Quality Assessment and Quality Improvement for UML Models

Dissertation
zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

vorgelegt von
Akhtar Ali Jalbani
aus Larkana (Sindh) - Pakistan

Göttingen 2011

Referent:	Prof. Dr. phil. nat. Jens Grabowski, Georg-August Universität Göttingen.
Koreferent:	Prof. Dr. Helmut Neukirchen, University of Iceland.

Tag der mündlichen Prüfung: 28 Februar 2011

Abstract

Models defined using the Unified Modeling Language (UML) are nowadays a common part of software documentation, specification and sometimes even implementation. However, there is a broad variety in how UML is used. Reasons can be found, for example, in the lack of generally accepted modeling norms and guidelines, in the semi-formal semantics of UML, or in the complexity of the language. In practice, these factors inevitably lead to quality problems in UML models that need to be addressed.

In this research, we present an integrated continuous quality assessment and improvement approach for UML models. The approach is based on a novel quality model for UML models and the exemplary instantiation of the quality model. A prototypical tool that implements the quality assessment and improvement approach for UML models has been developed. To demonstrate the applicability of our approach, a case study based on a UML practical course was conducted. UML models developed in the practical course were evaluated with our prototypical tool. The quality assessment and improvement results of the case study indicated that our approach is practically feasible and applicable.

Zusammenfassung

UML-Modelle sind heutzutage Teil der Dokumentation, der Spezifikation und manchmal sogar der Implementierung von Softwaresystemen. Allerdings kann UML sehr unterschiedlich benutzt werden. Die Gründe hierfür sind vielfältig. So fehlen zum Beispiel allgemein akzeptierte Normen und Richtlinien für die Verwendung von UML. Des Weiteren ist die Sprache UML sehr komplex und Teile der Sprache besitzen nur eine semi-formale Semantik. All diese Faktoren führen zu Qualitätsproblemen bei UML-Modellen, die untersucht und bearbeitet werden müssen.

In der vorliegenden Arbeit wird ein Verfahren für eine integrierte und kontinuierliche Qualitätsbewertung und -verbesserung von UML-Modellen vorgestellt. Das Verfahren basiert auf einem neuen Qualitätsmodell für UML-Modelle, dessen exemplarische Instanziierung in der Arbeit beschrieben wird. Es wurde ein prototypisches Werkzeug entwickelt, mit dessen Hilfe die Qualitätsbeurteilung und die -verbesserung von UML-Modellen automatisiert durchgeführt werden kann.

Zum Nachweis der Anwendbarkeit des vorgestellten Verfahrens wurde eine Fallstudie im Rahmen eines UML-Praktikums durchgeführt. Die Qualität der während des Praktikums entwickelten UML-Modelle wurde kontinuierlich bewertet und identifizierte Qualitätsprobleme mussten von den Teilnehmern des Praktikums fortlaufend beseitigt werden. Die Ergebnisse der Fallstudie unterstreichen die praktische Anwendbarkeit und das hohe Potential des vorgestellten Verfahrens zur automatisierten Qualitätsbewertung und -verbesserung von UML-Modellen.

Acknowledgments

First of all, I would like to thank to almighty Allah, who gave me courage and patience in the entire duration of the research project. I am thankful to my supervisor Professor Dr. Jens Grabowski for his continuous guidance, help and support throughout my research work. He was always kind to listen patiently to my raw ideas and gave shapes to those raw ideas with his wide vision and experience.

I would like to thank my thesis committee members Prof. Dr. Stephan Waack, Prof. Dr. Helmut Neukirchen, Prof. Dr. Carsten Damm, Prof. Dr. Wolfgang May and Prof. Dr. Winfried Kurth for taking valuable time from their busy schedule to provide valuable suggestions. I specially thank to Prof. Dr. Helmut Neukirchen for our telephone conferences to provide feedback on the contents of my thesis from Iceland. I would like to thank to my colleagues Dr. Wafi Dahman, Dr. Benjamin Zeiss, Dr. Edith Werner, Philip Makedonski, Thomas Rings, Patrick Harms, Xin Jin and Gunnar Krull who all helped me a lot during my research period, and also I would like to thank Annette Kadziora for translation of German into English, whenever I needed. I am thankful to Dr. Benjamin Zeiss for his valuable suggestions during discussions on the Quality Model, and I also want to thank Dr. Edith Werner for managing the UML practical course and providing feedback about the case study.

I would like to express my gratitude to the following people who I think should be mentioned here: Ayaz Ali, Zeeshan Hassan, Dr. Nazeer Hussain Kalhoro, Mustafa Junejo, Ali Raza Shah and Mansoor Hyder Depar.

Last but not least, I would like to thank my parents for their continuous support and for making it possible for me to pursue my professional goals. I would like to thank my dear wife Dr. Aneela Yasmin for her understanding and continuous support of my research endeavors. I could not forget the patience of my beloved children Yomna and Ali Saad during my research work.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

AGG Attributed Graph Grammar

AST Abstract Syntax Tree

ATL ATLAS Transformation Language

CIM Computation Independent Model

CMOF Complete Meta Object Facility

CWM Common Warehouse Model

DSL Domain Specific Language

EMF Eclipse Modeling Framework

EMOF Essential Meta Object Facility

EmpAnADa Empirical Analysis of Architecture and Design Quality Project

EMP Eclipse Modeling Project

EMT Eclipse Modeling Technology

FCM Factor-Criteria-Metrics

GQM Goal Question Metric

M2M Model-to-Model

M2T Model-to-Text

MDA Model Driven Architecture

MDD Model Driven Development

MDE Model Driven Engineering

MDSD Model Driven Software Development

MOF Meta Object Facility

MWE Modeling Workflow Engine

OCL Object Constraint Language

OMG Object Management Group

oAW openArchitectureWare

PIM Platform Independent Model

PSM Platform Specific Model

SDL Forum System Design Languages Forum

QVT Query/View/Transformation

RUP Rational Unified Process

TTCN-3 Testing and Test Control Notation version 3

UML Unified Modeling Language

XMI XML Metadata Interchange

XML Xtensible Mark-up Language

xUML Executable UML

Chapter 1

Introduction

When the Unified Modeling Language (UML) [72] was introduced in the late 90s, its primary application was to support the communication among software engineers. UML provided a standardized way to describe software artifacts. There were only limited and rarely successful attempts to reuse UML artifacts for the implementation phase of a software product. Therefore, the notion of quality regarding these early UML models primarily affected questions on the diagram layouts, i.e., the aesthetics and understandability of the diagrams.

With the advent of methodologies like Model Driven Architecture (MDA) [66] and the Model Driven Software Development (MDSD) [85] and platforms like Eclipse Modeling Technology (EMT) [20], a new wave of code generation tools emerged [3, 21, 86]. The focus of these technologies is on the design of abstract models that avoid implementation details, which can automatically be transformed into code by means of intelligent code generators. As a result of EMT, we are confronted with new technologies such as the Eclipse Modeling Framework (EMF) [86]. The EMF provides a practical and mature software framework that simplifies intelligent and customizable code generation.

This new situation and its associated demands, also changes the quality requirements on UML models. On the one hand, software development still deals with UML for communication purposes and on the other hand, the intention is to reuse these efforts to speed up the implementation phase. However, these models may still be expressed on different levels of abstraction and completeness.

Quality problems in the design of UML models that are used for code generation may lead to problems in the generated code. To overcome these type of problems, rules and guidelines for the development of a UML models could be

employed. However, a fixed, generally accepted catalog of modeling guidelines and constraints does not exist yet. The success of UML can be seen by its use in the industry [80]. The widespread usage of UML indicates that modeling is one of the key elements in many software development projects. The evolution of MDA and its associated technologies drastically altered software development processes. There is no restriction regarding a specific programming language, instead the focus lies on developing UML models based on the requirements and translating them into an executable software system in general.

This work has been inspired by research about the quality engineering for the Testing and Test Control Notation version 3 (TTCN-3) [59, 60, 61, 62, 93]. That approach is adopted to UML models, to investigate whether an integrated continuous quality assessment and improvement approach for UML models is possible and feasible.

1.1 Continuous Quality Assessment and Improvement Process for UML Models

The main objective is to setup a framework to measure the quality for UML models. The framework allows modeler's to establish a quality assessment and improvement process. The working process of this approach is illustrated in Figure 1.1, where the UML models are considered as the key artifact of this process.

The quality assessment is performed by applying metrics, rules and guidelines to UML models. The violations of these rules are considered to be issues of the UML model. The detected issues are then removed to improve the quality of the UML model based on refactoring.

The process described above is performed in a continuous way by observing the quality assessment results (i.e., detected issues) and taking actions (i.e., removing issues) to improve the quality of models. This cycle is performed repeatedly until the quality assessment results can be considered satisfactory.

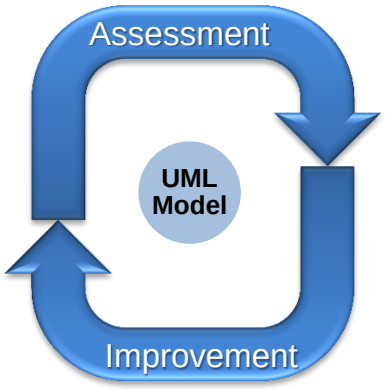


Figure 1.1: Continuous Quality Assessment and Improvement Process

1.2 Contributions

The contributions of this research are as follows and are also illustrated in Figure 1.2.

1. A novel quality model for UML, which examines the different quality characteristics for UML models based on an inclusion relationship between three types of model completeness types.

2. A prototypical instantiation of UML quality model for three quality attributes is presented. In this instantiation, we present how the Goal Question Metric (GQM) approach is applied to determine the rules and guidelines for each quality attribute. Furthermore, we also present ratio metrics to compare the different kinds of models.

3. A prototype tool for quality assessment and improvement for UML models, that provides a methods for both issue detection and issue removal.

4. A Case study of our approach has been performed during a UML practical course offered to the students. A bakery system is used as an exemplary model in the UML practical course. The results of the case study show the applicability and the feasibility of our approach.

1.3 Thesis Structure

The structure of this thesis is as follows. In Chapter 2, we discuss the foundations of this thesis that are needed across all chapters. Chapter 3 presents a state-of-the-art analysis of the most important work related to the quality of UML models.

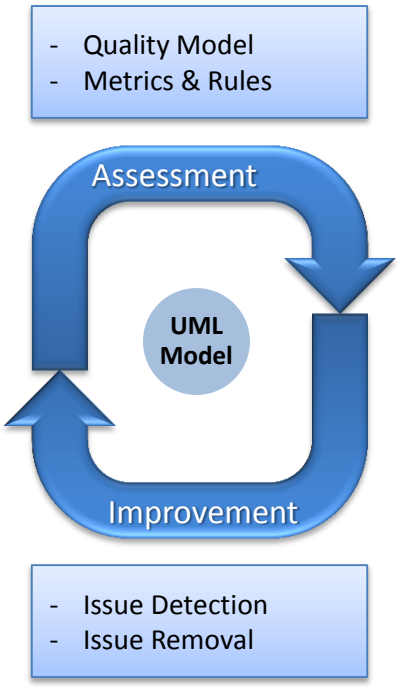


Figure 1.2: Contributions for Continuous Quality Assessment and Improvement Process

A novel quality model for UML model including its prototypical instantiation is proposed in Chapter 4. The implementation of a prototype tool and OCL based rules and guidelines are described in Chapter 5. A proof-of-concept case study is presented in Chapter 6. A summary of the work and directions for future research are discussed in Chapter 7.

Chapter 2

Foundations

This chapter presents the foundations on which this thesis is built. We present an overview of UML and associated technologies, afterword's, the UML diagrams are introduced and exemplified using the example of a UML model for a bakery system. The example of a bakery system is given in the foundation because it is used through out the thesis. The brief description of software quality and quality models is given. For software quality assessment and improvement, we discuss software metrics, code smells and refactorings.

2.1 UML and Associated Technologies

UML [72] is a general purpose modeling language, which provides a collection of graphical notations [29]. It provides only descriptive rules and graphical notations but no official way to model the structure and behavior of a system. Hence, UML leaves it open to the modeler to select an appropriate UML notation to describe their systems.

UML is classified into structural and behavioral diagrams. The structural diagrams are used to model the software structure, for example, classes, components and their relationships. The behavioral diagrams are used to model the behavior of the software.

Since the release of UML 1.0 in 1997, UML evolved adding more notations to the language and this makes the language more complex. The current UML version is 2.3 [74]. Our research is based on the UML 2.0 [72].

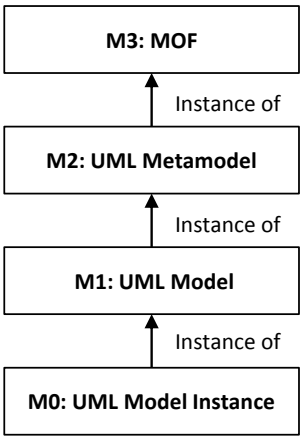


Figure 2.1: The UML Architecture

2.1.1 The UML Architecture

The UML architecture is composed of four layers (Figure 2.1). The M3 layer, the foundation of UML, is called the Meta Object Facility (MOF) [69]. In essence, MOF is a language that is used to model itself as well as other models or metamodels. In the context of UML, the MOF is used to define the UML metamodel (the M2 layer). MOF can be considered a meta-metamodel in this case. The MOF is used to specify the UML metamodel that consists of the *Infrastructure* [71] and *Superstructure* [72] standards. These standards define the abstract syntax of the language, i.e., basic UML modeling concepts, attributes, relationships, as well as the semantics of each modeling concept. The M1 layer is again an instance of the M2 layer. On the M1 layer, we find those models that we typically create for requirements or design specifications. The instance of a UML model is then finally found on the M0 layer, which describes instantiated objects.

The UML models we deal with everyday are typically the ones found on the M1 layer, i.e., we create instances of the UML metamodel. One common way to create such a model is to use the graphical notation provided by the UML Superstructure standard.

2.1.2 UML Models vs. UML Diagrams

It is crucial to understand that a UML model and a UML diagram are two different things. It is easy to draw a set of diagrams conforms to the UML notation on paper. However, on paper these cannot be validated, transformed, or used for code generation. Even if we transfer our diagrams as they are into a digital form, they are missing important pieces of information that is not part of the diagrams, for example, how the diagrams relate to each other and where the

definitions to model references can be found. If the graphical notation is used to create a UML model (i.e., by using a UML tool), each diagram represents only a partial view of the complete model. Thus, a UML model may be described by multiple diagrams or no diagram at all, a UML model may still contain all elements we know from the commonly used graphical notation without including a single diagram. However, there is no common and unified notation that can represent a UML model completely, but attempts to solve this problem exist, e.g., TextUML [1]. One way, although not entirely human-readable, to represent a complete UML model is the XML Metadata Interchange (XMI) format [67] which is, however, an exchange format rather than a useful notation for modeling.

To illustrate the difference between a model and diagrams, we present a simple specification of a weather information system in Figure 2.2. At the top of the figure, we have the graphical notation of a UML model consisting of a class and a sequence diagram. At the bottom part of the figure, we present the XMI representation of the same model. Figure 2.2 illustrates two things. First, a complete model can represent multiple diagrams may be part of a single UML model. In this case, the model contains the definitions from both class diagram and the sequence diagram. Second, the XMI representation explicitly references the previously defined UML classes. Such an explicit reference is not possible when we deal with diagrams in UML notation (that are created using pencil and paper or a diagramming tool) rather than UML models. The UML diagrams used in this thesis are described in Section 2.2.2.

2.1.3 Object Constraint Language (OCL)

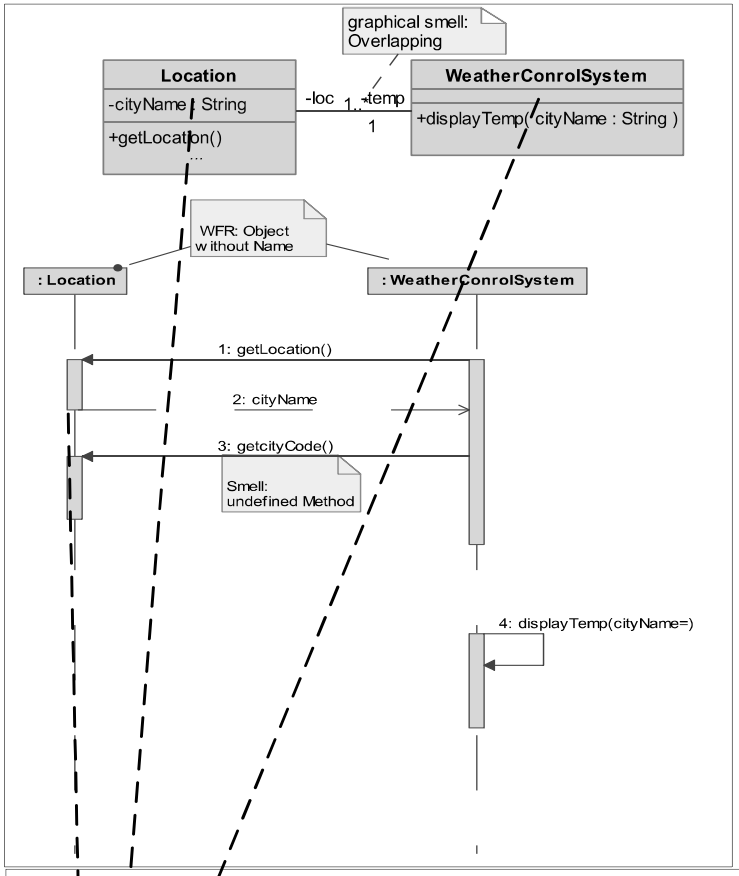
OCL [70] is a UML standard maintained by Object Management Group (OMG) [73]. The information, which can not be expressed with the graphical means by UML that can be expressed by OCL. OCL syntax is similar to programming language and the syntax of UML. It can be used in two ways: 1) as a constraint language; 2) as a query language.

