | XML Metadata Interchange |
| XML | eXtensible Markup Language |
| XSD | XML Schema Definition |
| xUML | Executable UML |

# 10 REFERENCES


**[1]** Object Management Group, *MDA Guide Version 1.0.1*

**[2]** Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno *An Introduction to UML Profiles,* European Journal for the Informatics Professional, 2004

**[3]** Object Management Group Website, *MDA: Executive Overview,* 2005 http://www.omg.org/mda/executive_overview.htm, last accessed 2005-10-21

**[4]** Maria Varsamau, Theodore Antonakopoulos, and Nikolaos Papandreou. *From Protocol Models to Their Implementation: A Versatile Testing Methodolgy*. University of Patras, IEEE 2004

**[5]** Searchnetworking.com, last accessed 2006-01-02 http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212839,00.html

**[6]** Java Website – J2EE, *http://java.sun.com/j2ee/index.jsp, last accessed 2005-12-20*

**[7]** John W. Creswell, *Research Design 2nd ed.,* Sage Publication, 2003

**[8]** Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineeing . An Introduction* Kluwer Academic Publishers 2000

**[9]** Colin Robson, *Real World Research 2nd ed.,* Blackwell Publishing, 2002

**[10]** Architecture Board ORMSC, *Model Driven Architecture (MDA),* 2001

**[11]** Alex E. Bell, *UML Fever,* 2005 pp. 48-56

**[12]** Anne Klepper, Jos Warmer, and Wim Bast, *MDA Explained – The Model Driven Architecture: Practise and Promise*, Addison-Wesley, 2003

**[13]** Object Management Group, *Unified Modeling Language Specification Version 2.0*, 2005

**[14]** Jordi Cabot and Cristina Gómez, *A simple yet useful approach to implementing UML Profiles in current CASE tools***,** Universitat Politècnica de Catalunya

**[15]** Anas Abouzahra, Jean Bézivin, Marcos Didonet Del Fabro, and Frédéric Jouault *A Practical Approach to Bridging Domain Specific Languages with UML profiles 2005* ATLAS Group, University of Nantes

**[16]** The Open Group Website, *http://www.opengroup.org/cio/MDA-ADM/,* last accessed 2005-10-17

**[17]** Chirs Raistrick, Paul Francis, John Wright, Colin Carter, and Ian Wilkie, *Model Driven Architecture with Executable UML,* Cambridge University Press, 2004

**[18]** Morgan Björkander, *Model-driven development, Telelogic TAU and MDA*

**[19]** Jean Bezivin, *Model Engineering for Software Modernization,* University of Nantes, The 11th IEEE Working Conference on Reverse Engineering, IEEE 2004

**[20]** Colin Atkinson, *The Role of Metamodeling in MDA,* Darmstadt University of Technology, Whitepaper

**[21]** Colin Atkinson and Thomas Kühne* *Model-Driven Development: A Metamodeling Foundation,* Darmstadt University of Technology & *University of Mannheim**,* IEEE 2003

**[22]** Ed Seidewitz, *What Models Mean,* InteliData Technologies, IEEE 2003

**[23]** Ann Fruhling, Kimberly Tyser, and Gert-Jan de Vreede *Experiences with Extreme Programming in Telehealth: Developing and Implementing a Biosecurity Health Care Application,* University of Nebraska at Omaha ,Proceedings of the 38th Hawaii International Conference on System Sciences 2005, IEEE 2005

**[24]** I. Sommerville *Software Engineering,* 7th ed. Harlow, England, New York : Pearson/Addison Wesley, 2004.

**[25]** Jean-Guy Schneider and, Lorraine Johnston, *eXtreme Programming—helpful or harmful in educating undergraduates?,* School of Information Technology, Swinburne University of Technology, The Journal of Systems and Software, Dec. 2003

**[26]** Phillipe Kruchten, *The Rational Unified Process – En Introduktion* (Swedish version) Addison Wesley, 2002

**[27]** Flashline.com Website, *http://www.flashline.com/content/Ambler/reuseRUP.jsp***,** last accessed 2005-11-16

**[28]** Kent Beck, *Embracing Change with Extreme Programming,* IEEE Oct.

**[29]** B. Boehm**,** *Get Ready for Agile Methods, with Care,* IEEE Computer, 2002, pp. 64-69.

**[30]** M. Muller and F. Padberg, *On the Economic Evaluation of XP Projects* In Proceedings of the ninth European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering, September 2003, pp. 168-177.

**[31]** TTCN-3 Website, *http://www.ttcn-3.org/* , last accessed 2005-10-06

**[32]** JUnit Webiste, http://www.junit.org, last accessed 2005-10-31

**[33]** Jeff Offutt and Aynur Abdurazuk, *Generating Tests from UML Specifications,* George Mason University, Asia-Pacific Software Engineering Conference (APSEC'04), IEEE 2004

**[34]** Mark Blackburn, Aaron Nauman, Bob Busser, *Defect Identification with Model-Based Test Automation,* Software Productivity Consortium, 2002

**[35]** Wuwei Shen, Kevin Compton, and James Huggins*, *A Toolset for Supporting UML Static and Dynamic Model Checking*, The University of Michigan, Kettering University*

**[36]** Blaine Simpson, *JUnit – Howto*, September 2005

**[37]** T-Vec, *We cover all the boundary for your software*

**[38]** I-Logix, *Rhapsody Automatic Test Generation (ATG) User Guide version 2.3*

**[39]** I-Logix, *Rhapsody TestConductor Tuturiol*

**[40]** Paul Baker and Zhen Ru Dai, Jens Grabowski, Øystein Haugen, Serge Lucio, Eric Samuelsson, Ina Schieferdecker, and Clay E.Williams, *The UML 2.0 Testing Profile*

**[41]** Oject Management Group, *UML Testing Profile Specification Version 1.0,*

**[42]** Magnus Persson and Lennart Löfgren, *Testing UML models, Telelogic*

**[43]** Scott Niemann, *Executable System Design UML 2.0,* I-Logix,

**[44]** Alan Hartman, *Agedis Model Based Test Generation Tools,* Agedis Consortium

**[45]** Francesca Basanieri, Antonia Bertoloni and Eda Marchetti, *The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects*
Istituto di Elaborazione della Infrmazione, Pisa

**[46]** F. Basinieri, A. Bertoloni, E. Marchetti, A.Ribolini, G. Lombardi*, and G. Nucera*
*An Automated Test Strategy Based in UML Diagrams,* Istituto di Elaborazione della Infrmazione, Pisa* Ericsson Lab Italy

**[47]** Wang Linzhang, Yuan Jiesong, Yu Xiaofeng, Hu Jun, Li Xuandong, Zheng Guoliang, and Jiangsu, Nanjing, *Generating Test Cases from UML Activity Diagram based on Gray-Box Method* State Key Laboratory of Novel Software Technology & Department of Computer Science and Technology, Nanjing University

**[48]** Pathfinder Solutions, *Spotlight User Guide version 1.9***,** July 2005

**[49]** In –S Chun and Malcom Munro, Applying Conventional Testing Techniques for Class Testing, IEEE 1996, pp 447-454

**[50]** Conformiq Website**,** http://www.conformiq.com/products/ctg.html, last accessed 2005-10-21

**[51]** Conformiq, Conformiq, *Conformiq Test Generator - Dynamic Model-based Test Automation*

**[52]** Compuware OptimalJ**,** *How transformation patterns transform UML models into high-quality J2EE applications,* 2005

**[53]** King's College London & University of York, *An Evolution of Compuware OptimalJ Professioanl Edition as an MDA Tool*, 2003

**[54]** Cow Suite Website, http://pacinotti.isti.cnr.it/ERI/cowsuite/index.html, *last accessed 2005-11-02*

**[55]** Francesca Basanieri, Anotnia Bertlino, and Eda Marchetti**,** *The Cow Suite Approach To Planning and Deriving Test Suites in UML Project*

**[56]** Eclipse TPTP Website, www.eclipse.org/tptp, *last accessed 2005-10-08*

**[57]** Eclipse Modeling Framework Website, www.eclipse.org/emf, last accessed 2005-10-08

**[58]** IBM Corporation and Intel Corporation, *Building a Custom Test Execution Environment*, Eclipse Conference 2005 Tutorial 6,

**[59]** Rational Rose RealTime, *Toolset Guide***,** June 2003

**[60]** Rational, *Rational Software Architect User Manual*

**[61]** Telelogic Website, http://www.telelogic.com, last accessed 2005-10-21

**[62]** Cris Kobryn and Eric Samuelsson, *Driving Architectures with UML2, The TAU Generation2 Approach to MDA,* , August 1, 2003, Telelogic

**[63]** Object Management Group, UMLTesting Profile Final Adopted Specification, August 2003

**[64]** Dean Leffingwell and Don Widrig. *Managing Software Requirements – A Use Case Approach*

**[65]** Ernst Forman, DSc. *Decision By Objectives – how to convince others that you are right,*
Georg Washington University.
Addison-Wesley, 2003

**[66]** Lena Karlsson, Patrik Berander, Rjörn Regnell* and Claes Wohlin* *Requirements Prioritisation: An Experiment on Exhaustive Pair-Wise Comparison versus Planning Game Partioning*, Lund University, * Blekinge Institute of Technology

**[67]** Newkirk, J. W. and Martin, R. C. *Extreme Programming in Practice*, 2001

**[68]** Social Research Methods Website, External Validiyt
http://www.socialresearchmethods.net/kb/external.htm

**[69]** Georgtown University website – Chi-square calculator,
http://www.georgetown.edu/faculty/ballc/webtools/web_chi.html,
last accessed 2005-12-16

**[70]** TITAN, TITAN User Guide, Ericsson R & D, Hungary

# APPENDIX A - JUNIT

JUnit is an open source-testing framework for Java. JUnit is mostly used by developers who implement unit tests in Java [**9].** In Java a unit is often a class. Test cases in JUnit are declared manually and then run automatically by its engine. JUnit provides an easy way to express the expected behaviour of the code. By expressing which expectations there are on the different methods implemented, the tester can use the JUnit test runners to verify that the code behaves according to his expectations.