A constraint is basically a restriction on one or more values of the model. These values are valid if the condition specified in constraint holds in the modeled system. A simple OCL constraint is shown in Listing 2.1. Line 1 defines that the constraint is on the class *Company* and Line 2 defines that the value of the attribute *noEmployees* must be less than or equal to 50.

```
1 context Company
2 inv: self.noEmployees <= 50
```

Listing 2.1: Sample OCL Constraint

OCL can also be used as a query language. When a query is executed on the model, the result of the query does not change the state of the system. The query




```
<?xml version="1.0" encoding="UTF-8"?>
 <uml:Model xmi:id="M1" name="Data">

 <packagedElement xmi:type="uml:Class" xmi:id="C1" name="Location">
    <ownedAttribute xmi:id="A1" name="cityName" visibility="private">
     </ownedAttribute>
    <ownedAttribute xmi:id="assEndC2" name="temp" visibility="private" type="C2" association="Association1">
     <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="littempUperValue" name="" value="1"/>
     <lowerValue xmi:type="uml:LiteralInteger" xmi:id="littempLowerValue" name="" value="1"/>
    </ownedAttribute>
    <ownedOperation xmi:id="M1" name="getLocation" visibility="public"/>
   </packagedElement>
   <packagedElement xmi:type="uml:Class" xmi:id="C2" name="WeatherConrolSystem">
    <ownedAttribute xmi:id="assEndC1" name="loc" visibility="private" type="C1" association="Association1">
     <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="litlocUperValue" name="" value="*"/>
     <lowerValue xmi:type="uml:LiteralInteger" xmi:id="litlocLowerValue" name="" value="1"/>
    </ownedAttribute>
    <ownedOperation xmi:id="M2" name="displayTemp" visibility="public">
     <ownedParameter xmi:id="P1" name="cityName" visibility="public">
     </ownedParameter>
    </ownedOperation>
   </packagedElement>
   <packagedElement xmi:type="uml:Association" xmi:id="Association1" name="result" memberEnd="assEndC1 assEndC2"/>

<packagedElement xmi:type="uml:Collaboration" xmi:id="collob1" name="WCS">
    <ownedBehavior xmi:type="uml:Interaction" xmi:id="Interaction1" name="WCS">
     <ownedAttribute xmi:id="Obj1" name="" visibility="private" type="C1"/>
     <ownedAttribute xmi:id="Obj2" name="" visibility="private" type="C2"/>
     <lifeline xmi:id="l1" name="" visibility="public" represents="Obj1" coveredBy="MO3 MO2 MO2Start MO2Finish"/>
     nish"/>
     <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="MO1" name="" visibility="public" covered="l2" message="SM1"/>
     sage="SM1"/>
     <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id="MO2Start" covered="l1" start="MO2" finish="MO2Finish"/>
     <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="MO2Finish" covered="l1" execution="MO2Start"/>
     sage="SM2"/>
     <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="MO4" name="" visibility="public" covered="l2" message="SM2"/>
     <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id="MO4Start" covered="l2" start="MO4" finish="MO4Finish"/>
     <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="MO4Finish" covered="l2" execution="MO4Start"/>
     <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="MO5" name="" visibility="public" covered="l2" message="SM3"/>
     sage="SM3"/>
     <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id="MO6Start" covered="l2" start="MO6" finish="MO6Finish"/>
       [.....]
     </message>
    </ownedBehavior>
</packagedElement>
</uml:Model></xmi:XMI>
```

Figure 2.2: Graphical and XMI Representation of a UML Model

simply returns a value or set of values. Listing 2.2 shows an OCL query on a UML model to to count the number of operations in a class. Line 1 the keyword *self* is used to refer to the contextual instance and *ownedOperation* is the property of the *Class* in a UML model and *size()* is the OCL function, which returns the size of the element. The OCL based rules and guidelines in Section 4.5 for *incomplete* and *complete* models are defined in the same style as described in Listing 2.2.

```
1 self.ownedOperation->size()
```

Listing 2.2: Sample OCL Query

In M2M transformation languages, OCL serves as a base language to query the model. The OCL expressions can be included in the model, or in the source code of one transformation language that supports OCL like syntax, for example, ATLAS Transformation Language (ATL), Query/View/Transformation (QVT) language and the Xtend language [21].

2.1.4 Rational Unified Process (RUP) and UML

The Rational Unified Process (RUP) [46, 64] is a software development process that executes projects in an iterative and incremental manner. A RUP compliant process delivers functionality in small increments as illustrated in Figure 2.3. Each increment builds upon the previous increment, and each increment is driven by use cases rather than being a construction of a subsystem.

There are several disciplines into which RUP is divided: business modeling, requirements, analysis and design, implementation, test, deployment, project management, environment and configuration and change management. Each discipline is expressed in terms of roles (who performs the task), activities (how they perform these tasks), and artifacts (what the activity achieves).

The main objective of the RUP is to estimate tasks and plan schedules by measuring the speed of iterations relative to their original estimates. The early iterations of the project are strongly focused on software architecture and the implementation of the product is not started until a firm architecture has been identified and tested.

UML notations are an integral part of the RUP [89]. It should be clarified that UML is not a process or a methodology but the software development process wrapped around the UML.

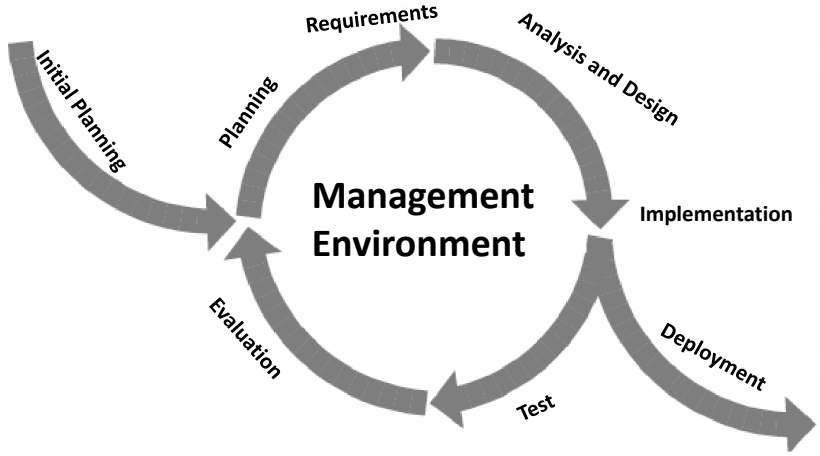


Figure 2.3: Rational Unified Process [64]

2.1.5 Model Driven Architecture (MDA)

The Model Driven Architecture (MDA) proposal [66] was initiated by the Object Management Group (OMG) in 2001. It provides an approach for the model driven development of software system. This proposal was made to resolve the middleware proliferation. The problem arises, when companies started to use middleware platform like CORBA, Java EE, .NET, COM+. With these middleware platforms, an interoperability with enterprises became the core issue. OMG managed the interoperability by providing MDA, which is a vendor and a middleware neutral solution [84]. Figure 2.4 shows a core architecture of MDA, that is based on UML, MOF.

Each model is independent of any middleware platform. Before thinking of any middleware platform, the first step is to construct the application based on a MDA using UML that should be platform independent. The second step is to convert the UML model built in the first step into the targeted specific platform, for example, CORBA, Java EE or .NET. Figure 2.4 shows these targeted platforms in a ring surrounding the core.

MDA approaches are independent of the target platform and focused only on the functionality and behavior of the system. It can also define system functionality by Platform Independent Model (PIM) and Platform Specific Model (PSM) by appropriate mapping using an appropriate Domain Specific Language (DSL). Figure 2.5 shows the process of model transformation from Computation Independent Model (CIM) to PIM, PIM to PSM and finally PSM to code. This process can also be carried out in reverse order for reverse engineering i.e., from code to model. The concepts of CIM, PIM and PSM are described in the next paragraph.

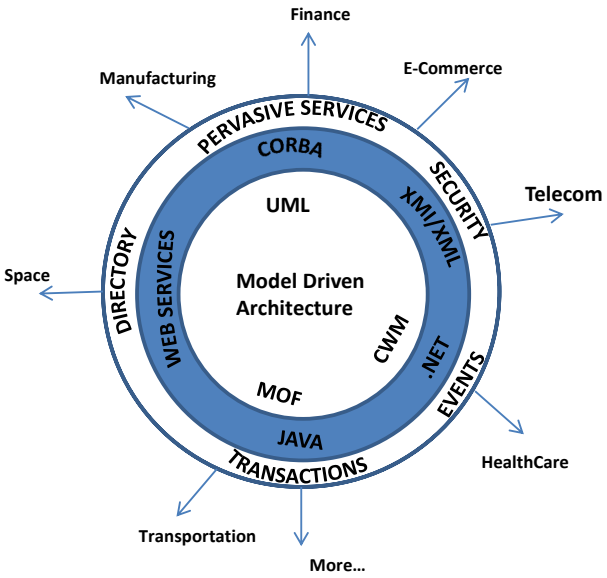


Figure 2.4: The Model Driven Architecture [66]

2.1.5.1 Computation Independent Model (CIM)

OMG described Computation Independent Model (CIM) in order to represent the application in its global environment. The main objective of CIM is to provide a link to the application client with other models.

2.1.5.2 Platform Independent Model (PIM)

The Platform Independent Model (PIM) models describe business behavior and functionalities of a system completely independent of the technical details or properties of the underlying technologies, for example, the operating system, programming languages, hardware and network performance. Hence, PIM presents the view of the system that is suitable for any type of execution platform.

2.1.5.3 Platform Specific Model (PSM)

The role of Platform Specific Model (PSM) is to ease code generation from the PIM through code models that fit the underlying execution platforms. UML profiles can be defined using PSM models, some profiles have been already defined for specific technologies, for example, the UML Profile for CORBA [75]. PSM allows the software designers to integrate platform-specific features into the model.

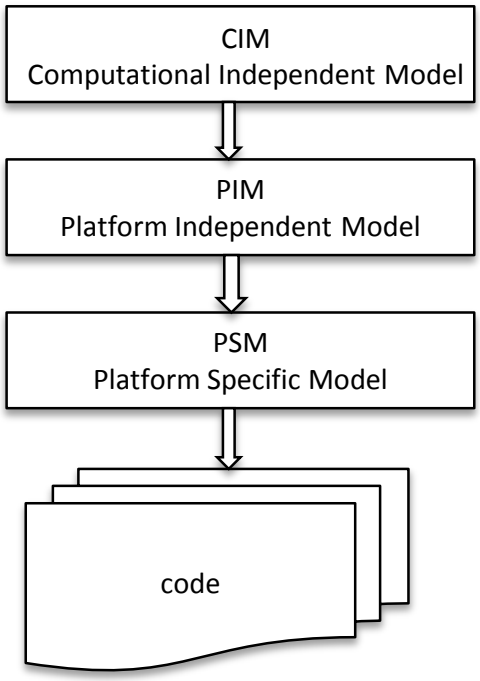


Figure 2.5: MDA Model Transformation Process

2.2 UML Model for a Bakery System

In this section, we introduce UML model for a bakery system. Using this model, we will clarify the general concept of UML diagrams. Additionally, model of a bakery system is the subject of the case study. Therefore, this example will help to understand the case study description and the quality assessment results described in Chapter 6.

2.2.1 Description of a Bakery System

The bakery software system handles all aspects of a bakery, including sales ordering, personal management and warehouse management of the products. The sales ordering system handles the selling of the bakery products. Personal management system manages the personal data of the employees. The warehouse system manages the inventory of the bakery products, and it also records the expiry date of each product. The bakery sales include several kinds of breads, rolls, cakes and pastries that vary every day. Seasonal offers include various cakes according to the season, such as strawberry cakes or christmas cakes.

The external actors of the system are shown at the top of Figure 2.6, where the three types of users can interact with the system. The *Administrator* is an actor who manages the software solutions and set privileges to the users of the system.

The external actors are *Salesman* and *Online Customer*. *Salesman* interacts with the system to handle customer orders and payments at the counter, where an *Online Customer* uses a web based system, where they can order, modify orders and make online payments with limited access to the system. The bakery system allows *Online Customer*, *Salesman*, and *Administrator* to access it as separate users with different privileges.

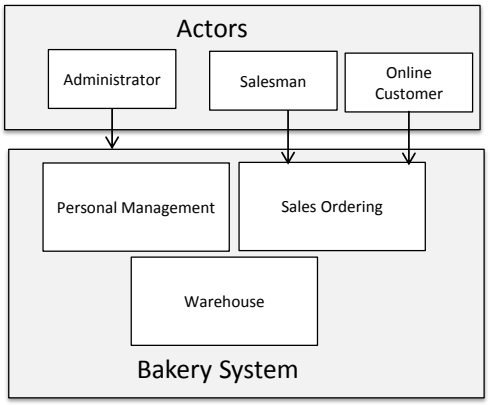


Figure 2.6: General Overview of the Bakery System Components

Existing users that are either *Salesman* or *Online Customer* are required to login to the system. New *Online Customers* are able to create a new account by registering using an online form. Once a *Salesman* or *Online Customer* are logged in, they will be directed to the sales ordering subsystem, where they can choose which product to order. After the *Online Customers* has decided the quantity of the selected item, it will be added to a cart. From the cart, the *Online Customer* can choose between two payments methods, either by credit card or by cash. Once the payment is successful, the order is executed by the system. *Administrator* can view the database, categorized into users, inventory and orders. In each category, *Administrators* can then perform various actions like add, edit, or delete entries.

2.2.2 Partial UML Model for the Bakery System

This section presents an overview of the diagrams used to develop the analysis model and design model for the bakery system.

2.2.2.1 Use Case Diagram

The three subsystems of the bakery software system are illustrated in Figure 2.6. In this section only the *Sales Ordering* subsystem is discussed. A use case diagram of this subsystem is visualized in Figure 2.7. There are three actors interacting

with the subsystem: *Salesman*, *Administrator* and an *Online Customer*. The *Salesman* interacts with different parts of the ordering process. *Salesman* interacts with the use case *Place Order* to place an order for the customer and the use case *Manage Orders* to manage the customer's orders. *Online Customer* interacts with the use case *Place Order* to place an order online and with the use case *Make Payment* to pay for the ordered item. *Make Payment* includes another use case *Choose Payment*, where *Online Customer* can choose a payment method. *Administrator* interacts with the use case *Manage Orders* to manage customer orders and with use case *Mange Customer Database* to manage the customer data base.

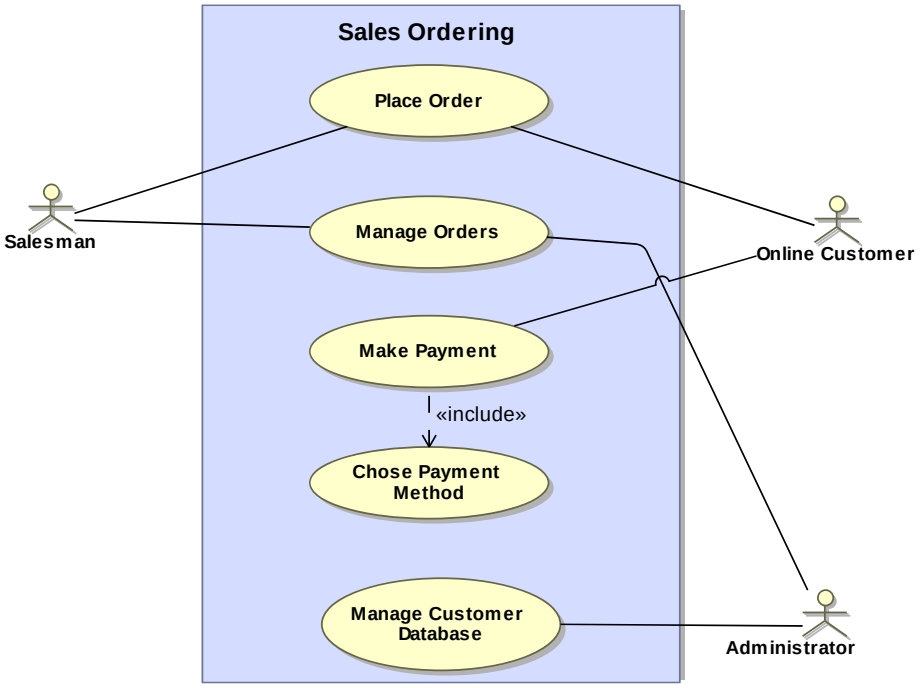


Figure 2.7: Use Case Diagram for Sales Ordering

2.2.2.2 Activity Diagram

Activity diagram are used to document the flow within a use case [89]. The flow can be sequential, branched or concurrent. Figure 2.8 shows the activity diagram for the *Make Payment* and *Choose Payment Method* use cases. The payment cost is calculated from the cart, where the ordered bakery products and their cost are listed. After the amount is verified, the user has to select a payment method. This functionality is provided by the *Choose Payment Method* use case.

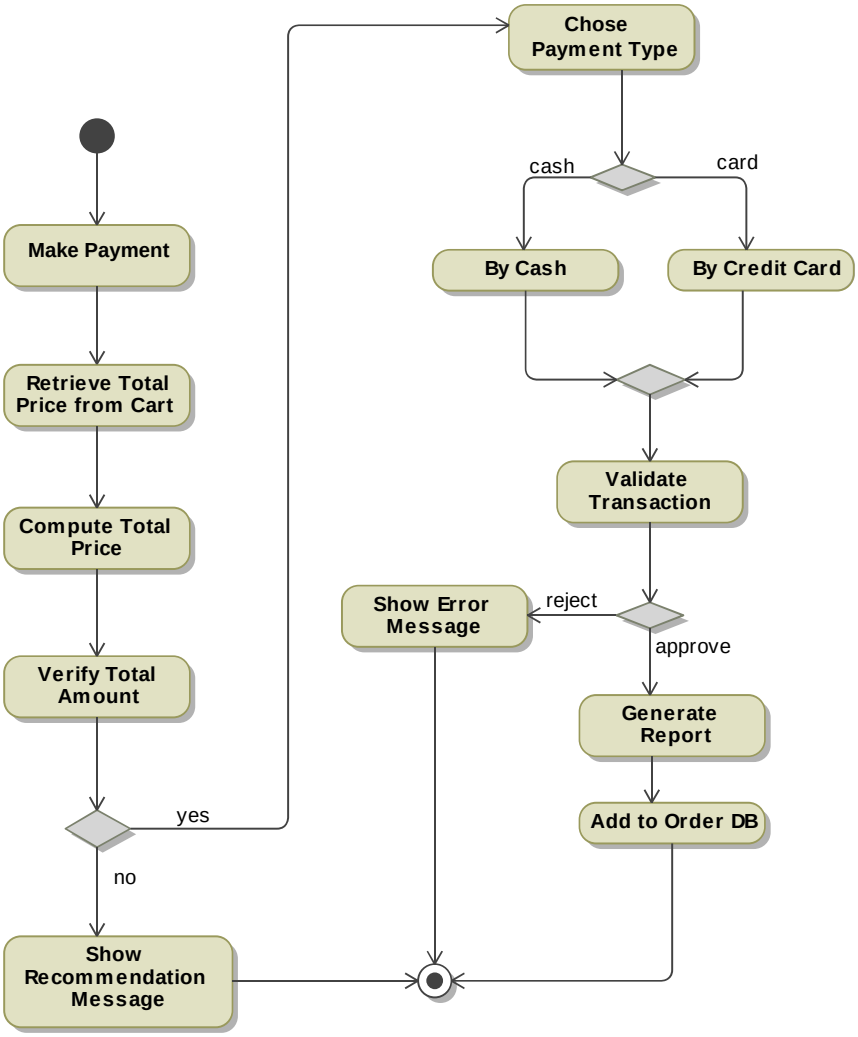


Figure 2.8: Activity Diagram for Make Payment

2.2.2.3 Class Diagram

Class diagrams show the static structure of classes of a system and the relationships between them. The relationships can be associations, generalizations, or aggregations. The class diagram for the *Sales Ordering* subsystem is illustrated in Figure 2.9. The figure shows how the different classes are associated with each other. Two types of associations are used in this diagram: composition and generalization. Figure 2.9 illustrate following classes for the *Sales Ordering subsystem*: *BakerySystem, BakerySystemDB, Customer, Payment, Cart and CreditCard.*

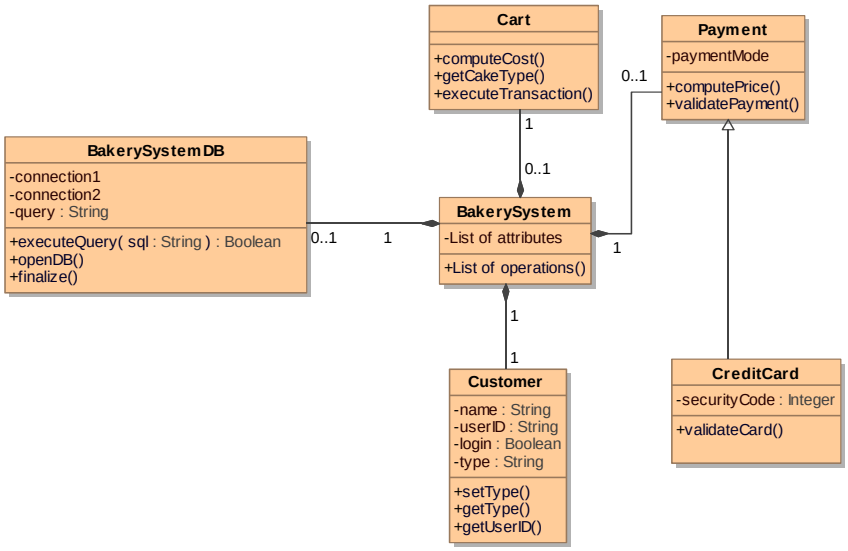


Figure 2.9: Class Diagram for the Bakery System

2.2.2.4 Sequence Diagram

Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. Sequence diagrams can also be used to model the behavior of use cases. Figure 2.10 refers to the use case *Place Order*, where the lifeline represents the actor and classes. The actor *Online Customer* interacts with the class *BakerySystem* and cthe lass *BakerySystem* interacts with the *BakerySystemDB*. Messages are represented by the arrows and every message is numbered. *Online Customer* selects the bakery item or items from the list of bakery products by invoking the *getOrder()* method for the selected bakery items. The class *BakerySystem* needs the actor to choose the payment method and *online Customer* invokes the method *getPayment()*, when the payment is made, the total amount is verified by the *BakerySystem*. To process the order *BakerySystemDB* class invoking *executeQuery* method for further action.

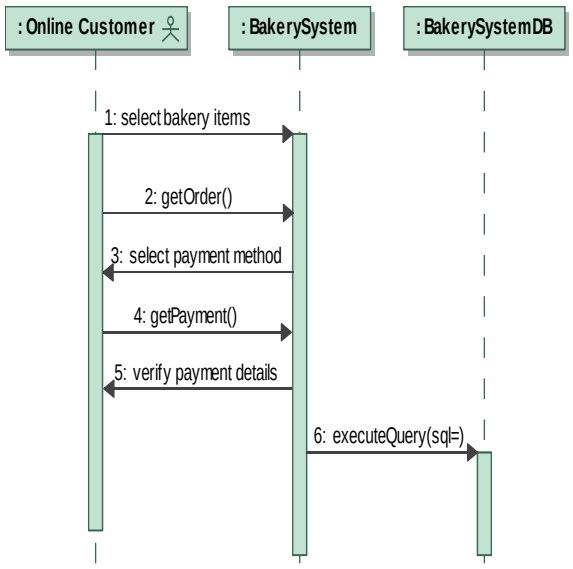


Figure 2.10: Sequence Diagram for Place Order

2.2.3 Relationship of the UML Diagrams

This section presents the relationship between the diagrams discussed in Section 2.2.2.

The analysis model of the bakery system provides a conceptual model, that focuses on the domain concepts, existing use cases and actors. The analysis model also describes the concepts (i.e., things, objects), association between concepts and attributes of the concepts. For analysis model following diagrams are used: use case diagram, activity diagram, sequence diagram and class diagram. The activity and sequence diagrams are used to model the use cases, while the class diagram presents the associated classes and their relationship.

The design model of the bakery system presents the blueprint for the developers to implement the software product. The goal of the design model is to provide a sufficient level of detail. The design model involves representation of both classes and objects depending upon the diagram. The connection between classes and their objects signify the basic relationships between classes as well as messaging between objects. The design model for the bakery system consists of class diagrams, sequence diagrams, activity diagrams and state machine diagrams as shown in Figure 2.11.

Implementation model is considered to be a set of diagrams from the design model to generate code manually or automatically. Not all diagrams are considered for

the code generation. Figure 2.11 shows the class and state machine diagrams to generate executable code. The implementation model is considered to be an executable model in Chapter 4. However, we developed only analysis and design models for the bakery system example described in Chapter 6.

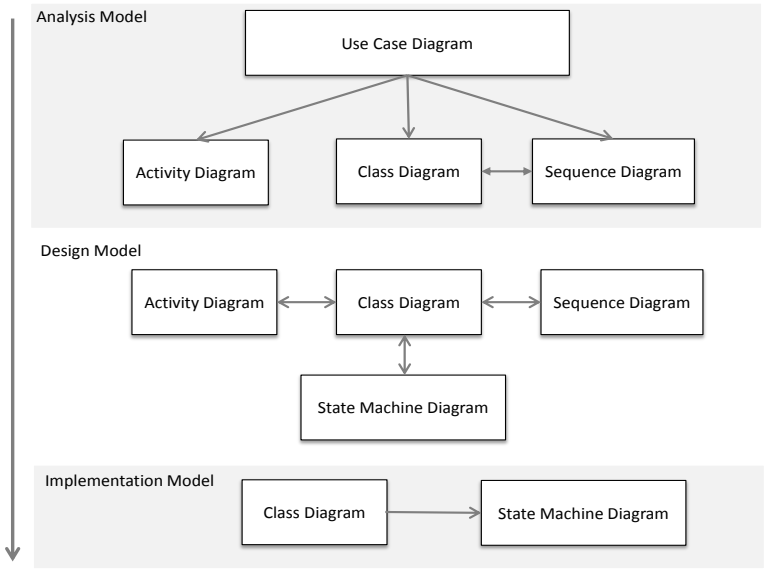


Figure 2.11: Flow of the Developed Model

2.3 Software Quality and Quality Models

Software quality refers to all attributes of a software product that show the appropriateness of the product to fulfill its requirements. For a software product, Fenton et al. [27] distinguish between attributes of *processes*, *resources*, and *products*. For each class, *internal* and *external attributes* can be distinguished. External attributes refer to how a process, a resource, or a product relates to its environment. Internal attributes, on the other hand, are properties of a process, a resource, or a product on its own, i.e., separate from any interactions with its environment. Hence, the assessment of external attributes of a product, the so-called *external quality*, requires the execution of the product, whereas usually static analysis is used for the assessment of its internal attributes, the so-called *internal quality*. Since this thesis addresses the quality characteristics for UML models, which are products that do not need to be executable, only internal quality is considered in the following.

Quality models are used to assess software quality, the prominent examples for *FCM-model* are the *McCall-model* [52] and the *ISO/IEC 9126-model* [39]. The highest level of the *McCall-model* are the three uses: *operation*, *transition* and *maintenance*. The operation use refers to quality characteristics that concern the product when it is being executed, i.e., its external quality. The transition use combines quality characteristics that concern the product when it is moved to another environment, and the maintenance use focuses on quality characteristics that concern the product when it is changed. As indicated by the abbreviation Factor-Criteria-Metrics (FCM), on the second, third and fourth level, the McCall model defines *factors*, *criteria* and *metrics*. A factor defines a high-level quality criterion such as efficiency. On the next lower level, criteria for judging factors are defined. For example, criteria for the factor efficiency are storage and execution efficiency. Metrics are then used to assess criteria, e.g., storage efficiency may be assessed by calculating the ratio between allocated and used storage.

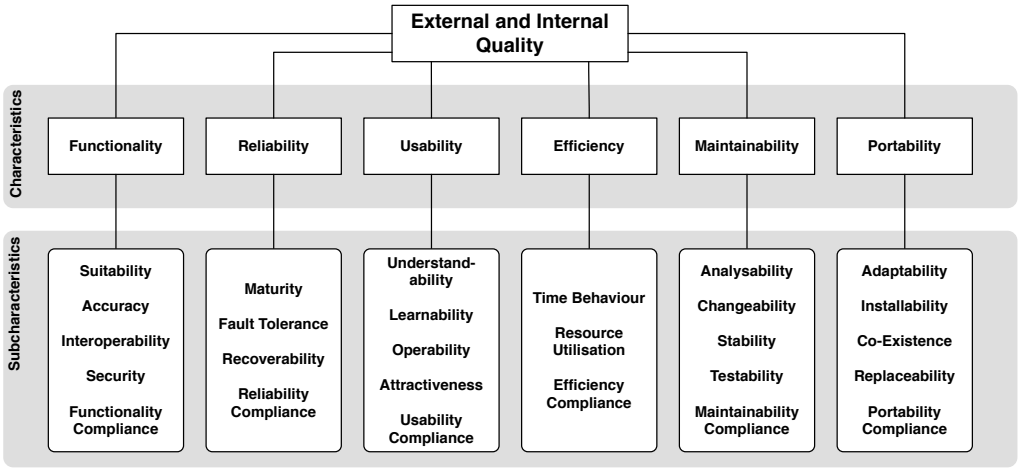


Figure 2.12: ISO/IEC 9126 Quality Model for Internal and External Quality

The *ISO/IEC 9126-model* defines no uses, but distinguishes between internal quality, external quality and quality-in-use. The quality *ISO/IEC 9126-model* is a generic quality model that covers internal and external quality in one abstract model (Figure 2.12). The model for *quality-in-use* is similar to the operation use of the McCall model. However, quality-in-use and external quality are out of the scope of this thesis, and therefore not discussed any further.

2.4 Software Metrics

Fenton et al. structured internal product metrics, i.e., metrics that measure internal quality, into *size* and *structural* metrics [27]. Size metrics measure properties of the number of usage of programming or specification language constructs, e.g., the *number of source statements*. Structural metrics analyze the structure of a program or specification. Popular examples of structural metrics are complexity metrics based on control flow or coupling metrics.

To make sure that reasonable metrics for quality assessment are chosen, Basili et al. suggest the GQM approach [9]: First, the goals which shall be achieved (e.g., improve maintainability) must be defined. Then, for each goal, a set of meaningful questions that characterize a goal is derived. The answers to these questions determine whether a goal has been met or not. Finally, one or more metrics are defined to gather quantitative data which can provide answers to each question.*

2.5 Smells

The metaphor of "*bad smells in code*" has been coined by Beck and Fowler [31]. They define smells as "*certain structures in the code that suggest (sometimes they scream for) the possibility of refactoring*". According to this definition, defects with respect to program logic, syntax, or static semantics are not smells since these defects cannot be removed by a refactoring. A refactoring only improves internal structure, but does not change observable behavior.

Beck and Fowler present smells for Java source code. They describe their smells using unstructured English text. A well-known smell is *Duplicated Code*. Code duplication affects in particular the *changeability* quality sub characteristic in the *ISO/IEC 9126-model*: if code that is duplicated needs to be modified, it usually needs to be changed in all duplicated locations. Smells provide only hints: whether the occurrence of an instance of a certain smell in a source code is considered as a sign of low quality may depend on preferences and the context of a project. For the same reason, a list containing code structures that are considered smells is never complete and may also vary from project to project and from domain to domain.

The notions of metrics and smells are not disjoint: each smell can be turned into a metric by counting the occurrences of a smell, and often, a metric can be used

*The GQM approach can also be used to define individual FCM quality models as goals are similar to factors and questions similar to criteria.

to locate a smell. The latter is the case, for example, when a long function is expressed by a metric that counts the lines of code of this function and a threshold is violated. However, the above smell of duplicated code and other pathological structures in code require a pattern-based detection approach and cannot be identified by using metrics alone.

2.6 Refactoring

Refactoring is defined as "*a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior*" [31]. This means that refactoring is a remedy against software aging [77]. While refactoring can be regarded as cleaning up source code, it is more systematical and thus less error prone than arbitrary code clean-up, because each refactoring provides a checklist of small and simple transformation steps, which are often automated by tools.

The essence of most refactorings is independent from a specific programming language. However, a number of refactorings make use of particular constructs of a programming language, or of a programming paradigm in general, and are thus only applicable to source code written in that language.

We discuss here basic concept of the rename and pull up refactorings, which we have implemented to refactor UML model. They have been adopted from Fowler refactoring catalog [30] to UML.

2.6.1 Rename Refactoring

The name of the model element does not reveal its purpose, the name of the element should be refactored. In the following paragraph, the rename refactoring motivation and mechanics for renaming a method is described. The rename refactoring can also be applied in a similar way to other elements.

Motivation: Names are an important part of the code style. The model element name should be described in a way that communicates its intention.

Mechanics: The following mechanics can be followed to use rename refactoring for a method.

- check whether the name of the method is poor or does not provide the useful meaning,
- declare a new method with the new name, copy the old body of the code over to the new name and make any changes to the name,

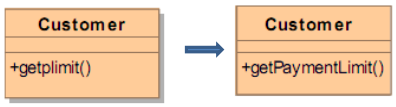


Figure 2.13: Rename Refactoring

- find all references to the old method name and change them to refer to the new one,
- remove the old method.

Figure 2.13 illustrates the renaming of method *getplimnit* to *getPaymentLimit*.

2.6.2 Pull Up Method Refactoring

Pull Up Method refactoring is used, when methods are identical in the subclasses. In the following paragraph motivation and mechanics of the pull up refactoring described.

Motivation: Any duplicate behavior should be eliminated. The duplication of methods may make problems, when a model is used for the code generation, for example, alteration of one method may not be made for the other method. The method with same body implying that there is a copy and paste of the method. This can be resolved by using pull up method refactoring.

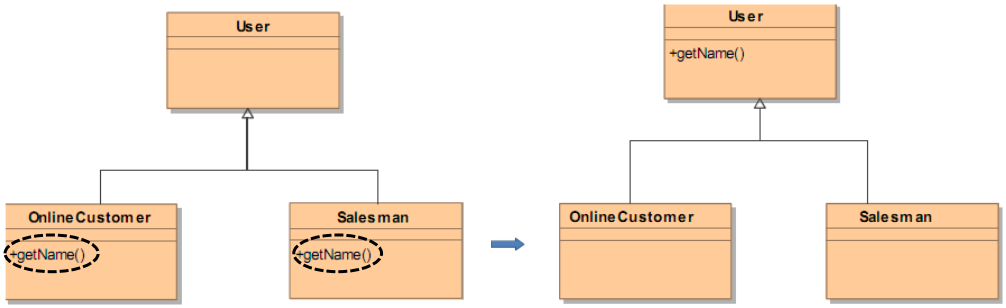


Figure 2.14: Pull Up Method Refactoring

Mechanics: The following mechanics can be followed to use pull up method refactoring.

- inspect the methods in the subclasses to ensure that they are identical,
- create a method in the superclass, copy the body of one of the method from the subclass to the super class,
- delete the subclass method.

An example is given in Figure 2.14, where *getName()* method is contained in the subclass *Online Customer* and *Salesman*. This should be refactored and the *getName()* method should be pulled up to the super class *User*.

Chapter 3

Related Work

This chapter discusses the state of the art aspects relating to the quality for UML models, in particular reference to our approach: quality models for UML models, metrics, and smells for UML models. Quality improvement work based on the refactoring is discussed, and in addition to that, the existing tool support for the quality assessment and improvement for UML is presented. In the last, the comprehensive discussion on the related work is discussed. This chapter is based on the joint work published in the proceedings of 14th System Design Languages Forum (SDL Forum) conference 2009 [41].

3.1 Quality Models for UML

A surprisingly small number of researchers have addressed the problem of quality assessment for UML models. The most comprehensive work in this area has been done by Lange and Chaudron [48, 50]. In [50], they discuss the difference between source code and UML models and highlight the particularities of UML models. As a consequence, a special quality model for UML has been developed (in the following called Lange-Chaudron-model). An overall view of the model is given in Figure 3.1.

Like the model developed by McCall, the Lange-Chaudron-model is a hierarchical model with four levels. On the highest level, the Lange-Chaudron-model defines the two uses *Maintenance* and *Development*. The maintenance use is taken from the McCall model. The other two uses from McCall, i.e., operation and *Transition*, are not relevant for the quality of UML models. The operation use is related to external quality attributes and the transition use is not related to the development phases in which UML is used, i.e., modeling and design phases.

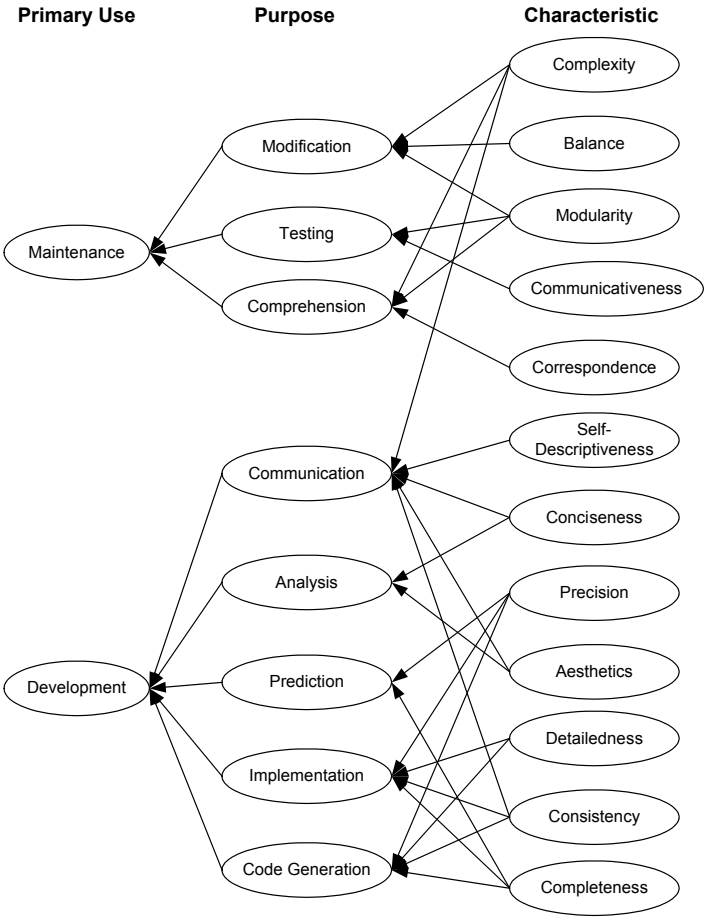


Figure 3.1: Lange-Chaudron Quality Model [48]

The second level of the Lange-Chaudron-model defines the purposes of modeling. For example, the purpose *Testing* indicates that the model is used for test generation and the purpose *Code Generation* denotes a usage for automatic code generation. The third level of the Lange-Chaudron-model identifies the characteristics of the purposes. The meaning of most characteristics in Figure 3.1 is straightforward. For example, the characteristic *complexity* measures the effort required to understand a model or a system.

Two special characteristics of the Lange-Chaudron-model are *Aesthetics* and *Balance*. The quality of the graphical diagrams is addressed by the aesthetics characteristic only. Aesthetics is defined by the extent that the graphical layout of a model or a system enables the ease of understanding of the described system. Lange and Chaudron define balance as the extent that all parts of a system are described at an equal degree. All characteristics are included in the balance

characteristic with the same weight. This has been criticized by Mohagheghi and Aagedal [57], because the assessment of the balance characteristic requires the evaluation of all metrics and rules defined in the fourth level, i.e., it is not a good abstraction. In [57], it is proposed to shift balance to the purpose level and to assess balance by using the characteristics completeness, conciseness, modularity, and self-descriptiveness.

The fourth level of the Lange-Chaudron-model (not shown in Figure 3.1) defines metrics and rules for the assessment of the characteristics. We discuss this part of Lange-Chaudron-model in the Sections 3.2 and 3.3. Lange and Chaudron underpinned their work with industrial case studies. They showed the applicability of their approach by interviewing representatives of project teams, analyzing UML models, and giving feedback to project teams.

A quality model for design documentation in model-centric domains has been developed by Pareto and Boquist [76]. The background of this work is experience with the RUP as model-centric software development process. Even though UML is an essential part of RUP, all kinds of artifacts on the abstraction levels between requirements specification and code are considered relevant. For the development of the quality model, Pareto and Boquist interviewed and discussed with designers, process engineers, line managers and architects. From these interviews and discussions, 22 quality attributes were identified and structured into six groups. Each group identified one quality characteristic. As the quality model is related to RUP also quality aspects for management are covered. However, they stop with the identification of quality attributes and quality characteristics. No means for the assessment of quality attributes and characteristics are provided.

Marín et al. [14] propose a quality model for conceptual models in terms of Model Driven Development (MDD). Their quality model is comprised of a metamodel and a set of rules. The metamodel provides the specifications for the conceptual model of the MDD environment. The set of rules are described using OCL constraints.

The 6C quality goals (Correctness, Completeness, Consistency, Comprehensibility by humans, Confinement, and Changeability) have been identified by Mohagheghi et al. [58] for the modeling domain. They provide a comprehensive literature review on model quality. They compared their proposed 6C's quality goals with the quality characteristics that have already been discussed in the literature review and categorized into three main quality goals, i.e., syntactic, semantic and pragmatic quality. However, they are still not sure that their 6C's quality goals can cover every aspect of the quality goals.