In JUnit the test designer define each test as a method. The test classes that contain these test methods will then be ascendants to TestCase, which is the JUnit test framework. Test cases are assembled into suites that make it possible to run multiple test cases together. In JUnit there are numerous ways to code assertions. The mostly used assertions methods are:

- assertTrue(). This method evaluates a Boolean method and fails when the condition is false.

- assertEquals(). This method evaluates whether two values or objects are equal and fails when they aren't.

JUnit was constructed so that it stops as soon as an assertion inside the test fails. The reason for this was that it doesn't matter in how many ways a test fails but what does matter is that it fails. The developer should then correct the error and rerun the test.

In JUnit there is also a possibility to add extensions to customize it for specialized use. These extensions usually interface with JUnit in a simple way by providing their own base class. One example of extensions in JUnit is HttpUnit, which is a framework that makes it possible to write, automated functional tests for websites with JUnit.

# APPENDIX B – QUESTIONNAIRE

## Information page

This questionnaire consists of 26 criteria and 2 questions. You will find the questionnaire on the next sheet called questionnaire (at the bottom left corner). The questionnaire is divided into three different dimensions. The first dimension is about the criteria groups. The second dimension is about the criteria inside each group and the third dimension is concerning the two questions at the end of the questionnaire.

First dimension: Please divide 100 points between the 6 criteria groups (Test case generation, Integration, Usage, Models, MDA and Testing). The most important group should get the most points.

Example: Test case generation 35 points, Integration 10 points, Usage 15 points, models 10 points, MDA 10 points, Testing 20 points. They all equal 100 points and the Test case generation group is considered the most important group.

Second dimension: Please divide 100 points between the criteria in each group. Where the most important criteria should get the most points.

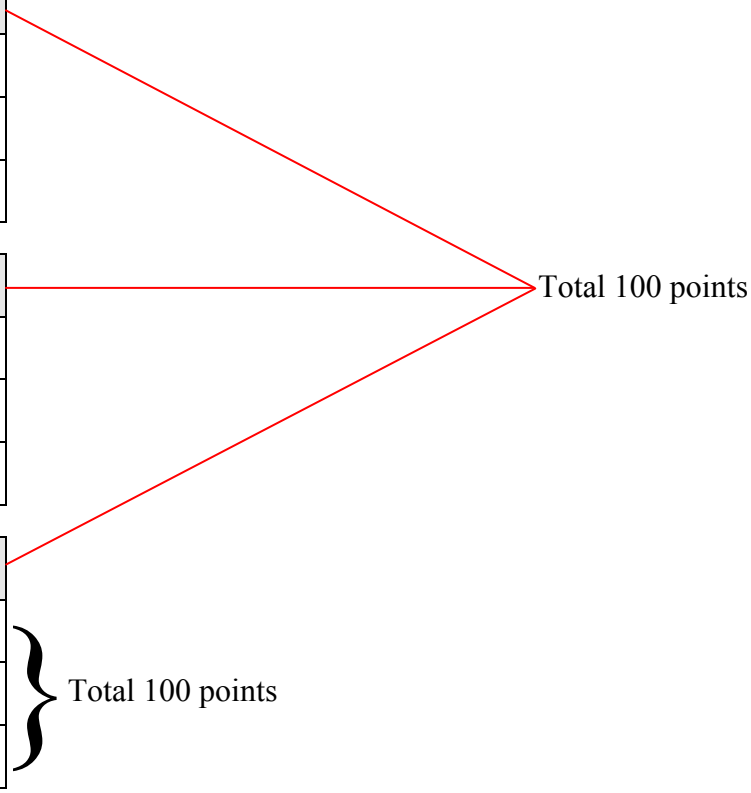
Example: In test case generation group: Generate ttcn-3 test cases 30 points, automatic generation of test cases from design models 30 points, Automatic generation of test cases from test models 40 points. They all equal 100 points and the Automatic generation of test cases is considered to be the most important criteria inside the Test case generation group.

Third dimension: Please answer the two questions at the bottom of the questionnaire.

| Dimension 1<br>Criteria group 1 | 30 |
|---|---|
| **Dimension 2<br>Criteria 1** | **30** |
| **Dimension 2<br>Criteria 2** | **30** |
| **Dimension 2<br>Criteria 3** | **40** |

| Dimension 1<br>Criteria group 2 | 30 |
|---|---|
| **Dimension 2<br>Criteria 1** | **20** |
| **Dimension 2<br>Criteria 2** | **30** |
| **Dimension 2<br>Criteria 3** | **50** |

| Dimension 1<br>Criteria group 3 | 40 |
|---|---|
| **Dimension 2<br>Criteria 1** | **65** |
| **Dimension 2<br>Criteria 2** | **15** |
| **Dimension 2<br>Criteria 3** | **20** |




**Example of dimension 1 & 2**

| | | |
|---|---|---|
| **Dimension 1** Group 1 | Test Case Generation | |
| **Dimension 2** Criteria 1 | **Generate ttcn-3 test cases** The test tool is able to generate test cases in TTCN-3 format, which can then be compiled and executed in the TTCN-3 environment. | |
| **Dimension 2** Criteria 2 | **Automatic generation of test cases from design models** Unit and component (basic test) test cases can be generated from the models that describe the system. | |
| **Dimension 2** Criteria 3 | **Automatic generation of test cases from test models** Test cases are generated from specific models only created for testing purposes. | |
| **Optional** | Please state and give points to any additional criteria you find important that are not included in the group. | |

Total points of criteria inside the group **0**

Points Left 100

| | | |
|---|---|---|
| **Dimension 1** Group 2 | Integration | |
| **Dimension 2** Criteria 1 | **Integration with Rational Software Architect (RSA)** The test tool will be able to integrate with RSA. | |
| **Dimension 2** Criteria 2 | **Import of models from Rational Rose** The test tool can import UML models from Rational Rose. | |
| **Dimension 2** Criteria 3 | **Integration with Requisite pro** The test tool will be able to integrate and work completely with Requisite pro. | |
| **Dimension 2** Criteria 4 | **Integration with Rational ClearCase UCM for configuration management** The test tool is able to work completely with ClearCase UCM and support the extra functions that exist in UCM. | |
| **Dimension 2** Criteria 5 | **Integration with Rational ClearCase base for version control** The test tool is able to work with ClearCase base. | |
| **Dimension 2** Criteria 6 | **Integration with the Eclipse platform** The test tool is able to integrate into the Eclipse platform. | |
| **Optional** | Please state and give points to any additional criteria you find important that are not included in the group. | |

otal points of criteria inside the group **0**

Points left 100

| | | |
|---|---|---|
| **Dimension 1** Group 3 | Usage | |
| **Dimension 2** Criteria 1 | **Good and easy to understand report of test execution/generation result** After the test execution/generation is completed the tester is able to see the result in a thoroughly organized format. | |
| **Dimension 2** Criteria 2 | **Generating report of test execution/generation in a custom format** After the test execution/generation the tester can see the results in a custom format. | |
| **Dimension 2** Criteria 3 | **Good documentation** The test tool will include good documentation to help the tester to learn and use the testing tool. | |
| **Dimension 2** Criteria 4 | **Good support** The test tool will include good support so that the tester can get assistance when the help can't be found in the documentation. | |
| **Dimension 2** Criteria 5 | **Support ports already defined** The test tool will be able to use the ports that have already been defined in TTCN-3 so that they don't need to be redefined in the tool. | |
| **Dimension 2** Criteria 6 | **Support the current process** The test tool will support the current process used for developing systems at Ericsson and will not force the designers to change the order in which they work. | |
| **Optional** | Please state and give points to any additional criteria you find important that are not included in the group. | |
| | Total points of criteria inside the group | **0** |
| | Points left | 100 |

| | | |
|---|---|---|
| **Dimension 1** Group 4 | Models | |
| **Dimension 2** Criteria 1 | **Use of models to test the system** Ability to test the system by the use of models like Sequence Diagrams and the UML Testing profile. | |
| **Dimension 2** Criteria 2 | **Syntax testing of the design models** The test tool will check that the models conform to accurate syntax rules. | |
| **Dimension 2** Criteria 3 | **Software inspections** The tool analyzes the UML models and check if they satisfy quality requirements such as correctness, consistency, testability and maintainability. | |
| **Optional** | Please state and give points to any additional criteria you find important that are not included in the group. | |
| | Total points of criteria inside the group | **0** |
| | Points left | 100 |

| | | |
|---|---|---|
| **Dimension 1** Group 5 | MDA (Model Driven Architecture) | |
| **Dimension 2** Criteria 1 | **Platform test on PIM** The test tool can check if it's possible to implement the Platform Independent Model (PIM) on the platform it is intended for. | |
| **Dimension 2** Criteria 2 | **Analysis of PIM to PSM** The test tool can analyse the Platform Specific Model (PSM) to see how well the transformation from PIM to PSM has been conducted. | |
| **Dimension 2** Criteria 3 | **Architecture analysis of Models** Test tool can check if the PIM and PSM Models follow specific architectural patterns. | |
| **Dimension 2** Criteria 4 | **PSM assumptions of the platform** Test tool checks that PSM works completely on the platform it is intended for. That it doesn't violate assumptions of the platform it is intended for. | |
| **Optional** | Please state and give points to any additional criteria you find important that are not included in the group. | |
| | Total points of criteria inside the group | **0** |
| | Points Left | 100 |

| | | |
|---|---|---|
| **Dimension 1** Group 6 | Testing | |
| **Dimension 2** Criteria 1 | **Visual runtime analysis of the system** Ability to execute and debug the system by sending messages and signals to see what happens inside of it. | |
| **Dimension 2** Criteria 2 | **Test according to XML schema message and boundary** The test tool will verify that the information given by the tester complies with the XML schema used for specification. | |
| **Dimension 2** Criteria 3 | **Use of stubs to basic test** The test tool has a function to use stubs so that components that are not implemented yet can be simulated. | |
| **Dimension 2** Criteria 4 | **Use of code to test the system** Ability to test the system with test cases written in code (for example Java test cases). | |
| **Optional** | Please state and give points to any additional criteria you find important that are not included in the group. | |
| | Total points of criteria inside the group | **0** |
| | Points Left | 100 |

..