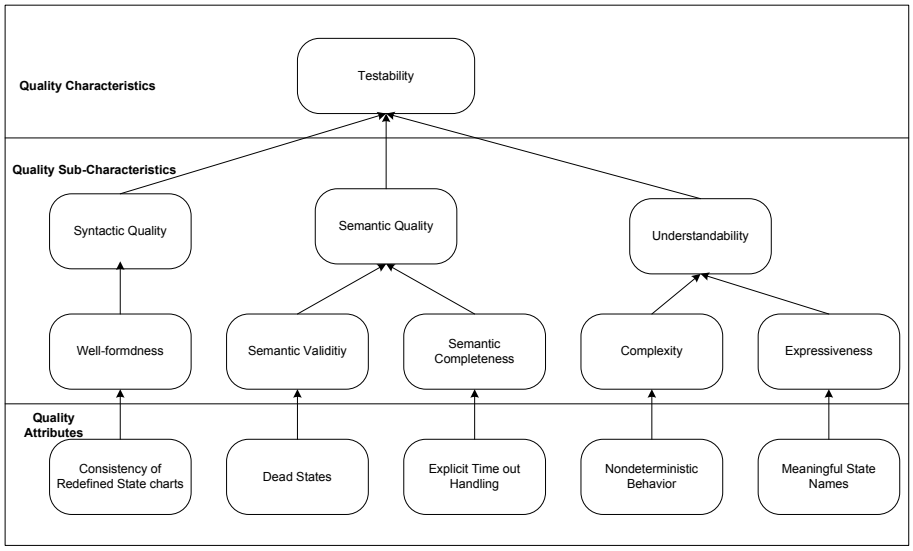


Figure 3.2: Quality Model for Testability of Models [90]

Voigt et al. [90] focused on the models used for testability. They introduced a quality plan for the assessment of quality models and quality characteristics for testability. They used state charts to show the instantiation of the model. For useful evaluation of the model, they have proposed a checklist. The quality model for the testability of models is shown in Figure 3.2. It is divided into three levels: quality characteristics, quality sub-characteristics and quality attributes. On top level there is a testability quality characteristic. They consider syntactic, semantic and understandability as their sub-characteristics, additionally, each of the sub-characteristics are further categorized. On the bottom level, they provide quality attributes for exemplary model introduced for the instantiation of the quality model. They did not cover important quality attributes for the testability such as controllability and observability [13].

3.2 Metrics for UML

A UML model is based on a specific structure: the UML metamodel. However, numerous proposals are based on metrics for UML but they often respect only diagrams, i.e., the graphical representation of UML with its partial views. In the following sections, we will present noteworthy literature on UML metrics. We differentiate between metrics that are based on the actual UML model and metrics that are solely based on the graphical notation, i.e., graphical metrics.

3.2.1 Model Metrics

Lange and Chaudron [47] uses metrics and rules (metrics with a binary result) and relates them to quality characteristics of his quality model (see Section 3.1) to assess the quality of a UML model. He reuses the most widely known metrics such as the metric suite from Chidamber and Kemerer [18] and describes them informally. He stresses that his list is by no means complete.

Kim and Boldyreff [44] propose 27 metrics for UML that are supposed to predict characteristics at earlier stages in the software lifecycle. The metrics are defined informally and no relationship between the UML model quality and the metrics is established.

Baroni et al. [7] propose to use OCL to describe UML metrics in a formal way in order to avoid ambiguities due to descriptions in natural language. By using several samples of different complexity, they demonstrate that OCL is a well suited formalism for defining UML metrics and that it is easier to understand than formulas using custom built mathematical frameworks. McQuillan and Power [53] extended this approach and use OCL to calculate coupling and cohesion metrics, as well as the metrics from the Chidamber and Kemerer metric suite [18]. They argue, however, that a metrics specific metamodel is a more generic solution than defining metrics directly over the UML metamodel. Furthermore, they demonstrate how to automatically generate test data and metamodel instances.

Another interesting way to formalize metrics is proposed by El-Wakil et al. [24]. They propose to define metrics using *XQuery* over the XMI representation of the UML model under analysis. They argue that using XQuery to express metrics eases tool building. Also, they claim that metric libraries specified in *XQuery* are easy to extend and provide a proof-of-concept implementation.

Kollmann and Gogolla [45] present an approach, which aims at using object-oriented metrics on class diagrams by isolating the coherent sub-modules. As large diagrams are difficult to understand, their approach looks feasible for isolating the coherent sub-module from the existing class diagram. The authors have realized that it is always hard to see the complete structure of the model at the same time. They have implemented their approach in their reverse engineering tool.

Ma et al. [51] compares different versions of UML meta-model. The measurement is based on object oriented metrics. Brenbach and Borotto [11] provides a metrics catalog for model driven requirements development based on good practices.

The Modelware project [55, 56] delivers three documents on how to measure the quality of models providing several metrics based on Model Driven Engineering (MDE) processes. The research in these documents is dedicated to the MDE artifacts, for example, the metamodel, processes and the UML models. The main focus on metrics is related to the metamodel and the processes, while little emphasis is put on the UML models.

3.2.2 Graphical Metrics

Graphical metrics for UML are not covered very well in the literature despite the fact that the quantification of visual elements can be an important part to assess the quality of a graphical layout. However, it seems that lay outing itself draws more attention in research than the assessment of a layout by numbers.

Kiewkanya and Muenchaisri [43] performed an experiment in which they evaluated whether metrics quantifying aesthetic aspects of class and sequence diagrams influence the maintainability of UML models. For the measurements, they selected aesthetic indicators that have been proposed by Purchase [79], Eichelberger [23], and others. Such aesthetic indicators are, for example, the maximum number of bends on the edges, the standard deviation of edge lengths, or the total numbers of edges fixed to an orthogonal grid divided by the total number of edges. Their conclusion is that aesthetic metrics can indeed be indicators for the maintainability of class and sequence diagrams.

Gronback [35] provides a general catalog of UML metrics to detect deviations from best practices. Some of them are derived from style guidelines provided by Ambler [2]. He suggests generic diagram metrics such as ‘‘number of colors on diagram’’ or diagram-specific metrics such as ‘‘depth of inheritance hierarchy’’ (for class diagrams) and even provides minimum and maximum thresholds for his metrics. The metrics presented by Gronback, however, mix graphical properties with properties that are part of the UML model.

3.3 Smells for UML

As discussed earlier, UML models do not have a standardized textual notation like typical general purpose programming languages. However, bad smell analysis in source code is rarely executed directly on the textual notation. An abstract grammatical representation of the notation, the Abstract Syntax Tree (AST), can in fact be regarded as a model for the textual notation of the programming language that is subject of the analysis. Analyzing UML models is therefore not that much different than analyzing an AST.

In the following section, we present related work that deals with bad smells in UML models. We differentiate between model smells and graphical smells. With model smells, we regard design flaws or possible defects that we find by analyzing the UML model (independently from any diagrams) such as possible inconsistencies, ambiguities, or constructs that complicate maintenance. Graphical smells, on the other hand, are related to the graphical notation of UML. They primarily concern the understandability aspect of the diagram. For example, diagrams with overlapping or crossing elements are harder to understand than diagrams with elements that are properly laid out with aesthetic aspects in mind.

3.3.1 Model Smells

Lange and Chaudron [48] with his goal to improve the overall quality of UML models discusses that undetected defects can cause large problems in later development stages and identifies generic UML defects such as the number of messages in a sequence diagram that do not correspond to a method in a defined class diagram. The presented smells were identified by discussions with industrial partners and by performing case studies. He assumes that a set of UML diagrams defines a system as a whole and that those diagrams have consistency relationships between each other. The defects partially overlap with the well-formedness rules and are related in their scope, but are described informally, without a relationship to the abstract syntax of UML.

Astels [6] presents UML smell detection in the context of UML refactoring. With smell detection, he locates where to refactor and which refactoring is suggested. He argues that the visual presentation of UML makes smell structures more evident and presents exemplary what classical bad smells from Fowler [31] (e.g., lazy class or middle man) look like in the graphical notation. His own statement is that his list is by no means complete. His work is described informally in the visual notation of UML.

3.3.2 Graphical Smells

Graphical smells concern the graphical notation of UML models excluding problems that are of logical nature or that may introduce issues in efficiency or maintenance. Therefore, the main aspect of graphical smells is how model elements are laid out and what elements are represented by the diagrams.

Ambler [2] provides more than 300 guidelines for all UML diagram types that primarily concern the graphical notation. The violations of these guidelines can be considered as graphical smells.

Purchase et al. [79] have studied graphical layout aesthetics in class and collaboration diagrams. By performing a case study where they questioned people in order to investigate their subjective preferences, they conclude that there are certain common aesthetic properties that seem to be unfavorable. Among these properties are, for example, arc crossings, or orthogonality (for class diagrams). From their results, they derive that the aesthetics of graph layouts is dependent on the domain, i.e., properties that are important for one diagram type may not be important for another one.

3.4 Refactorings for UML

UML refactoring is an emerging research topic that can already be considered as important as classical source-code refactoring. We again differentiate between model refactorings, i.e., semantically preserving model changes and graphical refactorings that improve the aesthetics of UML diagrams.

3.4.1 Model Refactorings

Astels [6] presents UML smells in class and sequence diagrams and describes a number of Fowler refactorings that are applicable to UML. His refactoring descriptions are based on UML diagrams and are informal. His examples are intended to motivate that UML refactoring is applicable in the context of agile development processes.

France and Bieman [32] in order to avoid uncontrolled change and increased evolution costs of a system due to deteriorating structure and system quality by introducing a goal-directed, cyclic process for object-oriented software when object-oriented models, such as UML models, are transformed and evaluated in each cycle. For the model transformation, they explicitly mention model refactoring to enhance quality attributes of the model that should be realized using patterns involving roles, i.e., each participant in the pattern plays a certain role with specific properties within the pattern description. A formal method for pattern-based transformation with role models does not exist yet.

Sunyé et al. [87] propose refactorings for class diagrams and state charts to make software easier to extend and maintain. Using pre and post conditions expressed in OCL, they ensure that transformation preserve behavioral properties.

Porres [78] presents how to describe and execute UML refactorings using a rule-based transformation formalism and he argues that an update-based mapping mechanism that modifies a model in place is more efficient for describing

refactorings than *mapping transformations* that transform into a different target model. For the realization and description of refactoring transformations, he uses his own language called *SMW* that operates on the UML metamodel --- when the paper was written, there were no widely adopted transformation languages available.

Dobrzański [19] provides a comprehensive survey on UML model refactorings in his master's thesis that deals with the refactoring of executable UML models [54]. He introduces an initial refactoring catalog for executable UML models. The refactorings are formalized with pre and post conditions in OCL. According to him, the main difference in refactoring executable models is that the update of the behavioral aspects of the models has to be taken into account.

More recent work on UML model refactoring and transformation is often based on the EMF representation of UML models. Biermann et al. [12] present work on an EMF model transformation framework that is based on graph transformations. They show how the rich theory of algebraic graph transformation can be applied to EMF model transformations. Using their method, the validation of the model transformations with respect to functional behavior and correctness is possible. They demonstrate their approach by using selected state chart refactorings.

Folli and Mens [28] proposed the usage of graph transformations for model refactorings and present, as a proof-of-concept, how they have implemented more complex UML model refactorings using the *AGG* [88] graph transformation tool.

3.4.2 Graphical Refactorings

Graphical refactorings are applied when the graphical notation of a UML model, i.e., corresponding diagrams containing partial views of the UML model are hard to read and understand. There are a huge variety of generic graph layout algorithms, and graph drawing itself is a very active research topic. Summaries can be found in a variety of textbooks, for example, Graph Drawing by Battista et al. [10].

Work on layouts of UML diagrams is rare. Ambler [2] provides informal guidelines that lack a systematic transformation mechanism to improve diagrams. However, it is arguable whether graphical refactorings should only change parts of a model using the refactoring mechanism or whether UML diagram specific transformations for complete optimal layouts are more desirable. Eichelberger and Gudenberg [23] discuss existing automatic layout methods for class diagrams and present their approach to laying out class diagrams that respect aesthetic rules, such as those described by [79].

Castello et al. [17] propose an automatic layout algorithm that improves the readability of state chart diagrams. It reduces the number of edge crossings and edge bends.

3.5 Tool Support

It is encouraged to use tools for measuring metrics, detecting smells, and applying refactorings to UML models. Manual application of refactorings, for example, is very error-prone and there is a risk that the changes are not semantically preserving due to human mistakes. Popular tools that support the automatic calculation of metrics and detection of bad smells in UML models are SDMetrics [82], Together [16], IBM Rational Systems Developer [36], and ArgoUML [4]. These tools partially use different terminologies for the term ''bad smell''. SDMetrics, for example, calls them *design rules*, Together calls them *audits*, or ArgoUML names them *design critics*.

The toolset from Chaudron et al. [49] calculates metrics on UML models, it detects rules in sequence diagrams, it checks model consistency, and visualizes metrics in a metric view tool [26]. Except for the commercial tool Poseidon for UML, which provides a refactoring browser supporting the refactorings from Boger at al. [15], none of the major commercial UML tools support refactoring beyond renaming and moving model elements. Tools that support more sophisticated UML refactorings are academic prototypes. An overview over existing academic UML refactoring tools is given by Dobrzański [19]. Van Gorp et al. [34] have implemented refactorings as plug-in for the Fujaba UML tool. Recently, several academic UML refactoring tools are evolving that build on EMF, for example, GaliciaUML [83] and MoDisco [38]. The latter provides extensible framework for the development of MDD tool support for existing systems. It includes quality assurance by identifying anti-patterns and computation of metrics.

Other most promising Eclipse based model transformation projects are ATL [37, 42], QVT [68] and Xpand [40]. EMF Refactor [25] is another eclipse based project for model refactoring, which is still in the proposal phase and some of the initial refactorings have already been defined for class and state machines.

3.6 Discussion

Our approach for a continuous quality assessment and improvement process for UML Models comprises a quality model defining quality characteristics, rules for quality assessment and the detection of issues and refactoring for quality improvement.

Instead of re-using the Lange-Chaudron quality model (Section 3.1) or defining another Factor-Criteria-Metrics (FCM) model (Section 2.3), we developed a new quality model that is based on an inclusion relationship instead of having a hierarchical structure. This has been done because (1) UML models are created on different level of abstractions, and at a different level of completeness. Due to the inclusion relationship of our model, the models are refined in the next stages and their level of details and the level of completeness increases (2) we did not classify the quality attributes by factors like FCM based models. The main reason was that the quality attributes which are applicable to UML models do not necessarily need further classification. If we do so, this may lead to confusion, especially in the additional model completeness dimensions.

The FCM quality models use metrics and rules for quality assessment of the different quality characteristics. Metrics are sometimes difficult to interpret and do not give an immediate hint of how to improve quality. Therefore, we concentrated on rules and assigned a set to each quality attribute. A rule can be seen as a binary metric and by counting rule violations; our approach becomes comparable to the FCM approach. A rule violation can be seen as a smell and thus, smell removal by means of refactoring is the natural way for quality improvement.

For the implementation of rules, we used OCL, i.e., adapted the approach of Baroni et al. [7], they used OCL to describe UML metrics in a formal way. Prototypically, we also implemented the rename and the pullup refactoring from the Fowler catalog, for this we used M2M transformation language [21].

Chapter 4

A Quality Model for UML and its Instantiation

This chapter explains a novel quality model for UML, different type of model completeness types used in modeled development process, their inclusion relationship to the quality model for UML, and its quality attributes. The instantiation of the quality model is presented based on the selection of a UML subset and clarification of rules and guidelines using the GQM approach.

4.1 Description of the Quality Model for UML Models

To assess the quality of UML artifacts, a model that classifies quality characteristics of such artifacts and that characterizes each of these characteristics for specific states of the model is needed. In current research, only a small number of researchers deal with quality assurance of UML while more generic software quality assurance research, especially dealing with internal source code based quality, is more popular.

Our observation is that there are three types of UML models that are produced throughout a software life cycle: *incomplete models*, *complete models*, and *executable models*. The model type to some degree corresponds to the progress made to the model development within the software development process. Incomplete models are usually found in earlier phases of a software development process. Complete models are the result of the design phase. Executable models are *complete* models that are refined in the implementation phase.

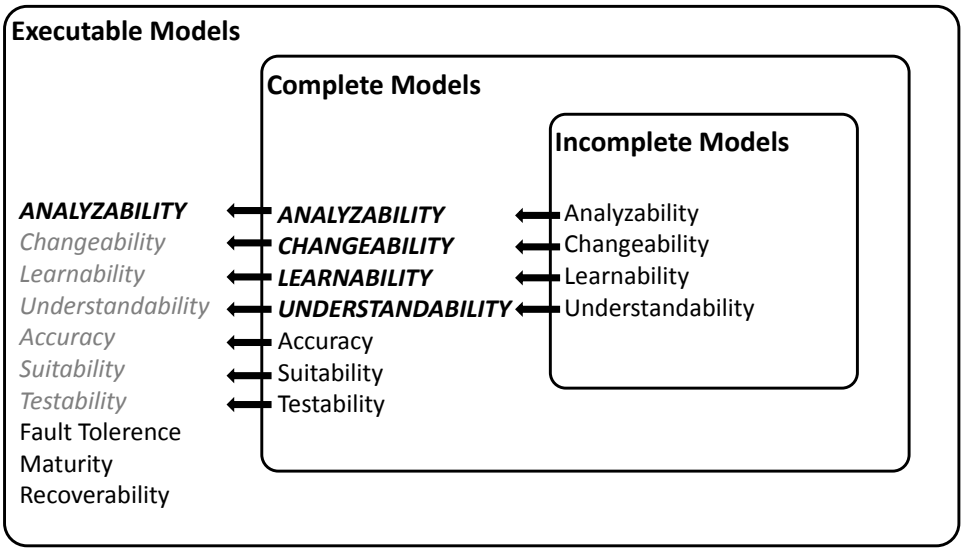


Figure 4.1: Proposed Quality Model for UML Models

The differentiation between these three kinds of model types is foundation of our newly proposed quality model for UML. It respects that UML models are created for different purposes, on different levels of abstraction, and at different levels of completeness.

Figure 4.1[†] illustrates our quality model for UML. This model relates to the *internal quality* of a UML model. *Internal quality* describes the totality of characteristics of a model from an internal view, i.e., characteristics that concern the model, the model documentation, or the model layout (in its visualized form). The measurement takes place with internal metrics. *External quality*, on the other hand, is the totality of characteristics of a model from an external view, i.e., characteristics that are measured when the model is executed or simulated. The measurement takes place with external metrics. Even though external quality plays a role in the context of *executable* models, this work and model is focusing on the assessment of quality attributes prior to any execution of the model. The model types are depicted as sets containing quality attributes that relate to each other with an inclusion relationship, i.e., the quality attributes that characterize an *incomplete model* are also quality attributes of *complete* models. Furthermore, all attributes of the *complete model* are as well attributes of *executable* models.

The inclusion relationship exists due to the fact that the degree of completeness rises with each inclusion relationship, i.e., *complete* models are refinements of

[†]Italic uppercase bold letters denote that the quality attribute is redefined. Italic without uppercase letters and are not bold denotes that the quality attribute is used with out any change in the meaning and the quality attributes without italic denotes that the quality attribute is new.

incomplete models and *executable* models are refinements of *complete* models. In the following, we discuss the model types in more detail.

The quality attributes in each model type set are derived from the *ISO/IEC 9126* standard [39] for internal and external quality of software artifacts. One specific characteristic of this model *ISO/IEC 9126-model* is that the number of its quality attributes is manageable in quantity and intuitive regarding their meaning. This means the problem of quality terms with overlapping meaning is not that apparent in this model. Therefore, we believe that deriving quality terms from this model is a good choice. However, our proposed quality model is not a *FCM-model*. While quality attributes (criteria) in *FCM-model* are classified by factors (e.g., the main characteristics in the *ISO/IEC 9126-model*), we do not classify the quality attributes by factors. The reason is that after analyzing the quality attributes that are applicable to UML models, the number of overall quality attributes has been narrowed down to such a small amount that a further classification is not necessary and on the contrary would have been rather confusing, since we have the additional model completeness dimension.

The inclusion relationship between the model type sets manifests itself in two different ways. Either, a quality attribute is redefined or it is implicitly included. Note that the italic uppercase letters and bold denotes that the quality attribute is redefined. Italic without uppercase letters and are not bold denotes that the quality attribute is used with out any change in the meaning and the quality attributes without italic denotes that the quality attribute is new. Quality attributes such as the ***ANALYZABILITY***, ***CHANGEABILITY***, ***LEARNABILITY*** and ***UNDERSTANDABILITY*** are redefined in *complete model* and **Accuracy, Stability, Suitability and Testability** are new quality attributes for the *complete model*.

In the *executable model* quality attribute ***ANALYZABILITY*** is redefined and other quality attributes of the *complete model* ***Changeability, Learnability, Understandability, Accuracy, Stability, Suitability*** and ***Testability*** are used without change in their meaning. **Fault Tolerance, Maturity** and **Recoverability** are newly added quality attributes. A repeated occurrence of a quality attribute is due to the inclusion relationship.

To instantiate the model, each quality attribute must be represented by a number of quantifiable numbers, i.e., metrics whose values can be used for the interpretation of these quality attributes.

4.2 Model Completeness Types and Quality Attributes

This section explains the model types and the quality characteristics of the quality model for UML in more detail.

4.2.1 Model Completeness Types

We consider three different completeness types of UML models for the quality assessment. These are *incomplete models*, *complete models*, and *executable models*. Completeness can be considered on various different levels. For instance, a UML model consisting only of use cases can in fact be considered as complete with regards to a requirements artifact. However, it cannot be considered complete with regards to a complete software system. For the model completeness types of our proposed quality model for UML, we consider completeness in relation to the contents of a software specification from which a complete *executable* software system can be derived. We assume that UML models are refined in each phase of a software development process. Therefore, all three model completeness types are consequently, part of a model-driven software development process. Furthermore, we assume that all model types are syntactically correct. This is a prerequisite for the analysis and, thus, also the assessment of the model.

- **Incomplete models**: Incomplete models are models that are syntactically correct, but that either miss information (i.e., they are logically incomplete) in order to be considered to be complete software specification, or they are ambiguous in what they are trying to express. For example, a UML model consisting of class diagrams can never be considered as complete as a model because class diagrams cannot express behavior and a software system must always include some kind of behavior. Incompleteness can also refer to inconsistencies within the model: there are situations when structural elements and behavior specification both exist in the model. However, when they are not properly linked together, the model is considered to be incomplete. Incomplete models are either models that are gradually refined in each phase of a software development process or they are special-purpose models that, for example, document one specific part of the software system and remain in their incomplete state.

- **Complete models**: Complete models are syntactically correct models that represent a structural and behavioral specification of a software system. The models are consistent and constrained to be as non-ambiguous as possible (for example, with the help of OCL expressions [70]). A *complete model* is the prerequisite for any derivative code generation. However, a *complete model* is not platform-specific, i.e., it makes no assumptions about the target framework, target architecture, or target implementation language in use. In

that sense, it basically corresponds to the idea of the Platform Independent Model (PIM) in the MDA approach [33]. However, with constraints, it presents a syntactically correct, consistent, non-ambiguous, and logically complete software specification.

As *complete* models are refined *incomplete* models, they inherit all quality attributes of *incomplete* models.

- **Executable models**: Executable models are *complete* models that are either directly executable with a model interpreter or translatable to executable machine code without any further additions. This implies that *executable* models are not as abstract as *complete* models. However, they contain platform-specific information as well (therefore, roughly corresponding to the PSM concept of MDA [54]). Furthermore, in comparison to *complete* models, they might be tailored to realize efficiency constraints towards the developed software as well (which is not the case for *complete* models). Executable models inherit all quality attributes of *complete* and *incomplete* models.

Each model type is assessed with the help of a number of quality attributes, which are described in the next paragraphs.

4.2.2 Quality Attributes

In the following, we describe quality attributes that we mention in our proposed quality model for UML as described in Section 4.1. We discuss each attribute along all model types. As mentioned before, we differentiate between redefined quality attributes and implicitly included attributes. Implicitly included attributes are described once for the model type in which they are defined first. The description of redefined quality attributes is refined for each redefinition.

4.2.2.1 Incomplete Models

- **Analyzability**: This is the ability to examine deficiencies or possible causes of failures in UML models. In the context of *incomplete* models, the amount of possible analysis that yield a meaningful conclusion is limited. This is caused by possible ambiguities and inconsistencies, which implies that such analysis often results in the detection of problems. However, we can still influence the analyzability quality attribute in the context of *incomplete* models. For example, we can avoid name clashes, link or reference model elements among each other to the degree that they exist, or in general we should avoid inconsistencies within the model.

- **Changeability**: Changeability is the capability of the model to allow specified modifications to be incorporated. Changeability is impeded by model complexity, hard to understand models (see *learnability*), or models that require shotgun surgery for changes (i.e., one change cascades a number of other necessary changes), and similar.

- **Learnability**: Learnability is the capability that enables the user to learn how the model works and how it is used. The user of the model must understand how the model is formed what kind of requirements are considered, and how different parts of the model (e.g., diagrams of different types) are interrelated. In addition, the learnability depends also on factors such as the consistent usage of guidelines for naming conventions.

- **Understandability**: Understandability is the capability that enables users to understand whether the model is suitable for particular tasks and for particular conditions of use.

4.2.2.2 Complete Models

- **Analyzability**: The aspects of *incomplete* models can be extended with actual logical analysis, for example, structure and behavior are expected to be complete. Therefore, the analyzability aspect is described by the degree to which a logical analysis of a UML model for deficiencies or possible failure causes is possible. Assuming that *complete* models are consistent and non-ambiguous, remaining issues regarding the analyzability of *complete* models lie, for example, in the complexity of the model semantics, for example, when a specific UML profile is in use.

- **Changeability**: For *complete* models, the changeability aspect is more profound in the sense that changes must additionally satisfy constraints and consistency rules. These need to be valid for *complete* models and they do not need to be enforced on *incomplete* models.

- **Learnability**: For *complete* models, additional constraints and consistency rules can exist that influence the difficulty of the learnability quality attribute.

- **Understandability**: As *incomplete* models do not represent a complete software specification, the degree to which we understand whether the model is a suitable for particular tasks and for particular conditions of use is influenced by the model elements that have been missing in the *incomplete model*.

- **Accuracy**: Accuracy is the capability to provide the correct or agreed results or effects with the needed degree of precision. Furthermore, a UML model is only accurate when its elements are traceable with respect to the requirement's specification.

- **Stability**: Stability is the capability of the model that avoids unexpected effects when modifications are applied to it. This refers to a side-effect free behavior specification.

- **Suitability**: Suitability is the capability that the model is appropriate for specified tasks and user objectives. This constitutes the degree to which structural and behavioral descriptions cover requirements.

- **Testability**: Testability is the capability that enables model validation after modifications. To validate such a model, the model must be syntactically correct, as non-ambiguous as possible, and consistent. Furthermore, testability deteriorates with increasing model complexity due to increased difficulty and effort to actually reveal problems.

4.2.2.3 Executable Models

- **Analyzability**: In addition to the analyzability issues mentioned in the *incomplete* and *complete* model descriptions, the analyzability attribute for *executable* models also requires that possible platform-specific additions to the model remain in a consistent state regarding the completeness properties. This incorporates also the degree to which any possible action languages in use are analyzable.

- **Fault Tolerance**: It is the ability to maintain a specified level of performance in cases of faults or misuse. The fault tolerance aspect is primarily steered through the behavioral specifications in the UML model including how behavior in action languages is specified and what action language is in use.

- **Maturity**: Maturity is the capability to avoid runtime failure as a result of faults in the model. This implies the degree to which model behavior is specified in a defensive way.

- **Recoverability**: The capability to re-establish a specified level of performance and recover itself in case of failure. In addition to behavior specification that concerns the frequency of failure, the recoverability aspect also covers to which degree the specified behavior provides recovery strategies for unexpected cases.

4.3 Towards an Instantiation of the Quality Model

In this section, we present the an exemplary instantiation of the quality model for UML. To evaluate and assess the quality of UML models, rules and guidelines need to be associated to the respective quality attributes. The violation of these rules are considered to be bad smells on models. The detected violations are improved by model to model transformations that refactor the model.

We distinguish between critical and non-critical rules. Before the transition from one model type to another model type, critical rules must be improved to ensure the model completeness. Non-critical rules are guidelines for the modeler which can be followed according to the specified quality requirements.

The instantiation of our quality model is based on three quality attributes for *incomplete* and *complete* models: analyzability, changeability and understandability. The selection of these three attributes is based on two reasons: 1) the models used for assessment are designed by students 2) the relation of these three attributes to the rules and guidelines is easy to follow by the students. The case study and their results are described in Chapter 6. Before going into further details of the instantiation of the quality model, we select a manageable set of the UML diagrams for each model type, which is described in the following section.

4.4 Selection of UML Subset Notations

UML provides a wide variety of possible diagrams to choose for modeling. This flexibility of selecting appropriate diagrams leads to problems.The OMG does not provide any method to select the appropriate notation. However different factors should be considered while selecting the subset of UML. This includes the experience of the modeler or the necessity of automatic code generation.

Our proposed model is based on three types of models: *incomplete*, *complete* and *executable*. The selection of a subset for the *incomplete* and *complete* model is presented in Table 4.1. The *incomplete model* captures the user view at the inception and elaboration cycle of unified process, while *complete model* captures the implementation view at elaboration cycle of unified process. The UML diagrams Use case diagram, class diagram, sequence diagram and activity diagrams are used for *incomplete* model and *complete* model adds the implementation specific diagrams, these are: class diagram, sequence diagram, activity diagram and state machine diagrams. The selection is based on the case study described in Chapter 6.

Model Type	Diagram Type	Purpose of a Diagram
Incomplete	Use Case Diagram	Capture the required use cases of the system from the point of view of the system user
	Activity Diagram	Used to describe a subsystem flow
	Sequence Diagram	Shows the exact order of a message flow, in case a particular use case has been invoked by an actor to perform the systems functionality
	Class Diagram	Used to design system structure
Complete	Class Diagram	Actual design level attributes, operations and other elements are added to address the completeness of the model
	Sequence Diagram	Describe how objects interact when messages are exchanged
	Activity Diagram	Used to model class operations
	Statemachine Diagram	Used to describe behavioral or protocol state machine for a class or for an interface

Table 4.1: UML Subset Selection

4.5 Classification of Rules and Guidelines

The application of our quality model is evaluated in an experimental case study in which the bakery system models (Section 2.2) are developed by students. The results are described in the Chapter 6 for *analyzability*, *changeability* and *understandability*. In this section, we apply the GQM [8] approach to select appropriate rule or guideline for these quality characteristics.

The rules and guidelines are further categorized into the critical and non-critical criteria. To distinguish between rules and guidelines, we used the keywords *must* for rules and *should* for guidelines. Therefore, rules must be followed to build quality UML models while guidelines are merely suggestions. The rules or guidelines are defined in plain text, as well as in OCL language as described in Section 2.1.3. The definition of OCL is based on the UML metamodel [72]. Therefore, OCL queries can be directly executed on UML models. These queries are used in the Xtend language with a small modifications. The list of the queries can be found in Appendix E for *incomplete* and *complete* models.

The GQM based approach in the following paragraph is described for the classification of rules or guidelines for each quality attribute. To answer the questions that cover rules or guidelines are used for multiple questions, hence the rule number described in the (Sections 4.5.1 to 4.5.6) does not follow the question number. The complete list of rules and guidelines in sequential order is described in Appendix B.1 for *incomplete* model and Appendix B.2 for *complete* model. The rules for *incomplete model* is represented by R_{in}, where i represents the

incomplete model and n denotes the rule number and the rule for *complete model* are represented by R_{cn}, where c represents the *complete model* and n denotes the rule number.

4.5.1 Analyzability for Incomplete Models

The analyzability is measured by considering consistency and traceability issues. Therefore we selected the rules which are important for our experimental cases study by asking questions according to the GQM approach and try to answer them with certain rules.

Q.1: Is the *incomplete model* consistent?

a) Rules or guidelines measuring or determining the consistency between use case diagram and sequence diagram.

R_{i4}: Each use case must be refined in a sequence diagram.

```
1 context uml::UseCase
2 inv: ownedBehavior->select(b|b.oclIsKindOf(Interaction) and
3 b.oclIsTypeOf(Interaction)->size() > 0
```

R_{i32}: Each sequence diagram should have at least one actor on a lifeline.

```
1 context uml::Interaction
2 inv: self.lifeline.represents.type->exists(oclIsTypeOf(Actor))->size() >0
```

b) Rules measuring or determining consistency between use case and activity diagram.

R_{i16}: Each subsystem should be refined by one activity diagram.

```
1 context uml::Package
2 inv: Activity.allInstances().name->includes(self.name)
```

c) Rules measuring or determining consistency between use case and class diagrams.

R_{i18}: Each subsystem of a use case diagram should be represented as a package in the class diagram.

```
1 context uml::Class
2 inv: Package.allInstances().name->includes(self.owner.name)
```

Q.2: Is the *incomplete model* traceable?

a) Rules measuring or determining traceability between activity and use case diagram.

R_{i17}: Each activity in an activity diagram should refer to a use case in use case diagram.

```
1 context uml::CallBehaviorAction
2 inv: UseCase.allInstances().name->includes(self.name)
```

b) Rules measuring or determining traceability between sequence and class diagrams.

R_{i33}: Each object or lifeline in a sequence diagram must have corresponding class in a class diagram.

```
1 context uml::Lifeline
2 inv: self.represents.type->exists(oclIsTypeOf(Class)) or
3  self.represents.type->exists(oclIsTypeOf(Actor)) or
4 self.represents.type->exists(oclIsTypeOf(Interface))
```

R_{i34}: Every call message received by a lifeline must have a corresponding operation in the class.

```
1 context uml::Message
2 inv: ((not receiveEvent.oclAsType(MessageOccurrenceSpecification).
3 event.oclIsUndefined())and(receiveEvent.oclAsType(
4 MessageOccurrenceSpecification).event.oclIsTypeOf(CallEvent))) implies not
5 (receiveEvent.oclAsType(MessageOccurrenceSpecification).event.oclAsType(CallEvent).
6 operation.oclIsUndefined())
```

R_{i35}: If there is a message call between two lifelines then there must also be an association between two classes.

```
1 context uml::Lifeline
2 inv: (MessageOccurrenceSpecification.allInstances().
3 covered->includes(self)) and (Association.allInstances().getEndTypes()->
4 select(oclIsTypeOf(Class))->asSet()->includes(self.represents.type))
```

4.5.2 Changeability for Incomplete Models

The changeability is measured by considering coupling and cohesion issues. Therefore, the rules are based on the following questions.

Q.1: How are the classes coupled in a class diagram?

a) Rules or guidelines measuring or determining coupling between classes.

R_{i20}: The depth of inheritance tree should not exceed 2.

```
1 context uml::Class
2 inv: self.superClass.superClass.superClass->size()=0
```

R_{i21}: Multiple inheritance must not exist.

```
1 context uml::Class
2 inv:self.general->select(oclAsType(Class))->size()<2
```

R_{i28}: Each class should have 1-5 associations.

```
1 context uml::Class
2 inv: self.attribute.association->size()>0 ||
3 self.attribute.association->size()< 6
```

Q.2: How are the use cases coupled within model?

a) Rules or guidelines measuring or determining coupling between use cases.

R_{i3}: The generalization between use cases must not be present in a use case diagram.

```
context uml::UseCase
inv: self.parents()->size()=0
```

R_{i10}: The depth of generalization of an actor should not exceed one.

```
context uml::Actor
inv: self.parents().parents()->size()=0
```

R_{i14}: The depth of include chain of a use case should not exceed one.

```
context uml::Include
inv: self.source->includes(self)->size()<2
```

R_{i15}: The depth of extend chain of a use case should not exceed one.

```
context uml::Extend
inv: self.source->includes(self)->size()<2
```

4.5.3 Understandability for Incomplete Models

The understandability is measured by considering various quality issues, for example, structural and behavioral complexity, overall complexity of the model and the UML coding conventions.

Q.1: What is the structural complexity of the model?

a) Rules that measure structural complexity of the *incomplete model*.

R_{i27}: Each association must specify multiplicity values at both ends.

```
context uml::Association
inv: self.memberEnd ->forAll ( n | (not n.lowerValue.oclIsUndefined()) or
(not n.upperValue.oclIsUndefined()))
```

R_{i30}: Classes should not be linked with composition or aggregation association type.

```
context uml::Property
inv: let opposite:Property = self.opposite.association.memberEnd->
any(e|e<>self) in (opposite.aggregation<>AggregationKind::shared)
and (not(opposite.isComposite))
```

R_{i31}: The links to classes belonging to another package must be unidirectional.

```
context uml::Association
inv: self.memberEnd.isNavigable()->includes(false) and
self.getEndTypes()->select(oclIsTypeOf(Class))->
exists(e1,e2|e1.owner <> e2.owner)
```

Q.2: What is the behavioral complexity of the model?

a) Rules that measure behavioral complexity of the *incomplete model*.

R_{i5}: A use case should not linked to more than 3 actors.

```
context uml::
inv: (getRelationships()->reject(oclIsTypeOf(Extend))
and getRelationships()->reject(oclIsTypeOf(Include)))
implies self.getRelationships().relatedElement->
select(oclIsTypeOf(Actor))->size()<=3
```

R_{i28}: Each class should have 1-5 associations.

```
context uml::Class
inv: self.attribute.association->size()>0 ||
self.attribute.association->
size()< 6
```

Q.3: What is the overall complexity of the *incomplete model*?

a) Rules that measure the overall complexity of the *incomplete model*.

R_{i1}: Each use case must be inside one subsystem.

```
context uml::UseCase
inv: self.owner->exists(oclIsTypeOf(Package))
```

R_{i2}: Each use case must be associated with an actor.

```
context uml::UseCase
inv: self.getRelationship()->reject(oclIsTypeOf(Extend) || oclIsTypeOf(Include)) and
self.getRelationships().relatedElement->select(oclIsTypeOf((Actor))->size() > 0
```

R_{i7}: Each subsystem should contain a minimum of 3 and a maximum of 5 use cases.

```
context uml::Package
inv: (self.allOwnedElements()->select(oclIsTypeOf(UseCase))->
size()>=3) and (self.allOwnedElements()->
select(oclIsTypeOf(UseCase))->size()<=5)
```

R_{i13}: A use case diagram should not contain more than 20 use cases.

```
context uml::Model
inv: UseCase.allInstances()->exists(uc|uc->size()<20
```

R_{i19}: Each package should not contain more than 20 classes.

```
context uml::Package
inv: self.allOwnedElements()->select(oclIsTypeOf(Class))->size() < 20
```

R_{i23}: An <<entity>> class should contain at least 3 attributes.

```
context uml::Class
inv: self.attribute->size()>=3 and self.getAppliedStereotypes().
name->includes('entity')
```

R_{i24}: A <<control>> class should contain 2 to 5 operations.

```
context uml::Class
inv: (self.ownedOperation->size()>=2 or self.ownedOperation->
size() <= 5) and self.getAppliedStereotypes().name->includes('control')
```

R_{i25}: If a class is an empty class, then it must be a <<boundary>> class.

```
context uml::Class
inv: self.allOwnedElements()->size() = 0 and
self.getAppliedStereotypes().name->includes('boundary')
```

Q.4: Is model compliant to UML coding conventions?

a) Rules that determine the *incomplete model* compliance to UML coding or style conventions.