**Dimension 3** Background questions

| | |
|---|---|
| Question 1 | What role do you have in the project? |
| Question 2 | How good knowledge do you have on MDA (in a scale from 1 to 5 where 1 is not good and 5 is very good) |

# APPENDIX C – STATISTICS

This appendix includes some statistics and diagrams. The following three tables are all dependent of each other and shall be observed as one long table. Table 1.1 illustrates a summary of the questionnaires result, the table includes criteria number, name of the groups (the gray row) and criteria, the result of questionnaire 1 to 9, the result of all the questionnaires (see chapter 6.3.3 for calculation), the result of the project members (questionnaire 3 to 9) and, the result of the testers (questionnaire 1 and 2).

| Nr | Criteria / Respondent | q1 | q2 | q3 | q4 | q5 | q6 | q7 | q8 | q9 | All Average | All Total | Project Member Average | Project Member Total | Tester Average | Tester Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Test Case Generation** | **5** | **20** | **30** | **15** | **10** | **15** | **15** | **25** | **5** | **15,6** | | **16,4** | | **12,5** | |
| 1 | Generate ttcn-3 test cases | 30 | 30 | 20 | 30 | 20 | 20 | 20 | 20 | 30 | 24,4 | 380,2 | 22,9 | 375,5 | 30,0 | 375,0 |
| 2 | Automatic generation of test cases from design models | 0 | 40 | 50 | 40 | 40 | 60 | 40 | 40 | 10 | 35,6 | 553,1 | 40,0 | 657,1 | 20,0 | 250,0 |
| 3 | Automatic generation of test cases from test models | 30 | 30 | 30 | 30 | 40 | 20 | 40 | 40 | 60 | 35,6 | 553,1 | 37,1 | 610,2 | 30,0 | 375,0 |
| | **Integration** | **25** | **10** | **20** | **20** | **20** | **15** | **20** | **20** | **25** | **19,4** | | **20,0** | | **17,5** | |
| 4 | Integration with Rational Software Architect (RSA) | 20 | 5 | 20 | 20 | 20 | 20 | 20 | 15 | 5 | 16,1 | 313,3 | 17,1 | 342,9 | 12,5 | 218,8 |
| 5 | Import of models from Rational Rose | 0 | 15 | 0 | 5 | 10 | 0 | 0 | 0 | 10 | 4,4 | 86,4 | 3,6 | 71,4 | 7,5 | 131,3 |
| 6 | Integration with Requisite pro | 25 | 8 | 20 | 20 | 10 | 20 | 20 | 20 | 25 | 18,7 | 363 | 19,3 | 385,7 | 16,5 | 288,8 |
| 7 | Integration with Rational ClearCase UCM for configuration management | 20 | 20 | 20 | 20 | 15 | 20 | 20 | 15 | 20 | 18,9 | 367,3 | 18,6 | 371,4 | 20,0 | 350,0 |
| 8 | Integration with Rational ClearCase base for version control | 10 | 42 | 20 | 20 | 15 | 20 | 20 | 10 | 10 | 18,6 | 360,8 | 16,4 | 328,6 | 26,0 | 455,0 |
| 9 | Integration with the Eclipse platform | 25 | 10 | 20 | 15 | 30 | 20 | 20 | 40 | 20 | 19,2 | 372,48 | 23,6 | 471,4 | 17,5 | 306,3 |

| Nr | Criteria / Respondent | q1 | q2 | q3 | q4 | q5 | q6 | q7 | q8 | q9 | All Average | All Total | Project Member Average | Project Member Total | Tester Average | Tester Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Usage** | **25** | **30** | **10** | **15** | **15** | **15** | **20** | **25** | **30** | **18,5** | | **18,6** | | **27,5** | |
| 10 | Good and easy to understand report of test execution/generation result | 10 | 20 | 30 | 20 | 20 | 40 | 10 | 20 | 25 | 21,7 | 400,8 | 23,6 | 437,8 | 15,0 | 412,5 |
| 11 | Generating report of test execution/generation in a custom format | 30 | 10 | 20 | 20 | 10 | 10 | 10 | 0 | 10 | 13,3 | 246,7 | 11,4 | 212,2 | 20,0 | 550,0 |
| 12 | Good documentation | 10 | 35 | 10 | 15 | 15 | 20 | 40 | 25 | 20 | 21,1 | 390,6 | 20,7 | 384,7 | 22,5 | 618,8 |
| 13 | Good support | 10 | 25 | 20 | 15 | 10 | 20 | 20 | 20 | 25 | 18,3 | 339,2 | 18,6 | 344,9 | 17,5 | 481,3 |
| 14 | Support ports already defined | 40 | 5 | 10 | 20 | 10 | 5 | 10 | 20 | 15 | 15,0 | 277,5 | 12,9 | 238,8 | 22,5 | 618,8 |
| 15 | Support the current process | 0 | 5 | 10 | 10 | 25 | 5 | 10 | 15 | 5 | 9,4 | 174,7 | 11,4 | 212,2 | 2,5 | 68,8 |
| | **Models** | **15** | **15** | **10** | **20** | **15** | **20** | **15** | **10** | **10** | **14,4** | | **14,3** | | **15,0** | |
| 16 | Use of models to test the system | 50 | 20 | 70 | 40 | 25 | 40 | 40 | 40 | 30 | 39,4 | 569,8 | 40,7 | 581,6 | 35,0 | 525,0 |
| 17 | Syntax testing of the design models | 30 | 50 | 15 | 30 | 50 | 30 | 20 | 30 | 20 | 30,6 | 441,4 | 27,9 | 398,0 | 40,0 | 600,0 |
| 18 | Software inspections | 20 | 30 | 15 | 30 | 25 | 30 | 40 | 30 | 20 | 26,7 | 385,2 | 27,1 | 387,8 | 25,0 | 375,0 |
| | **MDA (Model Driven Architecture)** | **0** | **10** | **10** | **10** | **10** | **25** | **15** | **0** | **15** | **10,6** | | **12,1** | | **5,0** | |
| 19 | Platform test on PIM | 0 | 20 | 20 | 25 | 40 | 30 | 40 | 20 | 25 | 24,4 | 258 | 28,6 | 346,9 | 10,0 | 50,0 |
| 20 | Analysis of PIM to PSM | 0 | 10 | 20 | 25 | 30 | 40 | 40 | 20 | 25 | 23,3 | 246,3 | 28,6 | 346,9 | 5,0 | 25,0 |
| 21 | Architecture analysis of Models | 0 | 30 | 20 | 25 | 20 | 15 | 10 | 30 | 10 | 17,8 | 187,7 | 18,6 | 225,5 | 15,0 | 75,0 |
| 22 | PSM assumptions of the platform | 0 | 40 | 20 | 25 | 10 | 15 | 10 | 30 | 40 | 21,1 | 222,8 | 21,4 | 260,2 | 20,0 | 100,0 |

| Nr | Criteria / Respondent | q1 | q2 | q3 | q4 | q5 | q6 | q7 | q8 | q9 | All Average | All Total | Project Member Average | Project Member Total | Tester Average | Tester Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Testing** | **30** | **15** | **20** | **20** | **30** | **10** | **15** | **20** | **15** | **19,4** | | **18,6** | | **22,5** | |
| 23 | Visual runtime analysis of the system | 40 | 45 | 30 | 30 | 50 | 50 | 20 | 35 | 20 | 35,6 | 691,4 | 33,6 | 623,5 | 42,5 | 956,3 |
| 24 | Test according to XML schema message and boundary | 25 | 35 | 20 | 20 | 10 | 10 | 20 | 35 | 20 | 21,7 | 389,2 | 19,3 | 358,2 | 30,0 | 675,0 |
| 25 | Use of stubs to basic test | 25 | 15 | 20 | 30 | 20 | 30 | 40 | 10 | 20 | 23,3 | 408,3 | 24,3 | 451,0 | 20,0 | 450,0 |
| 26 | Use of code to test the system | 10 | 5 | 30 | 20 | 20 | 10 | 20 | 20 | 20 | 17,2 | 334,9 | 20,0 | 371,4 | 7,5 | 168,8 |

**Table C.1** The questionnaires result

Table 1.2 illustrates each significant pair average on every group and criteria. For example, column q1q2 and row Test Case generation has the value 12,5. That is the average of the test case generation group between q1 and q2, the values can be observed in Table 1.1 (q1, 5 and q2, 20).