R_{i6}: A use case name should contain 1 to 4 words.

```
context uml::UseCase
inv: name.size() = 0 or ( let idx:Sequence(Integer) =
Sequence{1..name.size()} in idx->select(i| name.substring(i, i) = ' ')->
size()+1 <=4)
```

R_{i8}: A subsystem name should start with a capital letter, and should be consisting of one to two words.

```
context uml::Package
inv: (let startsWith:String=name.substring(1,1) in startsWith.toUpper()=
startsWith) and (name.size()=0 or ( let idx:Sequence(Integer)=
Sequence{1..name.size()} in idx->select(i| name.substring(i, i) = ' ')->
size()+1 <=2))
```

R_{i9}: Each actor name should start with a capital letter.

```
context uml::Actor
inv: (let startsWith:String = name.substring(1,1) in startsWith.toUpper()=
startsWith) and (name.size() = 0 or (let idx:Sequence(Integer)=
Sequence{1..name.size()} in idx->forAll(i| name.substring(i, i) <> ' ')))
```

R_{i11}: Each system name should start with a capital letter and contain one to two words.

```
context uml::Model
inv: (let startsWith:String = name.substring(1,1) in startsWith.toUpper()=
startsWith) and (name.size() = 0 or ( let idx:Sequence(Integer)=
Sequence{1..name.size()} in idx->select(i| name.substring(i, i) = ' ')->
size()+1 <=2))
```

R_{i12}: An Actor must be placed outside the system.

```
context uml::Model
inv: (self.allOwnedElements()->exists(oclIsTypeOf(Actor)))
```

R_{i22}: Each class name should start with a capital letter and should be one word.

```
1 context uml::Class
2 inv: (let startsWith:String = name.substring(1,1) in startsWith.toUpper()=
3  startsWith) and (name.size() = 0 or ( let idx:Sequence(Integer)=
4 Sequence{1..name.size()}in idx->forAll(i| name.substring(i, i) <> ' ')))
```

R_{i26}: Each association must have name.

```
1 context uml::Association
2 inv: self.name <> ' '
```

R_{i29}: Each association name should start with a lower case letter.

```
1 context uml::Association
2 inv: let startsWith:String = name.substring(1,1) in
3 startsWith.toLower()= startsWith
```

R_{i36}: Each message must be labeled.

```
1 context uml::Message
2 inv: self.name <> ' '
```

The following paragraph is described for the *complete* models, where quality model for UML is tailored for the design phase of software development process where implementation specific diagrams are used. For *complete model*, the rules and guidelines are selected in the same manner as defined for the *incomplete model*.

4.5.4 Analyzability for Complete Models

The factors for the measurement of analyzability for *complete* model is the same as we described for the *incomplete model*, i.e., consistency and traceability.

Q.1: Is the *complete model* consistent?

a) Rules or guidelines measuring or determining the consistency between use case diagram and sequence diagram.

R_{c23}: Each sequence diagram should have at least one actor on a lifeline (Same as R_{i32}).

Q.2: Is the *complete model* traceable?

a) Rules measuring or determining traceability between sequence and class diagram.

R_{c24}: Each object or lifeline in a sequence diagram must have a corresponding class in a class diagram (Same as R_{i33}).

R_{c25}: Every call message received by a lifeline must have a corresponding operation in the class (Same as R_{i34}).

R_{c26}: If there is a message call between two lifelines then there must also be an association between the two classes (Same as R_{i35}).

b) Rules measuring or determining traceability between activity diagram and class diagrams.

R_{c28}: One Activity diagram should reference to one class operation.

```
1 context uml::Activity
2 inv: self.specification.oclIsTypeOf(Operation)->size()==1
```

R_{c32}: Each activity in an activity diagram should reference a class operation.

```
1 context uml::CallBehaviorAction
2 inv: self.specification.oclIsTypeOf(Operation)->size()>1
```

R_{c34}: Each object of an activity diagram should have a corresponding class in a class diagram.

```
1 context uml::CentralBufferNode
2 inv: self.type->exists(oclIsTypeOf(Class))
```

4.5.5 Changeability for Complete Models

Below are appropriate rules or guidelines for the *changeability* quality attribute.

Q.1: How are the classes coupled in a class diagram?

a) Rules or guidelines measuring or determining coupling between classes.

R_{c3}: The depth of inheritance level should be less than 4 (Same as R_{i20} but with different threshold value because *complete* models are made at design level, therefore they are more complex.)

R_{c4}: Multiple inheritance must not exist (Same as R_{i21}).

R_{c8}: Each class should have 1-5 associations [5] (Same as R_{i28}).

Q.2: How are the packages coupled within model?

a) Rules or guidelines measuring or determining coupling between packages.

R_{c15}: The maximum package nesting level should be 2.

```
1 context uml::Package
2 inv: (self.owner.owner.owner->size())=0
```

Q.3: Are states in state machine diagram unique?

a) Rules or guidelines measuring or determining duplicate names of sates in state machine diagram.

R_{c36}: State names must be unique.

```
1 context uml::State
2 inv: State.allInstances()->forAll (p,q|p.name<>q.name implies p=q )
```

4.5.6 Understandability for Complete Models

The understandability for *complete model* is measured based on the same factors considered for *incomplete model*. The following questions refine the goal measurement.

Q.1: What is the structural complexity of the model?

a) Rules that measure structural complexity of the *complete model*.

R_{c10}: Each association must have a direction.

```
1 context uml::Association
2 inv: (self.memberEnd.isNavigable()->includes(false))
```

R_{c11}: Each association must specify multiplicity and it must be n to 1.

```
1 context uml::Association
2 inv: let opposite:Property = association.memberEnd->
3 any(e|e <> self) in (not opposite.oclIsUndefined() and
4 not upperValue.oclIsUndefined()) implies (upper = 1)
```

R_{c14}: The links to classes belonging to another package must be unidirectional (Same as R_{i31})

R_{c31}: Each activity diagram should contain one initial node and one exit point.

```
1 context uml::Activity
2 inv: self.allOwnedElements()->select(oclIsTypeOf(InitialNode))
3 ->size() = 1 and self.allOwnedElements()->select(oclIsTypeOf(
4 ActivityFinalNode))->size() = 1
```

R_{c37}: All states except root state and initial state should have at least one incoming transition.

```
1 context uml::StateMachine
2 inv: PseudoState.allInstances()->select(PseudoState=
3 PseudoStateKind::initial))->size()>=1
```

Q.2: What is the behavioral complexity of the model?

a) Rules that measure behavioral complexity of the *complete model*.

R_{c1}: Every class should have attributes.

```
1 context uml::Class
2 inv: self.ownedAttribute->size()>0
```

R_{c2}: Every class should have operations.

```
1 context uml::Class
2 inv: self.ownedOperation->size()>0
```

R_{c16}: Each attribute must have data type and must be private.

```
context uml::Property
inv: self.visibility = VisibilityKind::private and
self.type->notEmpty()
```

R_{c21}: Each operation must have a return type.

```
context uml::Operation
inv: self.ownedParameter->exists(e|e.direction =
ParameterDirectionKind::return)
```

R_{c22}: Each parameter must have a data type.

```
context uml::Parameter
inv: self.type->notEmpty()
```

Q.3: What is the overall complexity of *complete model*?

a) Rules that measure overall complexity of the *complete model*.

R_{c6}: Each class should have a maximum of 10 operations.

```
context uml::Class
inv: self.ownedOperation->size() <= 10
```

R_{c13}: Each subsystem / package should have maximum 20 classes.

```
context uml::Package
inv: (self.allOwnedElements()->select(oclIsTypeOf(Class))->size()) <= 20
```

R_{c18}: Each operation should have a maximum of 4 parameters.

```
context uml::Operation
inv: self.ownedParameter->reject(e|e.direction =
ParameterDirectionKind::return)->size()<=4
```

R_{c29}: The maximum number of decision points in activity diagram should be 12.

```
context uml::Activity
inv: self.allOwnedElements()->select(oclIsTypeOf(DecisionNode))
->size() <= 12
```

R_{c30}: Each activity diagram should contain 0 to 3 swim lanes.

```
context uml::Activity
inv: self.allOwnedElements()->select(oclIsTypeOf(ActivityPartition))
->size() <4
```

R_{c33}: Dead activities must not present in an activity diagram.

```
context uml::Action
inv: self.incoming ->size()<> 0 and self.outgoing->size() <> 0
```

R_{c35}: Dead states must not be present in a state machine diagram.

```
context uml::State
inv: self.incoming ->size()<> 0 and self.outgoing->size() <> 0
```

Q.4: Is the model compliant to UML coding conventions?

a) Rules that determine the *complete model* compliance to the UML coding or style conventions.

R_{c5}: Each class must should with a capital letter and should be one word. (Same as R_{i22})

R_{c7}: Each association must have a name (Same as R_{i26}).

R_{c9}: Each association name should start with a lower case letter (Same as R_{i29}).

R_{c12}: Association classes must not present in a model.

```
1 context uml::Model
2 inv: AssociationClass.allInstances()->size()=0
```

R_{c17}: If a class has a composition relationship, then the multiplicity must be 1 at the side of the owner.

```
1 context uml::
2 inv: let opposite:Property = association.memberEnd->any(e|e <> self)
3 in (not opposite.oclIsUndefined() and opposite.isComposite and not
4 upperValue.oclIsUndefined()) implies (upper = 1)
```

R_{c19}: Any <<entity>> class should have getters and setters.

```
1 context uml::Class
2 inv: self.ownedOperation->exists(name.substring(1, 3 ) =
3 'set' or name.substring(1, 3 ) = 'get')
```

R_{c20}: An abstract class should have abstract operations.

```
1 context uml::Operation
2 inv: (self.isAbstract implies self.owner->exists(isAbstract))
```

R_{c27}: If a message is empty, then it must be a return type message.

```
1 context uml::Message
2 inv: self.name =' ' implies self.messageSort=MessageSort::reply
```

4.6 Metric Selections for Quality Assurance

Metrics are important to measure the quality of the models. We have already achieved this main objective by using rules and guidelines (Section 4.5) to detect issues on *incomplete* and *complete* models.

There are five major types of software measurement scales, these are, nominal, ordinal, interval, ratio and absolute [27]. The scale with most information and flexibility is the *ratio* scale, which permits most sophisticated analysis. Measurements such as *number of defects* are ratio measurement and on UML models the ratio measurement relates to the *number of issues detected in the*

model. Another measurement could be to measure the physical size of the UML models. The ratio measurement scale starts at zero, which represents the non-existence of any issue in the UML model, while a 1.0 value indicates possible high values for detected issues in the model.

In addition, ratio metrics are useful to measure the quality of UML model by considering faults per element. The increase in faults per elements, decreases the quality of UML model and vice versa. In the following, we describe a general ratio metric, that can be adopted for the rules or guidelines described in Section 4.5.

M= Number of violations of the context element/Total number of context element

Furthermore, we present example ratio metrics. This include the M_{i1} metric, which is derived from R_{i1} of *incomplete model*. We look for the violations of R_{i1} and divide the total number of use cases in a use case diagram. The result zero represents that either there is no violation of rules or guidelines or there is no corresponding diagram in the model. Where as one represents that all rules have been violated.

M_{i1} = *Number of use cases outside the subsystem/Total number of the use cases in use case diagram*

The second example metric M_{c13} is a guideline from the list of rules and guidelines for *complete* models, where we measure the ratio metric for the R_{c13} of *complete* model.

M_{c13}= *Number of packages that contain more than 20 classes/Total number of packages*

This approach is extended based on the quality assessment results for the bakery system presented in Chapter 6.

Chapter 5

Implementation

This chapter presents the technologies used for the implementation of the quality assessment and quality improvement approach, and the implementation of the prototype tool.

5.1 Eclipse and Associated Technologies

This section focuses on Eclipse and associated technologies used in the implementation. The overview of these technologies serves as a base for the understanding of the reader, that how the approach is implemented and how the different parts collaborate.

5.1.1 The Eclipse Modeling Project (EMP)

Eclipse provides a whole range of modeling tools and frameworks, as a part of the Eclipse Modeling Project (EMP). The collection of tools was formed to coordinate and focus on Model Driven Development (MDD) technologies. EMP is a top level project which is logically organized into sub-projects that provide abstract syntax definitions, concrete syntax development, Model-to-Model (M2M) transformation, and Model-to-Text (M2T) transformation.

5.1.2 The Eclipse Modeling Framework (EMF)

Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for the building of tools and other applications. XMI is used to describe models and the EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified

using annotated Java, UML, XML documents, or modeling tools. Additionally, EMF provides the foundation for interoperability between EMF-based tools and applications [92]. Figure 2.2 [page 8] shows both graphical as well as an EMF representation of a UML model, in which each UML notation is described in XMI.

5.1.3 The XPand Project

The Xpand project focuses on the generation of textual artifacts from models and consists of various languages and components. Xpand is a Model-to-Text (M2T) transformation language featuring polymorphic template invocation, functional extension, model validation and model transformation based on the Xtend language. The implementation of our prototype tool is based on M2T the transformation with Xpand and M2M transformation with Xtend. In the following both languages are discussed in detail.

5.1.3.1 The Xpand Code Generation Language

Xpand is a modern template-based code generation language (i.e., Model-to-Text (M2T)) which has become popular in a very short time. Xpand language was originally developed as part of openArchitectureWare (oAW) [65] project before it became part of the EMP. Xpand language provides a powerful mechanism with its limited vocabulary.

In a single M2T transformation project, one or more than one Xpand language template can be defined. Each Xpand language template is defined using **DEFINE** and **ENDDEFINE** blocks. The Listing 5.1 shows a simple **DEFINE** block, where *simpleTemplate* is the name of the block, and *Type* is the type of the element, on which this block should be executed. Inside the block statements can be defined.

```
1 «DEFINE simpleTemplate FOR Type»
2
3   some statements...
4
5 «ENDDEFINE»
```

Listing 5.1: Simple Xpand Template

Xpand language provides multiple features to enable code generation in a smooth way. The following is a set of the important statements in Xpand language.

5.1.3.1.1 The IMPORT Statement

The **IMPORT** statement enables importing namespace and using the unqualified names contained in the imported namespace. Listing 5.2 imports the name spaces of UML metamodel for the current template.

```
1 «IMPORT uml»
```

Listing 5.2: Import Statement

5.1.3.1.2 The EXTENSION Statement

Extensions provide a flexible and convenient way of defining additional features of metaclasses. In Listing 5.3 An **EXTENSION** import points to the Xtend language file containing the required extensions.

```
1 «EXTENSION my::ExtensionFile»
```

Listing 5.3: Extension Statement

5.1.3.1.3 The FOREACH and EXPAND Statements

These statements enable the retrieval of a specific collection and its manipulation inside the body. **FOREACH** is used in Xpand language templates for retrieving a specific collection of model elements, as described in Listing 5.4, whereas *rule1* is the name of the **DEFINE** block that should be expanded for each use case. The **EXPAND** statement expands another **DEFINE** block (in a separate context), inserts its output at the current location and continues with the next statement. This is similar in concept to a subroutine call.

```
1 «EXPAND rule1 FOREACH eAllContents.typeSelect(UseCase)»
```

Listing 5.4: The FOREACH and EXPAND Statements

5.1.3.1.4 IF Statement

The **IF** statement supports conditional expansion. Any number of **ELSEIF** statements are allowed. The **ELSE** block is optional. Every **IF** statement must be closed with an **ENDIF**. The body of an **IF** block can contain any other statement, specifically, **IF** statements may be nested. Listing 5.5 shows a simple example of an **IF** block.

```
1  «IF expression»
2    statement
3      ....
4    «ELSEIF expression»
5      statement
6      ....
7    «ELSE»
8      statement
9      ...
10 «ENDIF»
```

Listing 5.5: IF Statement

5.1.3.1.5 REM Statement

The **REM** statement is used for comments. Listing 5.6 shows **REM** statement, the comments are written inside the block.

```
1 «REM»
2  This is a sample REM block in Xpand template.
3 «ENDREM»
```

Listing 5.6: REM Statement

5.1.3.2 The Xtend Model-to-Model (M2M) Transformation Language

The Model-to-Model (M2M) language Xtend is a part of the Xpand project. Xtend language can also help in formulating readable Xpand language templates by defining reusable operations that access the source model and retrieve data from it. In the prototype Xtend is used for two purposes: with OCL used in Xpand language templates to generate reports, and for refactoring of the UML models. Listing 5.7 shows a sample Xpand language template used to generate a HTML report for violated rules, which calls *myXtendFunction()* from the Xtend language file.

```
1 «DEFINE ruleNumber FOR uml::Element»
2       «IF myXtendFunction()===false»
3           <TR>
4           <TD>"Description of rule"</TD>
5             <TD>«qualifiedName»</TD>
6         </TR>
7       «ENDIF»
8 «ENDDEFINE»
```

Listing 5.7: XPand Language Example for M2T Transformation

Listing 5.8 shows, the definition of the *renameElement()* function in the Xtend, which sets new name for the UML element.

```
1 Boolean renameElement(uml::Element elem):
2   elem.setName("NewName")->true;
```

Listing 5.8: Xtend Language Example for M2M Transformation

5.1.4 Modeling Workflow Engine (MWE)

The Modeling Workflow Engine (MWE) is a generator engine, which can be configured to run independently or within Eclipse. MWE uses an XML based configuration language to setup a generator workflow, which may consist of one or more workflows. The workflow engine is used to execute transformations of models, i.e. M2T and M2M transformations. In Listing 5.9, Line 1 shows that Xtensible Mark-up Language (XML) version for the wokflow generator and encoding for windows operating system. Line 4 shows how to read a UML model. The name attribute is used to provide a name for the slot on which model should

be stored and value attribute take the uml model as an input from the specified location. Lines 6-7 show how to setup the out put directory, where the results are stored. Line 9 shows the setup of the metamodel, in our case UML metamodel is configured for this generator. Line 11 is used to instantiate the metamodel. Line 13-16 are used to read the EMF representation of UML model using XMI reader Eclipse component. Lines 18-20 show how to clean the output directory first. Lines 22-31 show how to invoke M2T transformation generator engine to create HTML reports.

```
1  <?xml version="1.0" encoding="windows-1252"?>
2  <workflow>
3    <!-- model which needs to be analyzed -->
4    <property name="model" value="myModel.uml" />
5    <!-- set the output directory-->
6    <property name="modeldir" value="myModel"/>
7    <property name="src-gen" value="src-gen/${modeldir}" />
8    <!-- Setup UML2 support -->
9    <bean class="org.eclipse.xtend.typesystem.uml2.Setup" standardUML2Setup="true" />
10   <!-- instantiate metamodel -->
11   <bean id="mm_emf" class="org.eclipse.xtend.typesystem.emf.EmfRegistryMetaModel"/>
12   <!-- load uml model -->
13   <component class="org.eclipse.xtend.typesystem.emf.XmiReader">
14       <modelFile value="${model}" />
15       <outputSlot value="modelSlot" />
16   </component>
17   <!-- Clean src directory -->
18   <component id="dirCleaner"
19     class="org.eclipse.emf.mwe.utils.DirectoryCleaner"
20     directory="${src-gen}$"/>
21   <!-- create html metrics report -->
22   <component class="org.eclipse.xpand2.Generator">
23     <metaModel idRef="mm_emf"/>
24     <fileEncoding value="ISO-8859-1"/>
25     <expand
26       value="templates::root::main FOR modelSlot" />
27     <outlet path="${src-gen}$" overwrite="false">
28       <postprocessor class="org.eclipse.xtend.typesystem.xsd.XMLBeautifier">
29       </postprocessor>
30     </outlet>
31   </component>
32 </workflow>
```

Listing 5.9: Example of a WorkFlow generator

5.2 Tool Implementation

This section presents the implementation of the prototype for quality assessment and quality improvement approach. The implementation is divided into two parts: the violation of rules or guidelines are described in Section 5.2.2 and the refactorings are described in Section 5.2.3.