| Nr | Criteria / Significant pair | q1q2 | q1q3 | q1q4 | q1q5 | q1q6 | q1q7 | q1q8 | q1q9 | q2q3 | q2q5 | q2q6 | q2q8 | q2q9 | q3q5 | q3q6 | q3q8 | q3q9 | q4q8 | q4q9 | q5q6 | q5q8 | q5q9 | q6q7 | q6q8 | q6q9 | q7q8 | q8q9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Test Case Generation** | 12,5 | 17,5 | 10,0 | 7,5 | 10,0 | 10,0 | 15,0 | 5,0 | 25,0 | 15,0 | 17,5 | 22,5 | 12,5 | 20,0 | 22,5 | 27,5 | 17,5 | 20,0 | 10,0 | 12,5 | 17,5 | 7,5 | 15,0 | 20,0 | 10,0 | 20,0 | 15,0 |
| 1 | Generate ttcn-3 test cases | 30,0 | 25,0 | 30,0 | 25,0 | 25,0 | 25,0 | 25,0 | 30,0 | 25,0 | 25,0 | 25,0 | 25,0 | 30,0 | 20,0 | 20,0 | 20,0 | 25,0 | 25,0 | 30,0 | 20,0 | 20,0 | 25,0 | 20,0 | 20,0 | 25,0 | 20,0 | 25,0 |
| 2 | Automatic generation of test cases from design models | 20,0 | 25,0 | 20,0 | 20,0 | 30,0 | 20,0 | 20,0 | 5,0 | 45,0 | 40,0 | 50,0 | 40,0 | 25,0 | 45,0 | 55,0 | 45,0 | 30,0 | 40,0 | 25,0 | 50,0 | 40,0 | 25,0 | 50,0 | 50,0 | 35,0 | 40,0 | 25,0 |
| 3 | Automatic generation of test cases from test models | 30,0 | 30,0 | 30,0 | 35,0 | 25,0 | 35,0 | 35,0 | 45,0 | 30,0 | 35,0 | 25,0 | 35,0 | 45,0 | 35,0 | 25,0 | 35,0 | 45,0 | 35,0 | 45,0 | 30,0 | 40,0 | 50,0 | 30,0 | 30,0 | 40,0 | 40,0 | 50,0 |

| Nr | Criteria / Significant pair | q1q2 | q1q3 | q1q4 | q1q5 | q1q6 | q1q7 | q1q8 | q1q9 | q2q3 | q2q5 | q2q6 | q2q8 | q2q9 | q3q5 | q3q6 | q3q8 | q3q9 | q4q8 | q4q9 | q5q6 | q5q8 | q5q9 | q6q7 | q6q8 | q6q9 | q7q8 | q8q9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Integration** | 17,5 | 22,5 | 22,5 | 22,5 | 20,0 | 22,5 | 22,5 | 25,0 | 15,0 | 15,0 | 12,5 | 15,0 | 17,5 | 20,0 | 17,5 | 20,0 | 22,5 | 20,0 | 22,5 | 17,5 | 20,0 | 22,5 | 17,5 | 17,5 | 20,0 | 20,0 | 22,5 |
| 4 | Integration with Rational Software Architect (RSA) | 12,5 | 20,0 | 20,0 | 20,0 | 20,0 | 20,0 | 17,5 | 12,5 | 12,5 | 12,5 | 12,5 | 10,0 | 5,0 | 20,0 | 20,0 | 17,5 | 12,5 | 17,5 | 12,5 | 20,0 | 17,5 | 12,5 | 20,0 | 17,5 | 12,5 | 17,5 | 10,0 |
| 5 | Import of models from Rational Rose | 7,5 | 0,0 | 2,5 | 5,0 | 0,0 | 0,0 | 0,0 | 5,0 | 7,5 | 12,5 | 7,5 | 7,5 | 12,5 | 5,0 | 0,0 | 0,0 | 5,0 | 2,5 | 7,5 | 5,0 | 5,0 | 10,0 | 0,0 | 0,0 | 5,0 | 0,0 | 5,0 |
| 6 | Integration with Requisite pro | 16,5 | 22,5 | 22,5 | 17,5 | 22,5 | 22,5 | 22,5 | 25,0 | 14,0 | 9,0 | 14,0 | 14,0 | 16,5 | 15,0 | 20,0 | 20,0 | 22,5 | 20,0 | 22,5 | 15,0 | 15,0 | 17,5 | 20,0 | 20,0 | 22,5 | 20,0 | 22,5 |
| 7 | Integration with Rational ClearCase UCM for configuration management | 20,0 | 20,0 | 20,0 | 17,5 | 20,0 | 20,0 | 17,5 | 20,0 | 20,0 | 17,5 | 20,0 | 17,5 | 20,0 | 17,5 | 20,0 | 17,5 | 20,0 | 17,5 | 20,0 | 17,5 | 15,0 | 17,5 | 20,0 | 17,5 | 20,0 | 17,5 | 17,5 |
| 8 | Integration with Rational ClearCase base for version control | 26,0 | 15,0 | 15,0 | 12,5 | 15,0 | 15,0 | 10,0 | 10,0 | 31,0 | 28,5 | 31,0 | 26,0 | 26,0 | 17,5 | 20,0 | 15,0 | 15,0 | 15,0 | 15,0 | 17,5 | 12,5 | 12,5 | 20,0 | 15,0 | 15,0 | 15,0 | 10,0 |
| 9 | Integration with the Eclipse platform | 17,5 | 22,5 | 20,0 | 27,5 | 22,5 | 22,5 | 32,5 | 22,5 | 15,0 | 20,0 | 15,0 | 25,0 | 15,0 | 25,0 | 20,0 | 30,0 | 20,0 | 27,5 | 17,5 | 25,0 | 35,0 | 25,0 | 20,0 | 30,0 | 20,0 | 30,0 | 30,0 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | **Usage** | 27,5 | 17,5 | 20,0 | 20,0 | 20,0 | 22,5 | 25,0 | 27,5 | 20,0 | 22,5 | 22,5 | 27,5 | 30,0 | 12,5 | 12,5 | 17,5 | 20,0 | 20,0 | 22,5 | 15,0 | 20,0 | 22,5 | 17,5 | 20,0 | 22,5 | 22,5 | 27,5 |
| 10 | Good and easy to understand report of test execution/generation result | 15,0 | 20,0 | 15,0 | 15,0 | 25,0 | 10,0 | 15,0 | 17,5 | 25,0 | 20,0 | 30,0 | 20,0 | 22,5 | 25,0 | 35,0 | 25,0 | 27,5 | 20,0 | 22,5 | 30,0 | 20,0 | 22,5 | 25,0 | 30,0 | 32,5 | 15,0 | 22,5 |
| 11 | Generating report of test execution/generation in a custom format | 20,0 | 25,0 | 25,0 | 20,0 | 20,0 | 20,0 | 15,0 | 20,0 | 15,0 | 10,0 | 10,0 | 5,0 | 10,0 | 15,0 | 15,0 | 10,0 | 15,0 | 10,0 | 15,0 | 10,0 | 5,0 | 10,0 | 10,0 | 5,0 | 10,0 | 5,0 | 5,0 |
| 12 | Good documentation | 22,5 | 10,0 | 12,5 | 12,5 | 15,0 | 25,0 | 17,5 | 15,0 | 22,5 | 25,0 | 27,5 | 30,0 | 27,5 | 12,5 | 15,0 | 17,5 | 15,0 | 20,0 | 17,5 | 17,5 | 20,0 | 17,5 | 30,0 | 22,5 | 20,0 | 32,5 | 22,5 |
| 13 | Good support | 17,5 | 15,0 | 12,5 | 10,0 | 15,0 | 15,0 | 15,0 | 17,5 | 22,5 | 17,5 | 22,5 | 22,5 | 25,0 | 15,0 | 20,0 | 20,0 | 22,5 | 17,5 | 20,0 | 15,0 | 15,0 | 17,5 | 20,0 | 20,0 | 22,5 | 20,0 | 22,5 |
| 14 | Support ports already defined | 22,5 | 25,0 | 30,0 | 25,0 | 22,5 | 25,0 | 30,0 | 27,5 | 7,5 | 7,5 | 5,0 | 12,5 | 10,0 | 10,0 | 7,5 | 15,0 | 12,5 | 20,0 | 17,5 | 7,5 | 15,0 | 12,5 | 7,5 | 12,5 | 10,0 | 15,0 | 17,5 |
| 15 | Support the current process | 2,5 | 5,0 | 5,0 | 12,5 | 2,5 | 5,0 | 7,5 | 2,5 | 7,5 | 15,0 | 5,0 | 10,0 | 5,0 | 17,5 | 7,5 | 12,5 | 7,5 | 12,5 | 7,5 | 15,0 | 20,0 | 15,0 | 7,5 | 10,0 | 5,0 | 12,5 | 10,0 |

| Nr | Criteria / Significant pair | q1q2 | q1q3 | q1q4 | q1q5 | q1q6 | q1q7 | q1q8 | q1q9 | q2q3 | q2q5 | q2q6 | q2q8 | q2q9 | q3q5 | q3q6 | q3q8 | q3q9 | q4q8 | q4q9 | q5q6 | q5q8 | q5q9 | q6q7 | q6q8 | q6q9 | q7q8 | q8q9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Models** | 15,0 | 12,5 | 17,5 | 15,0 | 17,5 | 15,0 | 12,5 | 12,5 | 12,5 | 15,0 | 17,5 | 12,5 | 12,5 | 12,5 | 15,0 | 10,0 | 10,0 | 15,0 | 15,0 | 17,5 | 12,5 | 12,5 | 17,5 | 15,0 | 15,0 | 12,5 | 10,0 |
| 16 | Use of models to test the system | 35,0 | 60,0 | 45,0 | 37,5 | 45,0 | 45,0 | 45,0 | 40,0 | 45,0 | 22,5 | 30,0 | 30,0 | 25,0 | 47,5 | 55,0 | 55,0 | 50,0 | 40,0 | 35,0 | 32,5 | 32,5 | 27,5 | 40,0 | 40,0 | 35,0 | 40,0 | 35,0 |
| 17 | Syntax testing of the design models | 40,0 | 22,5 | 30,0 | 40,0 | 30,0 | 25,0 | 30,0 | 25,0 | 32,5 | 50,0 | 40,0 | 40,0 | 35,0 | 32,5 | 22,5 | 22,5 | 17,5 | 30,0 | 25,0 | 40,0 | 40,0 | 35,0 | 25,0 | 30,0 | 25,0 | 25,0 | 25,0 |
| 18 | Software inspections | 25,0 | 17,5 | 25,0 | 22,5 | 25,0 | 30,0 | 25,0 | 20,0 | 22,5 | 27,5 | 30,0 | 30,0 | 25,0 | 20,0 | 22,5 | 22,5 | 17,5 | 30,0 | 25,0 | 27,5 | 27,5 | 22,5 | 35,0 | 30,0 | 25,0 | 35,0 | 25,0 |
| | **MDA (Model Driven Architecture)** | 5,0 | 5,0 | 5,0 | 5,0 | 12,5 | 7,5 | 0,0 | 7,5 | 10,0 | 10,0 | 17,5 | 5,0 | 12,5 | 10,0 | 17,5 | 5,0 | 12,5 | 5,0 | 12,5 | 17,5 | 5,0 | 12,5 | 20,0 | 12,5 | 20,0 | 7,5 | 7,5 |
| 19 | Platform test on PIM | 10,0 | 10,0 | 12,5 | 20,0 | 15,0 | 20,0 | 10,0 | 12,5 | 20,0 | 30,0 | 25,0 | 20,0 | 22,5 | 30,0 | 25,0 | 20,0 | 22,5 | 22,5 | 25,0 | 35,0 | 30,0 | 32,5 | 35,0 | 25,0 | 27,5 | 30,0 | 22,5 |
| 20 | Analysis of PIM to PSM | 5,0 | 10,0 | 12,5 | 15,0 | 20,0 | 20,0 | 10,0 | 12,5 | 15,0 | 20,0 | 25,0 | 15,0 | 17,5 | 25,0 | 30,0 | 20,0 | 22,5 | 22,5 | 25,0 | 35,0 | 25,0 | 27,5 | 40,0 | 30,0 | 32,5 | 30,0 | 22,5 |
| 21 | Architecture analysis of Models | 15,0 | 10,0 | 12,5 | 10,0 | 7,5 | 5,0 | 15,0 | 5,0 | 25,0 | 25,0 | 22,5 | 30,0 | 20,0 | 20,0 | 17,5 | 25,0 | 15,0 | 27,5 | 17,5 | 17,5 | 25,0 | 15,0 | 12,5 | 22,5 | 12,5 | 20,0 | 20,0 |
| 22 | PSM assumptions of the platform | 20,0 | 10,0 | 12,5 | 5,0 | 7,5 | 5,0 | 15,0 | 20,0 | 30,0 | 25,0 | 27,5 | 35,0 | 40,0 | 15,0 | 17,5 | 25,0 | 30,0 | 27,5 | 32,5 | 12,5 | 20,0 | 25,0 | 12,5 | 22,5 | 27,5 | 20,0 | 35,0 |
| | **Testing** | 22,5 | 25,0 | 25,0 | 30,0 | 20,0 | 22,5 | 25,0 | 22,5 | 17,5 | 22,5 | 12,5 | 17,5 | 15,0 | 25,0 | 15,0 | 20,0 | 17,5 | 20,0 | 17,5 | 20,0 | 25,0 | 22,5 | 12,5 | 15,0 | 12,5 | 17,5 | 17,5 |
| 23 | Visual runtime analysis of the system | 42,5 | 35,0 | 35,0 | 45,0 | 45,0 | 30,0 | 37,5 | 30,0 | 37,5 | 47,5 | 47,5 | 40,0 | 32,5 | 40,0 | 40,0 | 32,5 | 25,0 | 32,5 | 25,0 | 50,0 | 42,5 | 35,0 | 35,0 | 42,5 | 35,0 | 27,5 | 27,5 |
| 24 | Test according to XML schema message and boundary | 30,0 | 22,5 | 22,5 | 17,5 | 17,5 | 22,5 | 30,0 | 22,5 | 27,5 | 22,5 | 22,5 | 35,0 | 27,5 | 15,0 | 15,0 | 27,5 | 20,0 | 27,5 | 20,0 | 10,0 | 22,5 | 15,0 | 15,0 | 22,5 | 15,0 | 27,5 | 27,5 |
| 25 | Use of stubs to basic test | 20,0 | 22,5 | 27,5 | 22,5 | 27,5 | 32,5 | 17,5 | 22,5 | 17,5 | 17,5 | 22,5 | 12,5 | 17,5 | 20,0 | 25,0 | 15,0 | 20,0 | 20,0 | 25,0 | 25,0 | 15,0 | 20,0 | 35,0 | 20,0 | 25,0 | 25,0 | 15,0 |
| 26 | Use of code to test the system | 7,5 | 20,0 | 15,0 | 15,0 | 10,0 | 15,0 | 15,0 | 15,0 | 17,5 | 12,5 | 7,5 | 12,5 | 12,5 | 25,0 | 20,0 | 25,0 | 25,0 | 20,0 | 20,0 | 15,0 | 20,0 | 20,0 | 15,0 | 15,0 | 15,0 | 20,0 | 20,0 |

**Table C.2** The group and criteria average of each significant pair

Table 1.3 illustrates the average of all the significant pairs. For example the cell which is in the All Significant Average column and the Test Case Generation row has the value 15,4. This value is the average of all the 27 significant pairs shown in Table 1.2 (the average from q1q2 to q8q9). For the calculation of the total columns please see chapter 6.3.3.

| Nr | Criteria / Respondent | All Significant Average | All Significant Total | Project Member Average | Project Member Total | Testers Average | Tester Total |
|---|---|---|---|---|---|---|---|
| | **Test Case Generation** | **15,4** | | **16,8** | | **12,5** | |
| 1 | Generate ttcn-3 test cases | 24,4 | 375,7 | 22,5 | 377,7 | 30,0 | 375,0 |
| 2 | Automatic generation of test cases from design models | 33,9 | 520,9 | 39,6 | 665,4 | 20,0 | 250,0 |
| 3 | Automatic generation of test cases from test models | 35,7 | 549,3 | 37,9 | 635,5 | 30,0 | 375,0 |
| | **Integration** | **19,6** | | **20,0** | | **17,5** | |
| 4 | Integration with Rational Software Architect (RSA) | 15,6 | 307,2 | 16,3 | 325,0 | 12,5 | 218,8 |
| 5 | Import of models from Rational Rose | 4,4 | 85,4 | 3,6 | 71,4 | 7,5 | 131,3 |
| 6 | Integration with Requisite pro | 18,9 | 371,9 | 19,5 | 389,3 | 16,5 | 288,8 |
| 7 | Integration with Rational ClearCase UCM for configuration management | 18,7 | 367,1 | 18,2 | 364,3 | 20,0 | 350,0 |
| 8 | Integration with Rational ClearCase base for version control | 17,6 | 346,1 | 15,4 | 307,1 | 26,0 | 455,0 |
| 9 | Integration with the Eclipse platform | 23,4 | 413,0 | 25,4 | 507,1 | 17,5 | 306,3 |

| Nr | Criteria / Respondent | All Significant Average | All Significant Total | Project Member Average | Project Member Total | Testers Average | Tester Total |
|---|---|---|---|---|---|---|---|
| | **Usage** | **21,3** | | **19,5** | | **27,5** | |
| 10 | Good and easy to understand report of test execution/generation result | 22,3 | 475,2 | 25,2 | 490,1 | 15,0 | 412,5 |
| 11 | Generating report of test execution/generation in a custom format | 13,1 | 280,0 | 10,0 | 194,6 | 20,0 | 550,0 |
| 12 | Good documentation | 20,1 | 427,9 | 20,0 | 389,3 | 22,5 | 618,8 |
| 13 | Good support | 18,3 | 390,4 | 19,1 | 371,9 | 17,5 | 481,3 |
| 14 | Support ports already defined | 15,9 | 339,2 | 12,9 | 250,3 | 22,5 | 618,8 |
| 15 | Support the current process | 9,1 | 193,2 | 11,4 | 222,4 | 2,5 | 68,8 |
| | **Models** | **14,0** | | **13,6** | | **15,0** | |
| 16 | Use of models to test the system | 39,6 | 554,1 | 40,4 | 547,7 | 35,0 | 525,0 |
| 17 | Syntax testing of the design models | 30,9 | 432,4 | 28,2 | 382,9 | 40,0 | 600,0 |
| 18 | Software inspections | 25,6 | 357,3 | 26,1 | 353,8 | 25,0 | 375,0 |

| Nr | Criteria / Respondent | All Significant Average | All Significant Total | Project Member Average | Project Member Total | Testers Average | Tester Total |
|---|---|---|---|---|---|---|---|
| | **MDA (Model Driven Architecture)** | **9,9** | | **11,8** | | **5,0** | |
| 19 | Platform test on PIM | 22,6 | 223,8 | 27,3 | 322,0 | 10,0 | 15,0 |
| 20 | Analysis of PIM to PSM | 21,7 | 214,7 | 27,7 | 326,2 | 5,0 | 10,0 |
| 21 | Architecture analysis of Models | 17,4 | 172,5 | 19,1 | 225,2 | 15,0 | 20,0 |
| 22 | PSM assumptions of the platform | 21,3 | 211,0 | 23,0 | 271,5 | 20,0 | 25,0 |
| | **Testing** | **19,8** | | **18,4** | | **22,5** | |
| 23 | Visual runtime analysis of the system | 36,9 | 730,2 | 35,0 | 643,8 | 42,5 | 956,3 |
| 24 | Test according to XML schema message and boundary | 22,2 | 440,3 | 20,0 | 367,9 | 30,0 | 675,0 |
| 25 | Use of stubs to basic test | 21,7 | 429,3 | 21,8 | 400,7 | 20,0 | 450,0 |
| 26 | Use of code to test the system | 16,7 | 330,2 | 19,6 | 361,3 | 7,5 | 168,8 |

**Table C.3** The total average of the significant pairs

# APPENDIX D – DIAGRAMS

This appendix includes diagrams from the result of the questionnaires.