5.2.1 Common Infrastructure

In the Eclipse IDE, each project is managed with the project dependencies. Listing 5.10 shows all the dependencies for running the workflow properly for both quality assessment and improvement approaches for the prototype tool. Lines 1-2 show the Manifest version numbers. Lines 3-4 represent the name of our Eclipse project. Line 5 shows the bundle version of our prototype tool. Lines 6-17 show all the required bundles and Line 28 shows the required execution environment. Line 19 shows the imported OCL package and Lines 20-21 show the UML dependencies for OCL component.

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: swe.uml.rules
4  Bundle-SymbolicName: swe.uml.rules; singleton:=true
5  Bundle-Version: 1.0.0
6  Require-Bundle: org.eclipse.jdt.core;bundle-version="3.5.0",
7    org.apache.log4j;resolution:=optional,
8    org.antlr.runtime;bundle-version="3.0.0",
9    org.eclipse.core.runtime;bundle-version="3.5.0",
10   org.eclipse.emf.mwe.utils;bundle-version="0.7.0",
11   org.eclipse.xpand;bundle-version="0.7.0",
12   org.eclipse.xtend;bundle-version="0.7.0",
13   org.eclipse.xtend.util.stdlib;bundle-version="1.0.0",
14   org.eclipse.xtend.typesystem.xsd;bundle-version="1.0.0",
15   org.eclipse.xtend.typesystem.uml2;bundle-version="1.0.0",
16   org.eclipse.xtend.profiler.source;bundle-version="1.0.0"
17   org.eclipse.uml2;bundle-version="3.0.0",
18 Bundle-RequiredExecutionEnvironment: J2SE-1.5
19 Import-Package: org.eclipse.ocl.uml,
20   org.eclipse.ocl.uml.options,
21   org.eclipse.ocl.uml.util
```

Listing 5.10: Project Dependencies and the Project Information

5.2.2 Implementation of the Quality Assessment Approach

This section describes how the quality assessment approach is implemented using the Xpand modeling project.

5.2.2.1 OCL Evaluator

The OCL Evaluator class is the main class for executing the OCL queries. This class has been adopted from the Eclipse OCL Interpreter example [22], which is part of the OCL Eclipse project. It describes the usage of OCL expressions on a model. It includes a parser/interpreter for OCL that works on EMF models.

Listing 5.11 illustrates Java code that evaluates a single query. The context object contains the UML model that will be validated. First, a factory is initialized with the model (Line 1). The enumeration object named *ModelingLevel* is set to M2, as

all the OCL constraints operate on OMG's M2-level, i.e., on metamodel elements (Line 2). An OCL object is built using the factory and modeling level objects, and a corresponding helper is created that will parse the string containing the OCL expression (Line 3–5). The method *setContext* in Line 7 associates the given context to the helper. Since the modeling level is always M2, the first case block is executed. The variable expression is a string containing the OCL constraint. In Line 8, the helper parses it and produces an OCLExpression object. Now, the OCL object can evaluate the parsed expression on the UML model contained in the context (Line 9). Lines 10-13 present the check, whether the query has successfully parsed or not. Line 14-16 is used to catch the parser exceptions.

```
1  IOCLFactory<Object> oclFactory = new UMLOCLFactory(context);
2  ModelingLevel modelingLevel = ModelingLevel.M2;
3  OCL<?, Object, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?> ocl;
4  ocl = oclFactory.createOCL(modelingLevel);
5  OCLHelper<Object, ?, ?, ?> helper = ocl.createOCLHelper();
6  try {
7      modelingLevel.setContext(helper, context, oclFactory);
8      OCLExpression<Object> parsed = helper.createQuery(expression);
9      Object results = ocl.evaluate(context, parsed);
10     if (results instanceof Boolean) {
11         boolean bool = (Boolean) results;
12     if (bool == false)
13         issues.addError("OCL Check failed: " + currentDescription);
14   }catch (ParserException e) {
15     issues.addError(e.getMessage());
16  }
```

Listing 5.11: Sample Code to Execute OCL Query

5.2.2.2 Java Extension in Xtend Language

Java extensions are used to express logic that can not be expressed using Xtend. Since file *OCLEvaluator* is implemented in Java, hence Java extension is required to execute this function within Xtend language. Listing 5.12 illustrates two Java extensions. The function *dump()* on (Line 2) is used to call *Helper* Java class (Line 3), that prints messages in the console, which helps debugging. The *evaluateOCL()* method (Line 5) used to call the *OCLEvaluator* class (Line 6) to execute OCL queries.

```
1  import uml;
2  cached Void dump(String s) :
3    JAVA helper.Helper.dump(java.lang.String);
4
5  cached Void evaluateOCL(String ms, uml::Element model) :
6    JAVA helper.OCLEvaluator.evaluateOCL(java.lang.String, org.eclipse.uml2.uml.Element );
```

Listing 5.12: Java Extension in Xtend Language

5.2.2.3 Rules and Guidelines in Xtend Language

The implementation of OCL rules and guidelines in Xtend is illustrated in Listing 5.13, which shows the implementation of R_{c6} [Section 4.5.6, page 54]. In Line 2, a *cached* keyword is used to cache the query result, so that it can be reused in the Xpand templates to save execution time. The method parameter is the context for the OCL query, which is a *Class*. Line 3 describes the OCL query as a string. The query is passed as a string to the *evaluateOCL()* function (Line 4). It returns a boolean value. The complete implementation of rules and guidelines is described in Appendix E for *incomplete model* and Appendix F for *complete model*.

```
1 // R6 Each class should have maximum 10 operations.
2 cached Boolean totalOperations(uml::Class cs):
3   let query ="self.ownedOperation->size() <= 10":
4     query.evaluateOCL(cs);
```

Listing 5.13: Implementation of Rule in Xtend Language

5.2.2.4 Quality Assessment Report Generation with Xpand

Quality assessment results are stored as HTML documents. The process of generating a report is described as M2T transformation Xpand language. As discussed earlier, that functions described in Xtend language can be called in the Xpand templates.

```
1 «IMPORT uml»
2 «DEFINE main FOR Model»
3   «REM»Report for analysis model «ENDREM»
4   «EXPAND templates::analysisModel::analysisModel»
5 «ENDDEFINE»
```

Listing 5.14: Root Xpand Language Template

The root Xpand language template is illustrated in the Listing 5.14. Line 1 signifies that the template is defined for the UML metamodel and Line 2 means that the template works on the UML model. Line 4 describes the structure to execute the *analysisModel* or *designModel* templates, which are defined in the templates package.

The *analysisModel* template is shown in Listing 5.15, which calls the OCL based rules defined in Xtend. Line 1 shows that *rule6* is the name of the **DEFINE** block and it works on all classes present in the UML model. In Line 2 *totalOperation()* Xtend function is described for class elements. Therefore it is necessary to take care that the context element in both the Xtend function and the **DEFINE** block must be the same, otherwise workflow generates an error message. Line 3 checks the boolean results. If it violates the rule, the result is stored in the HTML file with its location (Line 4-9).

```
1  «DEFINE rule6 FOR uml::Class»
2      «LET totalOperations() AS maxOperation»
3          «IF maxOperation==false»
4          <TR>
5            <TD><FONT FACE="Geneva, Arial" SIZE=2>
6              "Rc6: Each Class Should have maximum 10 operations"
7            </TD>
8              <TD>«getQualifiedName()»</TD>
9          </TR>
10         «ENDIF»
11       «ENDLET»
12 «ENDDEFINE»
```

Listing 5.15: Partial Xpand Language Template to Generate Html Report

5.2.2.5 Modeling Workflow Engine (MWE) for Quality Assessment

The workflow generator generates the HTML output for the violated rules. Listing 5.16 shows the Xpand language component which invokes the root template, described in Listing 5.14. Line 5-6 refer to the templates, which is the name of the package. The root is an Xpand template file inside the templates package, where main is a **DEFINE** block, which is invoked by the generator.

```
1  <!-- create html report for violated rules-->
2    <component class="org.eclipse.xpand2.Generator">
3      <metaModel idRef="mm_emf"/>
4      <fileEncoding value="ISO-8859-1"/>
5      <expand
6        value="templates::root::main FOR modelSlot" />
7      <outlet path="${src-gen}$" overwrite="false">
8        <postprocessor class="org.eclipse.xtend.typesystem.xsd.XMLBeautifier">
9        </postprocessor>
10     </outlet>
11   </component>
```

Listing 5.16: MWE Xpand Language Component

5.2.3 Implementation of the Quality Improvement Approach

This section presents the implementation details of the quality improvement approach using the Xtend M2M transformation language.

5.2.3.1 Refactorings in Xtend Language

In this section, the implementation of two refactorings i.e., Rename and Pull up method are described. The basic concepts of these two refactorings are described in Sections 2.6.1 and 2.6.2. These two refactorings are used to show the applicability of an automated approach for the quality improvement of UML models.

5.2.3.1.1 Rename Refactoring

Listing 5.17 describes the rename refactoring in a generic way that can be applied to any UML element, for example, use cases, classes, operations and so on. Line 1 illustrates that refactoring works on the UML metamodel. Line 2 defines the *transform()* function, which is called in the MWE to execute this refactoring. All elements are stored in a list in Line 3, while Line 4 calls the *renameElement()* function and the *findElement()* function returns the element that needs to be refactored (Lines 7-8). Lines 10-11 show the *renameElement()* function in which the element is given a new name.

```
1  import uml;
2  uml::Model transform(uml::Model model):
3    let elementList = model.eAllContents.typeSelect(UML::Element).collect(e|e):
4    elementList.forAll(e|renameElement(findElement(elementList, "Name of the Element")))->
5      model;
6  // find uml::Element in the model
7  List[uml::Element] findElement(List[uml::Element] elem, String name):
8    elem.select(e|e.name == name);
9  // rename UML::Element in the model
10 Boolean renameElement(List[uml::Element] elem):
11   elem.setName("New Name for the Element")->true;
```

Listing 5.17: Rename Refactoring

5.2.3.1.2 PullUp Method Refactoring

The implementation of the Pullup Method refactoring is described in Listing 5.18. Line 1 means that this refactoring works on the UML metamodel. Line 2 defines the *transform()* function which takes the UML model as an argument and returns the result as a UML model. Lines 4-8 represent how to look for the existence of a superclass and its subclasses in the hierarchy and save each of them in a different list (Line 10). Lines 12-13 pick a class and check for identical operations. Lines 18-28 define the *collectOperations()* method, which save the identical operations. If identical operations are found in the subclasses then move the operation from the subclasses to the super class(Lines 30-36). Lines 38-41 show how the identical operations are deleted from the subclasses. A new operation is created in superclass as shown in (Lines 43-46).

```
1  import uml;
2  uml::Model transform(uml::Model model):
3    // generalization class
4    let generalClass = model.eAllContents.typeSelect(Class).
5                       generalization.collect(e|e):
6    // list with all classes that inherit from the superclass
7    let classlist = model.eAllContents.typeSelect(Class).select(
8                c|c.superClass.exists(s|s.name == generalClass)) :
9    // list with operations to be moved
10   let operationlist = newList() :
11   // 1- pick one class and check the operations
12   classlist.first().ownedOperation.forAll(o|collectOperations(
13                       o, classlist, operationlist)) ->
14   // 2- delete all operations in list
15   operationlist.forAll(o|pullUpOperation(
16              (String)o, classlist, generalClass)) ->model;
17 // collect operations
18 Boolean collectOperations(Operation o, List classlist, List operationlist) :
19      classlist.forAll(c|((Class)c).ownedOperation.exists(op|op.name == o.name))
20      ? (// operation found in all classes
21        // put method in list
22        operationlist.add(o.name) ->
23        dump("Added operation '"+o.name+"' to list")
24      )
25      : (// operation not in all classes
26        dump("Operation '"+o.name+"' exists not in all classes of list")
27      ) ->
28      true;
29 // Pull Up the operation to the super class
30 Boolean pullUpOperation(String o, List classlist, List[uml::Class] generalClass) :
31      //create Operation in superClass
32      generalClass.ownedOperation.add(newOperation(o)) ->
33      //remove operation in classes of classlist
34      classlist.forAll(c|deleteOperation((Class)c, o)) ->
35      dump("Operations '"+o+"' of classes in list successfully moved to superclass") ->
36      true;
37 // delete the operations from the child classes.
38 Boolean deleteOperation(Class c, String oName):
39   c.ownedOperation.remove(c.ownedOperation.selectFirst(o|o.name == oName))->
40   dump("Removed operation "+oName+" from class "+c.name) ->
41   true;
42 // create a new class to the super class with same signature to the child class operation
43 create Operation newOperation(String name) :
44   this.setName(name);
45   create List newList():
46   this;
```

Listing 5.18: Pull Up Refactoring

5.2.3.2 The Modeling Workflow Engine (MWE) for Refactoring

The workflow generator is used to invoke the M2M transformation. Listing 5.19 illustrates the Xtend language component (Lines 2-6) used for M2M transformation. Lines (2-3) refer to the Xtend language component based on the UML metamodel. The refactoring is invoked in Line 4, where *renameElement* is the name of the Xtend language file and *transform()* is the Xtend function described in the

Listing 5.17. In Lines 9-12 EMF writer component is used to write the EMF representation of UML model into the output directory. It writes the transformed model into the *src-gen* directory.