## D.1 Test case generation

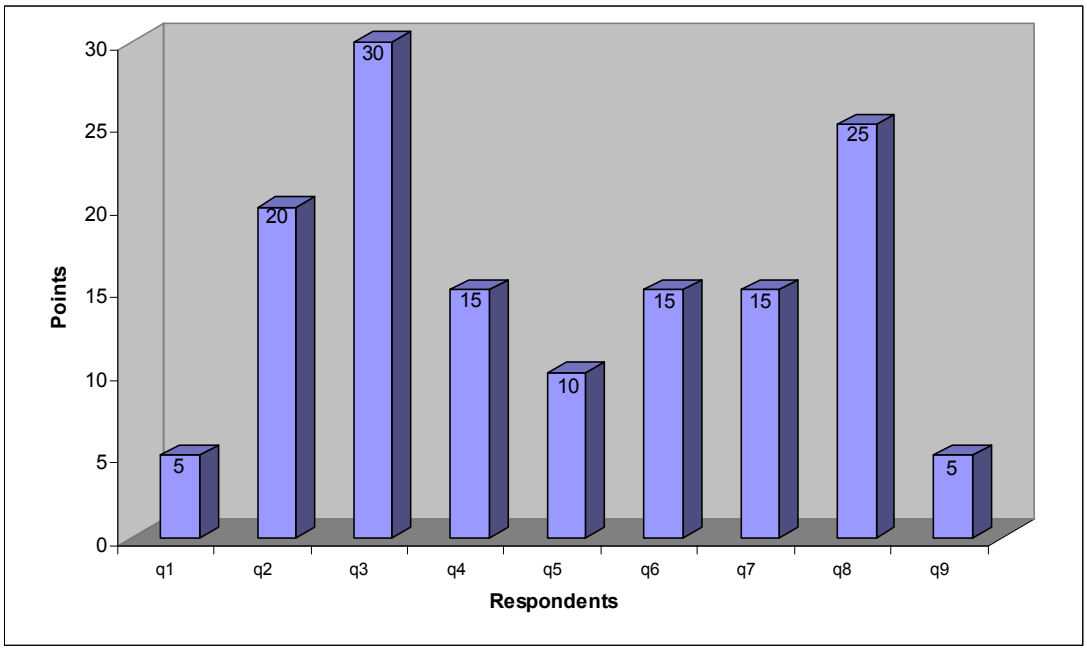



**Figure D.1** The respondents' prioritization of the test case generation group

Figure D.1 illustrates how each respondent prioritized of the test case generation group. Please look at diagram 6.1 for the average points diagram.

## D.2 Integration group

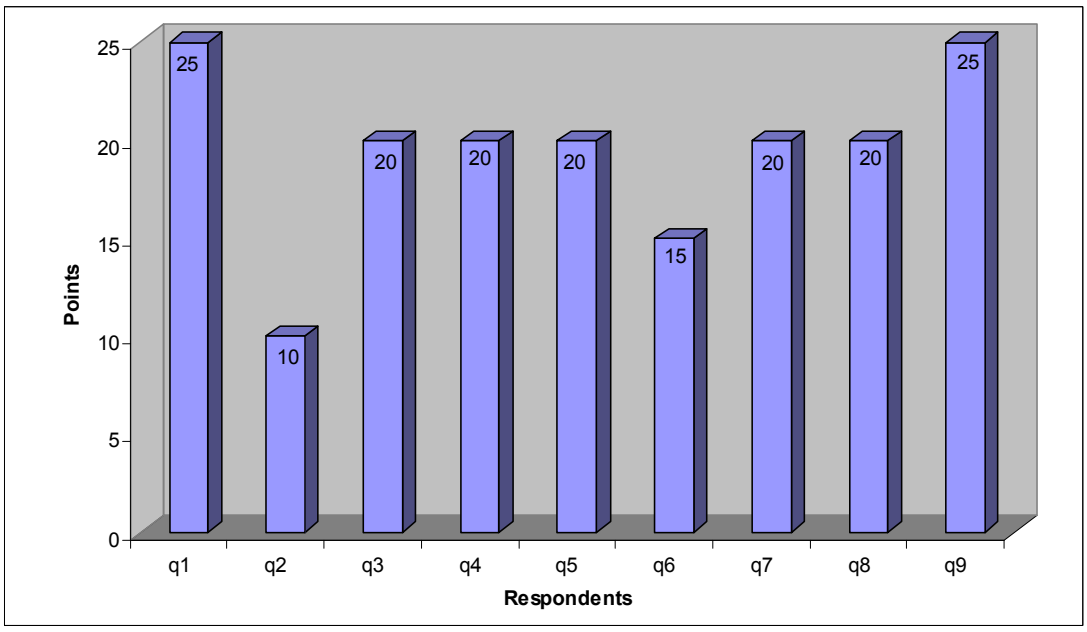



**Figure D.2** The respondents' prioritization of the integration group

Figure D.2 illustrates how each respondent prioritized of the test case generation group.

## D.3 Usage group

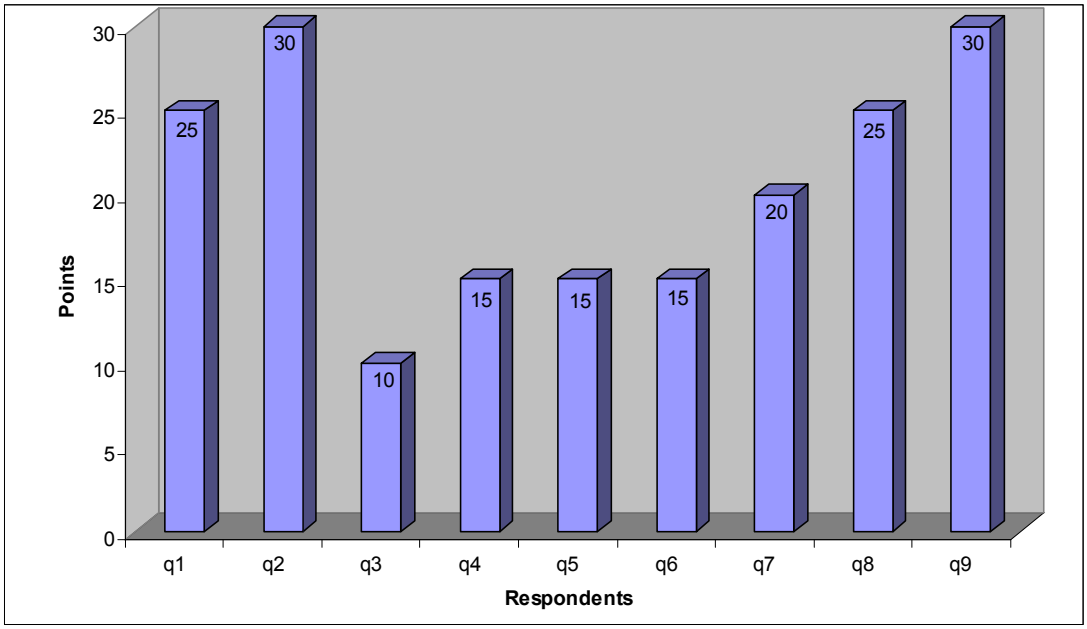



**Figure D.3** The respondents' prioritization of the usage group

Figure D.3 illustrates how each respondent prioritized of the usage group.

## D.4 Models group

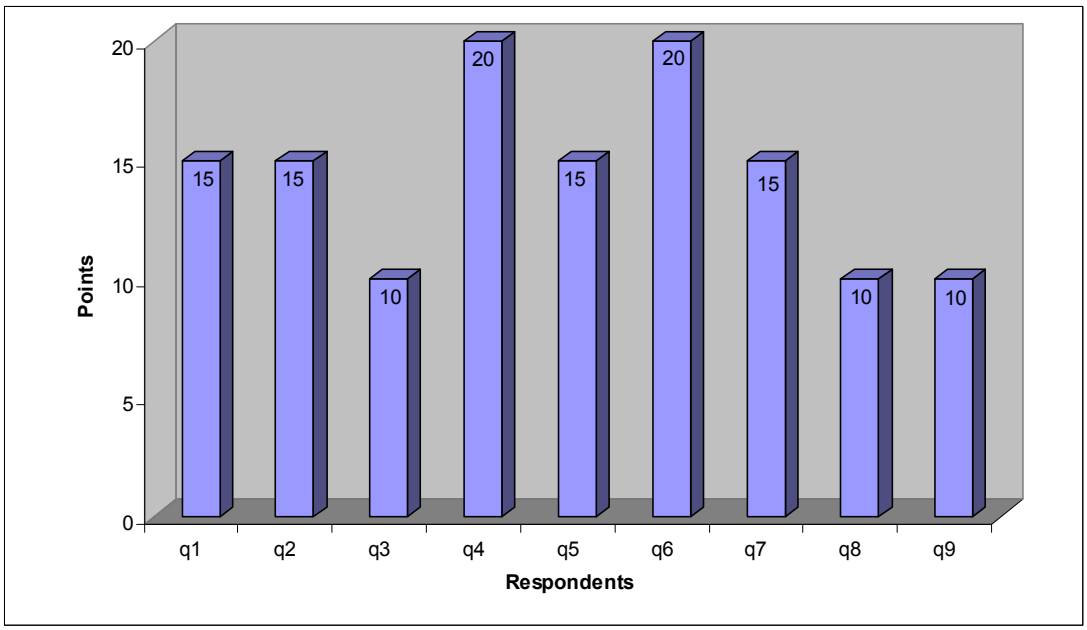



**Figure D.4** The respondents' prioritization of the usage group

Figure D.4 illustrates how each respondent prioritized of the models group.

## D.5 Model Driven Architecture group

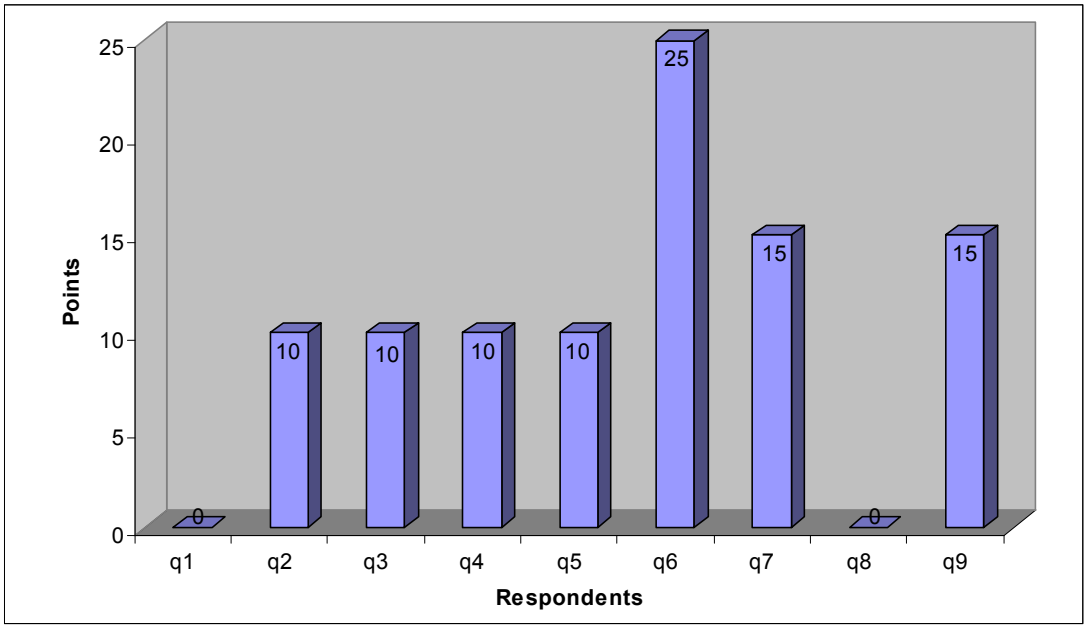



**Figure D.5** The respondents' prioritization of the MDA group

FigureD.5 illustrates how each respondent prioritized of the MDA group.

## D.6 Testing group

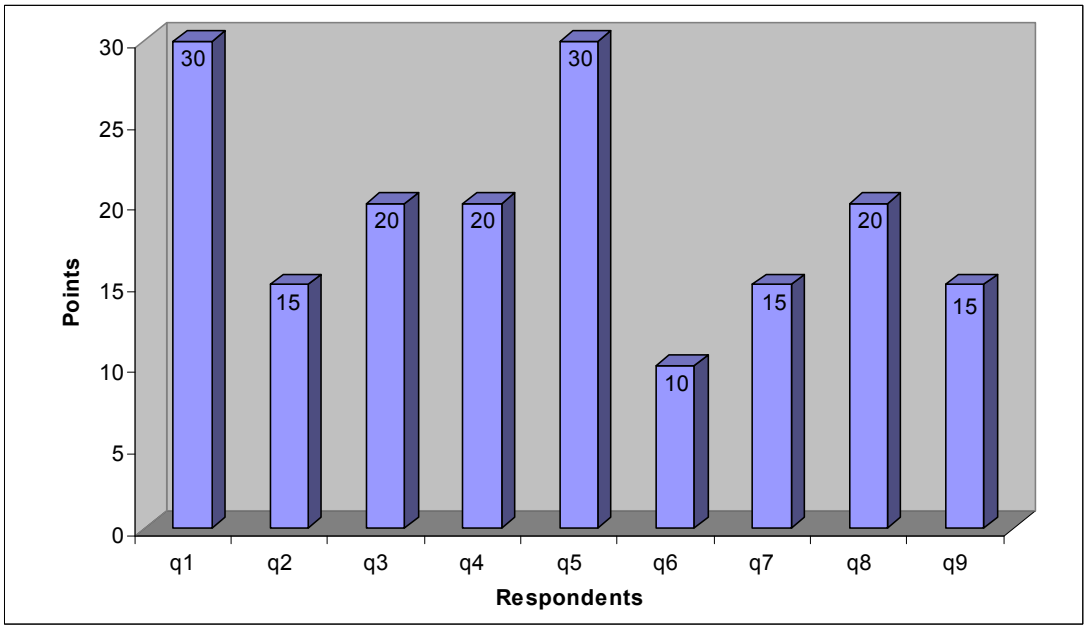



**Figure D.6** The respondents' prioritization of the testing group

FigureD.7illustrates how each respondent prioritized of the testing group.

# D.7 Criteria prioritization

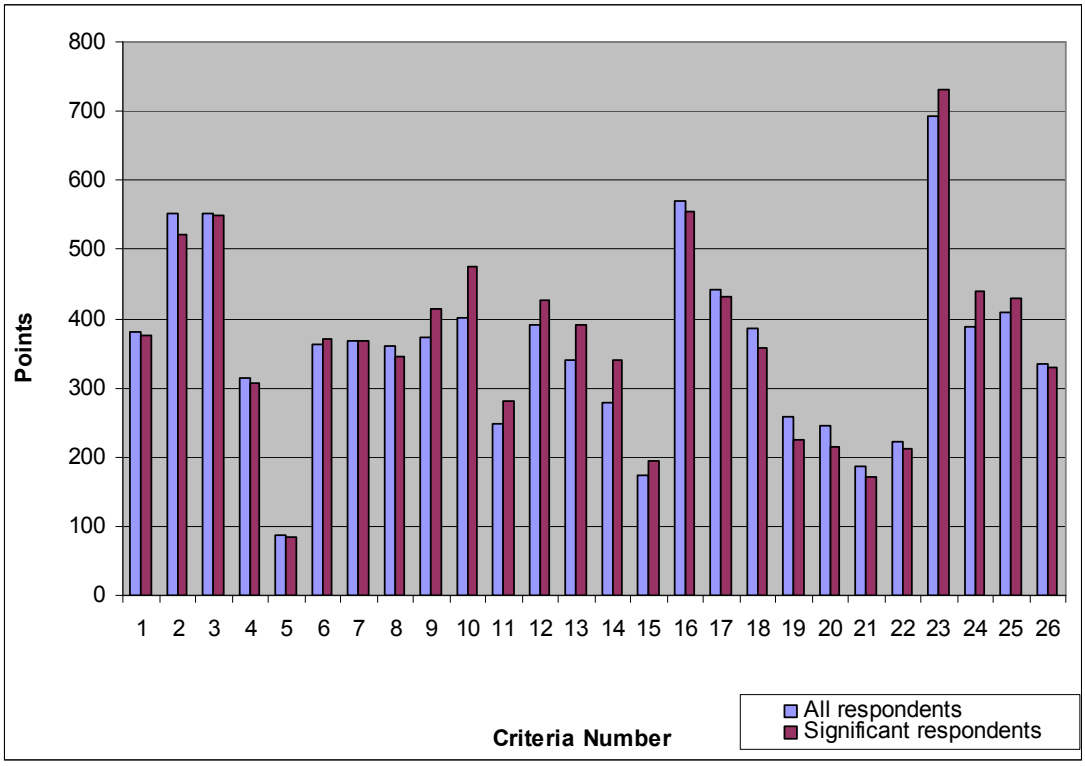



**Figure D.7** Criteria Prioritization

Figure D.7 illustrates the average point for each criteria.

# APPENDIX E – CHI-SQUARE AND SIGNIFICANCE LEVEL

Appendix E includes all the information for how the chi-square and significance level was calculated. Please see chapter 6.3.4.

## E.1 Chi-square and significance level for each pair

| 1 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| **Tester Q1** | 5 | 25 | 25 | 15 | 0 | 30 |
| **Tester Q2** | 20 | 10 | 30 | 15 | 10 | 15 |
| **Chi-square** | **24,83** | | | | | |
| **p-value** | **<= 0,001** | | | | | |

| 2 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| **Tester Q1** | 5 | 25 | 25 | 15 | 0 | 30 |
| **Designer Q3** | 30 | 20 | 10 | 10 | 10 | 20 |
| **Chi-square** | **37,84** | | | | | |
| **p-value** | **<= 0,001** | | | | | |

| 3 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| **Tester Q1** | 5 | 25 | 25 | 15 | 0 | 30 |
| **Designer Q5** | 10 | 20 | 15 | 15 | 10 | 30 |
| **Chi-square** | **14,72** | | | | | |
| **p-value** | **<= 0,025** | | | | | |

| 4 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| **Tester Q1** | 5 | 25 | 25 | 15 | 0 | 30 |
| **Designer Q6** | 15 | 15 | 15 | 20 | 25 | 10 |
| **Chi-square** | **45,71** | | | | | |
| **p-value** | **<= 0,001** | | | | | |

| 5 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| **Tester Q1** | 5 | 25 | 25 | 15 | 0 | 30 |
| **Designer Q7** | 15 | 20 | 20 | 15 | 15 | 15 |
| **Chi-square** | **26,11** | | | | | |
| **p-value** | **<= 0,001** | | | | | |

| 6 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| **Tester Q1** | 5 | 25 | 25 | 15 | 0 | 30 |
| **Designer Q8** | 25 | 20 | 25 | 10 | 0 | 20 |
| **Chi-square** | **17,88** | | | | | |
| **p-value** | **<= 0,01** | | | | | |

| 7 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| **Tester Q1** | 5 | 25 | 25 | 15 | 0 | 30 |
| **Designer Q9** | 5 | 25 | 30 | 10 | 15 | 15 |
| **Chi-square** | 18,7 | | | | | |
| **p-value** | **<= 0,01** | | | | | |

| 8 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Tester Q1 | 5 | 25 | 25 | 15 | 0 | 30 |
| Coordinator Q4 | 15 | 20 | 15 | 20 | 10 | 20 |
| Chi-square | 18,12 | | | | | |
| p-value | <= 0,01 | | | | | |

| 9 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Tester Q2 | 20 | 10 | 30 | 15 | 10 | 15 |
| Designer Q3 | 30 | 20 | 10 | 10 | 10 | 20 |
| Chi-square | 17,04 | | | | | |
| p-value | <= 0,01 | | | | | |

| 10 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Tester Q2 | 20 | 10 | 30 | 15 | 10 | 15 |
| Designer Q5 | 10 | 20 | 15 | 15 | 10 | 30 |
| Chi-square | 16,67 | | | | | |
| p-value | <= 0,01 | | | | | |

| 11 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Tester Q2 | 20 | 10 | 30 | 15 | 10 | 15 |
| Designer Q6 | 15 | 15 | 15 | 20 | 25 | 10 |
| Chi-square | 14,86 | | | | | |
| p-value | <= 0,025 | | | | | |

| 12 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Tester Q2 | 20 | 10 | 30 | 15 | 10 | 15 |
| Designer Q7 | 15 | 20 | 20 | 15 | 15 | 15 |
| Chi-square | 7,04 | | | | | |
| p-value | <= 1 | | | | | |

| 13 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Tester Q2 | 20 | 10 | 30 | 15 | 10 | 15 |
| Designer Q8 | 25 | 20 | 25 | 10 | 0 | 20 |
| Chi-square | 16,05 | | | | | |
| p-value | <= 0,01 | | | | | |

| 14 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Tester Q2 | 20 | 10 | 30 | 15 | 10 | 15 |
| Designer Q9 | 5 | 25 | 30 | 10 | 15 | 15 |
| Chi-square | 17,4 | | | | | |
| p-value | <= 0,01 | | | | | |

| 15 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Tester Q2 | 20 | 10 | 30 | 15 | 10 | 15 |
| Coordinator Q4 | 15 | 20 | 15 | 20 | 10 | 20 |
| Chi-square | 10,47 | | | | | |
| p-value | <= 0,10 | | | | | |

| 16 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q3 | 30 | 20 | 10 | 10 | 10 | 20 |
| Designer Q5 | 10 | 20 | 15 | 15 | 10 | 30 |
| Chi-square | 14 | | | | | |
| p-value | <= 0,025 | | | | | |

| 17 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q3 | 30 | 20 | 10 | 10 | 10 | 20 |
| Designer Q6 | 15 | 15 | 15 | 20 | 25 | 10 |
| Chi-square | 19,8 | | | | | |
| p-value | <= 0,01 | | | | | |

| 18 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q3 | 30 | 20 | 10 | 10 | 10 | 20 |
| Designer Q7 | 15 | 20 | 20 | 15 | 15 | 15 |
| Chi-square | 11.04 | | | | | |
| p-value | <= 0,10 | | | | | |

| 19 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q3 | 30 | 20 | 10 | 10 | 10 | 20 |
| Designer Q8 | 25 | 20 | 25 | 10 | 0 | 20 |
| Chi-square | 16,88 | | | | | |
| p-value | <= 0,01 | | | | | |

| 20 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q3 | 30 | 20 | 10 | 10 | 10 | 20 |
| Designer Q9 | 5 | 25 | 30 | 10 | 15 | 15 |
| Chi-square | 30,12 | | | | | |
| p-value | <= 0,001 | | | | | |

| 21 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q3 | 30 | 20 | 10 | 10 | 10 | 20 |
| Coordinator Q4 | 15 | 20 | 15 | 20 | 10 | 20 |
| Chi-square | 9,33 | | | | | |
| p-value | <= 0,10 | | | | | |

| 22 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q5 | 10 | 20 | 15 | 15 | 10 | 30 |
| Designer Q6 | 15 | 15 | 15 | 20 | 25 | 10 |
| Chi-square | 18,85 | | | | | |
| p-value | <= 0,01 | | | | | |

| 23 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q5 | 10 | 20 | 15 | 15 | 10 | 30 |
| Designer Q7 | 15 | 20 | 20 | 15 | 15 | 15 |
| Chi-square | 7,71 | | | | | |
| p-value | <= 0,20 | | | | | |

| 24 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q5 | 10 | 20 | 15 | 15 | 10 | 30 |
| Designer Q8 | 25 | 20 | 25 | 10 | 0 | 20 |
| Chi-square | 21,93 | | | | | |
| p-value | <= 0,001 | | | | | |

| 25 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q5 | 10 | 20 | 15 | 15 | 10 | 30 |
| Designer Q9 | 5 | 25 | 30 | 10 | 15 | 15 |
| Chi-square | 14,22 | | | | | |
| p-value | <= 0,025 | | | | | |

| 26 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q5 | 10 | 20 | 15 | 15 | 10 | 30 |
| Coordinator Q4 | 15 | 20 | 15 | 20 | 10 | 20 |
| Chi-square | 3,71 | | | | | |
| p-value | <= 1 | | | | | |

| 27 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q6 | 15 | 15 | 15 | 20 | 25 | 10 |
| Designer Q7 | 15 | 20 | 20 | 15 | 15 | 15 |
| Chi-square | 19,33 | | | | | |
| p-value | <= 0,01 | | | | | |

| 28 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q6 | 15 | 15 | 15 | 20 | 25 | 10 |
| Designer Q8 | 25 | 20 | 25 | 10 | 0 | 20 |
| Chi-square | 37,38 | | | | | |
| p-value | <= 0,001 | | | | | |

| 29 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q6 | 15 | 15 | 15 | 20 | 25 | 10 |
| Designer Q9 | 5 | 25 | 30 | 10 | 15 | 15 |
| Chi-square | 19,33 | | | | | |
| p-value | <= 0,01 | | | | | |

| 30 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q6 | 15 | 15 | 15 | 20 | 25 | 10 |
| Coordinator Q4 | 15 | 20 | 15 | 20 | 10 | 20 |
| Chi-square | 10,48 | | | | | |
| p-value | <= 0,10 | | | | | |

| 31 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q7 | 15 | 20 | 20 | 15 | 15 | 15 |
| Designer Q8 | 25 | 20 | 25 | 10 | 0 | 20 |
| Chi-square | 19,77 | | | | | |
| p-value | <= 0,01 | | | | | |

| 32 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q7 | 15 | 20 | 20 | 15 | 15 | 15 |
| Designer Q9 | 5 | 25 | 30 | 10 | 15 | 15 |
| Chi-square | 8,56 | | | | | |
| p-value | <= 0.20 | | | | | |

| 33 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q7 | 15 | 20 | 20 | 15 | 15 | 15 |
| Coordinator Q4 | 15 | 20 | 15 | 20 | 10 | 20 |
| Chi-square | 3,14 | | | | | |
| p-value | <= 1 | | | | | |

| 34 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q8 | 25 | 20 | 25 | 10 | 0 | 20 |
| Designer Q9 | 5 | 25 | 30 | 10 | 15 | 15 |
| Chi-square | 30,06 | | | | | |
| p-value | <= 0,001 | | | | | |

| 35 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q8 | 25 | 20 | 25 | 10 | 0 | 20 |
| Coordinator Q4 | 15 | 20 | 15 | 20 | 10 | 20 |
| Chi-square | 18,33 | | | | | |
| p-value | <= 0,01 | | | | | |

| 36 | TCG | Integration | Usage | Models | MDA | Testing |
|---|---|---|---|---|---|---|
| Designer Q9 | 5 | 25 | 30 | 10 | 15 | 15 |
| Coordinator Q4 | 15 | 20 | 15 | 20 | 10 | 20 |
| Chi-square | 15,6 | | | | | |
| p-value | <= 0,01 | | | | | |

## E.2 Significant and non significant pairs

**Significant pairs**

| | | | |
|---|---|---|---|
| 12 | <= 0,001 | 36 | <= 0,010 |
| 13 | <= 0,001 | 38 | <= 0,010 |
| 14 | <= 0,010 | 39 | <= 0,001 |
| 15 | <= 0,025 | 48 | <= 0,010 |
| 16 | <= 0,001 | 49 | <= 0,010 |
| 17 | <= 0,001 | 56 | <= 0,010 |
| 18 | <= 0,010 | 58 | <= 0,001 |
| 19 | <= 0,010 | 59 | <= 0,025 |
| 23 | <= 0,010 | 67 | <= 0,010 |
| 25 | <= 0,010 | 68 | <= 0,001 |
| 26 | <= 0,025 | 69 | <= 0,010 |
| 28 | <= 0,010 | 78 | <= 0,010 |
| 29 | <= 0,010 | 89 | <= 0,001 |
| 35 | <= 0,025 | | |

**Non-significant pairs**

| | |
|---|---|
| 24 | <= 0,100 |
| 27 | <= 1,000 |
| 34 | <= 0,100 |
| 37 | <= 0,100 |
| 45 | <= 1,000 |
| 46 | <= 0,100 |
| 47 | <= 1,000 |
| 57 | <= 0,200 |
| 79 | <= 0.200 |

The first numbers represents the table numbers in E.1 while the second number is the significance level.