```
1  <!-- model to model transformation -->
2    <component class="org.eclipse.xtend.XtendComponent">
3      <metaModel class="org.eclipse.xtend.typesystem.uml2.UML2MetaModel" />
4      <invoke value="renameElement::transform(modelSlot)"/>
5      <outputSlot value="outputSlot"/>
6    </component>
7
8  <!-- write output UML model -->
9    <component id="writer" class="org.eclipse.emf.mwe.utils.Writer">
10     <modelSlot value="modelSlot"/>
11     <uri value="./src-gen/Transformations/tranformedModel.uml"/>
12   </component>
```

Listing 5.19: MWE Xtend Language Component

Chapter 6

Case Study

This chapter presents a case study, its academic context, a description and details of the quality assessment results for the bakery system introduced in Section 2.2. Furthermore, size and ratio metrics for *incomplete* and *complete* models are given. In the end, the case study feedback and conclusions are discussed.

6.1 Academic Context and Learning Objectives

The case study in this thesis is primarily related to a UML practical course, in which UML models were developed for the bakery system. The description of the bakery system, how the bakery system model is developed, and which UML diagrams are suitable for the analysis and design models have already been introduced in Section 2.2 [page 12]. The analysis model is considered as the *incomplete model* and the design model considered to be the *complete model* based on our proposed quality model for UML, which is described in Chapter 4.

A UML practical course was announced for the students to learn the UML modeling language and the duration of the course was three weeks. During that course, their task was to develop the analysis and design models. These models were used for the quality assessment based on the proposed quality model for UML and models were refactored by the students based on the quality assessment results. Due to time constraints, only two iterations were performed for the quality assessment and quality improvement approach. The language of instruction for the course was German language. A complete description of the bakery system in the German language is described in Appendix A.

The UML practical course was designed to allow students to develop UML models for the bakery system by applying many of the principles learned through their course work. eleven Bachelor and three Masters students of computer science at the University of Goettingen, Germany, participated in this course. The students had a basic knowledge about UML and object oriented programming concepts. Four students were in fourth semester. Seven students were in sixth semester, and three students were in tenth semester. In this practical course, lectures were presented on the Unified Process, and how UML fits into each phase of the process. The assignments related to the lecture topics were usually based on the development of a UML model from the requirements. The primary goal for using the bakery system as an example in a UML practical course was that students can learn the UML modeling language in a simple way and how to use UML in the RUP process.

As already described, fourteen students participated in this practical course. These students were divided into two groups (BLUE and RED), so that two models could be processed for the quality assurance of this case study. Each group was further sub-divided based on the decomposition of their models, so that they could work on different parts of the model. The three week UML course was managed as follows: at the end of the first week, students handed over their analysis model, and on Monday, they got the overall quality assessment results for those models. The students performed refactorings on the models on Monday. On Tuesday their models again went through the quality assessment tool for purposes of evaluating them. The same process is repeated for the design model in the second week and in the last week the results of both models were discussed and the students provide feedback about the course.

6.2 Quality Assessment Results for the Bakery System

This section presents the quality assessment results of the BLUE and RED group for the *incomplete model* and *complete model*.

6.2.1 Quality Assessment Results for Incomplete Model

The analysis model for the bakery system is considered here to be an *incomplete model*. The quality attributes *analyzability*, *changeability* and *understandability* were considered for the quality assessment of the *incomplete model* types. The BLUE and RED group did not violate any rule or guideline for the *changeability* quality attribute. Therefore, the following paragraph only discusses the results for the *analyzability* and *understandability* quality attributes for both groups.

6.2.1.1 BLUE Group Incomplete Model

Table 6.1 shows the quality assessment results for the *analyzability* quality attribute and Table 6.2 shows the *understandability* related violations for the BLUE group.
Each table contains the name of the violated rule or guideline and the number of times that rule or guideline was violated in the model for first and second iteration. The results are described in the following paragraphs.

6.2.1.1.1 Analyzability for Incomplete Model

Table 6.1 shows the violations related to the *analyzability* quality attributes. The GQM based categorization of rules for *analyzability* is described in Chapter 4, page 46 for the *incomplete model*, hence this guideline was violated in both iterations.

R_{i16} [page 46] is violated for all three subsystems of the use case diagram in the first and second iteration. Students did not know how to use the UML tool to draw activity diagrams for the subsystems of a use case diagram.

R_{i17} [page 46] is violated for 20 activities in an activity diagram in their first iteration, while in the second iteration, there was only one single violation of this guideline. Our quality assessment tool, helped to improve the quality of the model.

R_{i18} [page 46] is violated for all three subsystems of a use case diagram in the first iteration. Students did not know, how to use the UML tool to describe a subsystem of a use case diagram as a package in a class diagram but with the help of the instructor, they managed to improve their models in the second iteration.

R_{i34} [page 47] is violated nine times in the first iteration, while in the second iteration, the R_{i34} was violated 48 times, which is five times more than in first iteration. This happened because they introduce new sequence diagrams, and messages on sequence diagrams were deleted from the diagram pane, whereas they still had messages in their model containment tree. Hence, our quality assessment tool detected these messages as a violation of R_{i34}.

6.2.1.1.2 Understandability for Incomplete Model

Table 6.2 presents the *understandability* related quality assessment results for the *incomplete model*. The GQM based selection of the rules and guideline is already described in Chapter 4, page 48 for the *understandability* quality attribute.

R.No	Name of the violated rule	No. of times the rule was violated (Iteration 1)	No. of times the rule was violated (Iteration 2)
R_{i16}	Each subsystem should be refined by one activity diagram	3	3
R_{i17}	Each activity in an activity diagram should refer to a use case in a use case diagram	20	1
R_{i18}	Each subsystem of a use case diagram should be represented as a package in the class diagram	3	0
R_{i34}	Every call message received by a lifeline should have a corresponding operation in the class	9	48

Table 6.1: BLUE Group's Violations of Rules for Analyzability of Incomplete Model Type

In the first iteration, R_{i1} [page 49] is violated one single time, while in the second iteration, they fixed this problem. Hence the quality assessment tool helped to improve the quality of the model.

R_{i5} [page 49] is a guideline for the modelers. However, when more use cases communicate to a single actor it means more responsibility for one actor, and this should be avoided. R_{i5} is violated seven times in first iteration, and in second iteration the problem was solved.

R_{i6} [page 50] is also a guideline, related to the naming convention and the length of the name string for a use case name should not be more than four words. The BLUE group violated R_{i6} three times in first iteration, but this was resolved in the second iteration.

R_{i7} [page 49] is also a guideline, that suggests the size of the subsystem. This guideline was violated for all three subsystems in first and second iteration. This was violated because too many use cases had been defined in first iteration. They tried to reduce the size of the subsystem in second iteration, but still they had one single violation.

R_{i22} [page 51] is related to the naming convention of a class. This was violated seven times in first iteration and resolved in second iteration.

R_{i26} [page 51] is a rule that was violated only once and resolved in second iteration. This helped in understanding the associated classes.

R_{i27} [page 48] suggests the addition of multiplicity values at both ends, and this was violated by the BLUE group only once. In the second iteration, no violations were found in their models. Multiplicity indicates how many instances of one class are required.

R_{i29} [page 51] is related to the naming convention of an association. In the first iteration, only one single violation was identified. In the second iteration, no violations were found for this guideline.

R_{i30} [page 48] is a guideline. In the first iteration, it was violated 17 times, while in the second iteration, they still had a one single violation. It is too early to decide an aggregation in the analysis stage, in that case we provide a guideline R_{i30} to the students, to the effect that they should not provide any aggregation or composition type relationship in their *incomplete model*.

R.No	Name of the violated rule	No. of times the rule was violated (Iteration 1)	No. of times the rule was violated (Iteration 2)
R_{i1}	Each use case must be inside the one subsystem	1	0
R_{i5}	A use case should not linked to more than three actors	7	0
R_{i6}	A use case name should contain 1 to 4 words	3	0
R_{i7}	Each subsystem should have a minimum of 3 and a maximum of 5 use cases	3	3
R_{i22}	Each class name should start with a capital letter and should be one word	7	0
R_{i26}	Each association must have a name	1	0
R_{i27}	Each association must specify multiplicity values at both ends	1	0
R_{i29}	Each association name should start with a lower case letter	1	0
R_{i30}	Classes should not be linked to a composition or aggregation association type	17	1

Table 6.2: BLUE Group's Violations of Rules for Understandability of Incomplete Model Type

The results described in the Table 6.1 and Table 6.2 are exemplified in Figure 6.1 for a sample use case diagram and Figure 6.2 for a sample sequence diagram and some classes from a class diagram are shown in Figure 6.3.

In Figure 6.1, the violation of R_{i6} [page 50] is highlighted. R_{i6} is related to the length of the use case name, which is more than four words, that leads to the violations of *understandability* UML coding and naming conventions described in Chapter 4, [page 50].

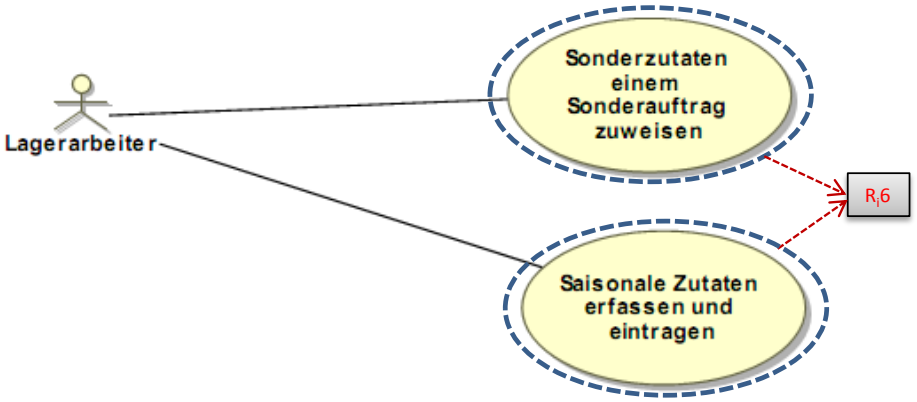


Figure 6.1: BLUE Group's Use Case Diagram of Incomplete Model Type

The sequence diagram and its corresponding classes are illustrated in the Figure 6.2, where the violations R_{i34} [page 47] were detected in sequence diagrams. This violation is related to the message between the lifelines. If there is a method call on the message, then that method must exist in the corresponding class in the class diagram. In Figure 6.2, messages are used for the method call but that corresponding method is not present in the corresponding class of the class diagram.

Figure 6.3 shows some extracted classes of the class diagram. R_{i22}, R_{i26} and R_{i29} are related to the UML naming or coding convention for the model. R_{i22} is related to the class name violation, for example, class name *Kundenverwaltung GUI* contains two words. R_{i26} and R_{i29} are related to the association name. None of the classes shown in Figure 6.3 contain any name for their associations. R_{i27} is related to the multiplicity values at the end of the association. In Figure 6.3, class associations do not have any multiplicity values at the ends of association. R_{i30} violation is related to the use of composition or aggregation type of associations in the *incomplete models*.

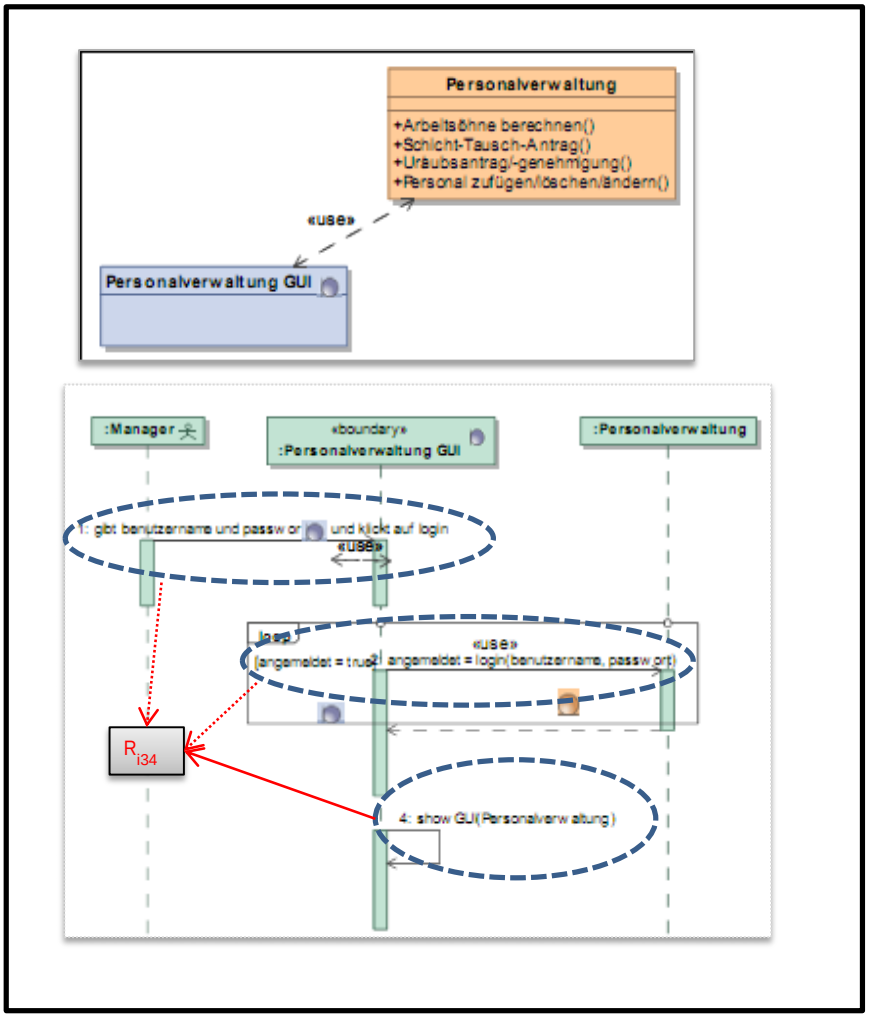


Figure 6.2: BLUE Group's Class and Sequence Diagram of Incomplete Model Type

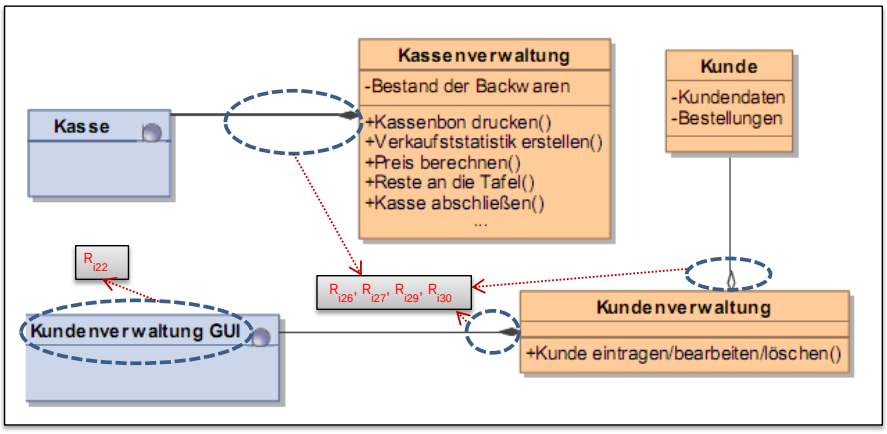


Figure 6.3: BLUE Group's Class Diagram of Incomplete Model Type

6.2.1.2 RED Group Incomplete Model

The results of the violation of the rules by the RED group are shown in Table 6.3 for the *analyzability* and in Table 6.4 for the *understandability* quality attributes. The description of the quality assessment results is presented in the following sections.

6.2.1.2.1 Analyzability for Incomplete Model

Table 6.3 presents the quality assessment results for the RED group.

R_{i16} [page 46] was not violated in the first iteration but in the second iteration the RED group violated this rule two times. R_{i16} relates to the consistency between a use case, and an activity diagrams. Activity diagrams are refined from the subsystem of use case diagrams. In the second iteration, the RED group introduced an additional activity diagram, and because of that the quality assessment prototype tool detects a violation of R_{i16}, which suggests a one to one mapping between the subsystem of a use case diagram and the activity diagram. Therefore, one subsystem should be refined by one activity diagram.

R_{i17} [page 46] was violated by the RED group more than the BLUE group in the first iteration, and that was 31 times. In the second iteration only 18 violations are removed, i.e., 13 violations are still present in their diagram. Due to the addition of new activity diagrams in second iteration, the activities in the activity diagrams increased and those activities were not consistent with the use case diagram. That means the activities were not traceable to the use cases in the use case diagram.

R_{i18} [page 46] was violated for all the three subsystems of a use case diagram in the first iteration, but they managed to improve their model in the second iteration. The violation of this rule refers to the inconsistency between the use case diagram and a class diagram, whereby the subsystems are not refined as packages in the class diagram.

R.No	Name of the violated rule	No. of times the rule was violated (Iteration 1)	No. of times the rule was violated (Iteration 2)
R_{i16}	Each subsystem should be refined by one activity diagram	0	2
R_{i17}	Each activity in activity diagram should refer to a use case in a use case diagram	31	13
R_{i18}	Each subsystem of use case diagram should be represented as a package in the class diagram	3	0

Table 6.3: RED Groups's Violations of Rules for Analyzability of Incomplete Model

6.2.1.2.2 Understandability for Incomplete Model

Table 6.4 shows the violations related to the *understandability* quality attribute for the RED group.

R_{i7} [page 49] is related to the complexity of the use case diagram in terms of the size of the subsystem. R_{i7} is violated for all three subsystem of a use case diagram in first iteration and second iteration. In the first iteration, students defined too many use cases for each subsystem. They tried to reduce the size to a maximum of five use cases, but still they had violations in the second iterations.

R_{i8} [page 50], R_{i22} [page 51], R_{i26} [page 51] and R_{i29} [page 51] are related to the compliance with the UML naming convention described in Chapter 4, whereby R_{i8} is related to the subsystem name. R_{i22} is related to the class name, and R_{i26} is related to the associations without a name. The RED group has violated R_{i8} two times in first iteration, and it resolved the problem in second iteration. R_{i22} was violated six times in first iteration and resolved in second iteration. R_{i26} and R_{i29} are violated only once in first iteration, and the problem is resolved in second iteration.

R_{i27} [page 48], is violated one single time, but this is resolved in the second iteration. R_{i27} is related to the presence of multiplicity values at each association.

A sample use case diagram is illustrated in Figure 6.4. The diagram contains the violations of R_{i7} [page 49] and R_{i8} [page 50]. R_{i7} violation is related to the size of

R.No	Name of the violated rule	No. of times the rule was violated (Iteration 1)	No. of times the rule was violated (Iteration 2)
R_{i7}	Each subsystem should contain a minimum of 3 and a maximum of 5 use cases	3	3
R_{i8}	A subsystem name should start with a capital letter and should be consisting of one to two words	2	0
R_{i22}	Each class name should start with a capital letter and should be one word	6	0
R_{i26}	Each association must have a name	1	0
R_{i27}	Each association must specify multiplicity values at both ends	1	0
R_{i29}	Each association name should start with a lower case letter	1	0

Table 6.4: RED Group's Violations of Rules for Understandability of Incomplete Model Type

the subsystem, which is more than five use cases for the subsystem. R_{i8} is related to the naming convention, in which the name of the subsystem does not start with a capital letter.

There were no violations found in their sequence diagrams, whereas in their class diagrams some violations were found. These are illustrated in the Figure 6.5. The RED group has violated almost the same rules or guidelines, which were violated by the BLUE group. The violations of R_{i22} are related to the naming convention for a class. R_{i26} is related to an association that has no name and R_{i29} is related to the association name starting with a lower case letter. An association that does not contain any value at both ends of the association is detected according to R_{i27}.

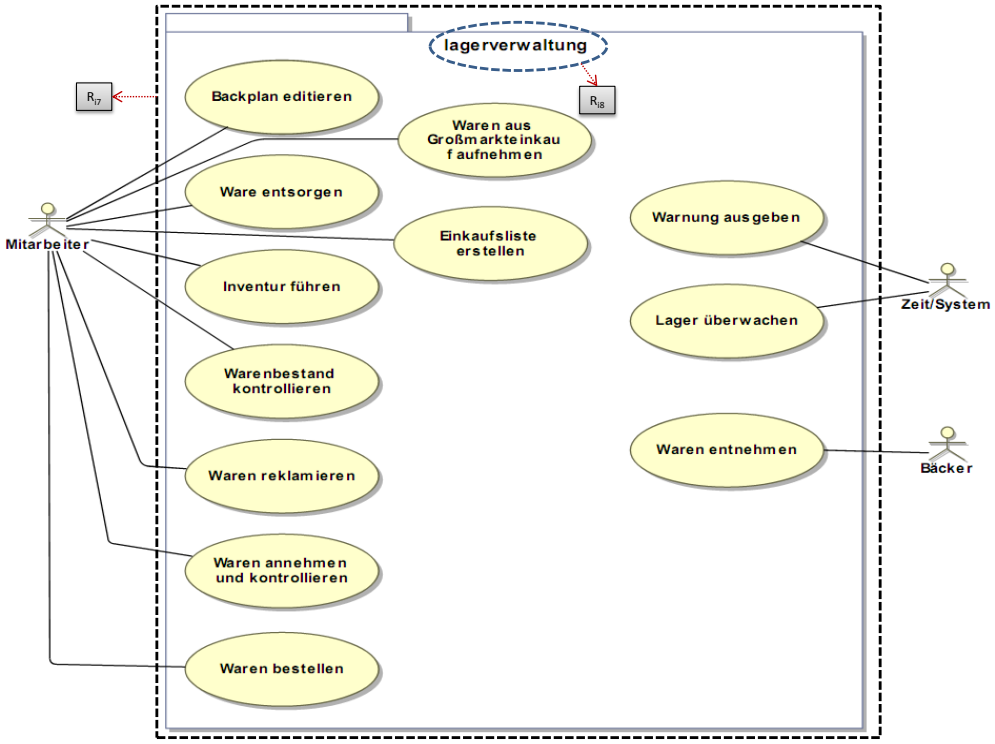


Figure 6.4: RED Group's Use Case Diagram of Incomplete Model

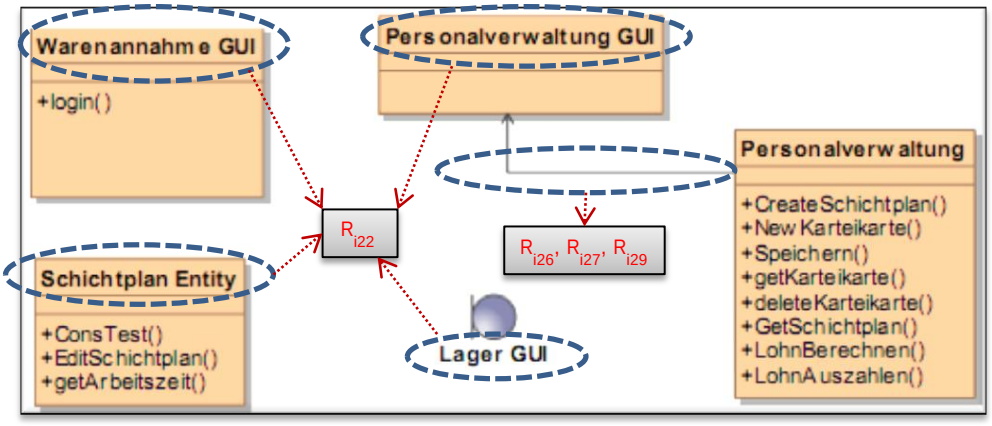


Figure 6.5: RED Group's Class Diagram of Incomplete Model

6.2.2 Quality Assessment Results for Complete Model

The design model of the bakery system is considered to be a *complete model* in our proposed quality model described in Chapter 4. The quality assessment is performed for *analyzability*, *changeability* and *understandability* quality attributes. Both groups have not violated any rule or guideline related to the *changeability* quality attribute, therefore the following paragraph describes the details of the quality assessment results for *analyzability*, and *understandability* quality attributes.

6.2.2.1 BLUE Group Complete Model

Table 6.5 illustrates the *analyzability* related violations and Table 6.6 depicts the *understandability* related violations for the *complete* model.

6.2.2.1.1 Analyzability for Complete Model

R_{c24} [page 51] is related to the traceability problem between a class diagram and sequence diagrams. This rule was violated in both iterations (Table 6.5). In the second iteration, they still had 15 violations of this rule. The rule was violated because the BLUE group did not properly merge their decomposed models into a single project. Hence, their sequence diagrams lacked the information from class diagram modules. The lifeline in the sequence diagrams is present but their corresponding classes in the class diagram are either changed or deleted. R_{c24} suggests that in the sequence diagram, each lifeline must have a class in the class diagram.

R_{c25} [page 51] is also a traceability problem. The messages in the sequence diagram do not represent the actual methods in their corresponding classes of a class diagram. R_{c25} requires that every message, which is represented by a method *must* be traceable to their corresponding operation in a class. In the first iteration, they violated this rule 37 times, which is a high value. In the second iteration, they still had ten violations of this rule. As described earlier for R_{c24} this happens, when smaller modules of the decomposed project are combined to form a single project. During this process, BLUE group's model produced inconsistencies between class and sequence diagrams. Therefore, the message, which require methods are not traceable to their corresponding classes in a class diagram.

6.2.2.1.2 Understandability for Complete Model

R_{c1} [page 53] and R_{c2} [page 53] violations are related to a class, which does not have any attribute or an operation respectively. In the first iteration ten violations were detected for R_{c1}, and 12 violations for R_{c2}. In the second iteration,

R.No	Name of the violated rule	No. of times the rule was violated (Iteration 1)	No. of times the rule was violated (Iteration 2)
R_{c24}	Each object or lifeline in a sequence diagram must have a corresponding class in a class diagram	33	15
R_{c25}	Every call message received by a lifeline must have a corresponding operation in the class	37	10

Table 6.5: BLUE Group's Violations of Rules for Analyzability of Complete Model Type

there were still two violations for R_{c1} and one for R_{c2} detected. In first iteration, students did not focus on the class attributes and operations and defined only classes. After refining the class diagrams, they did not delete the corresponding classes from the model containment tree, and our quality assessment tool detected the violations for R_{c1} and R_{c2}.

R_{c6} [page 54] is a guideline, that has been violated ten times in both iterations. R_{c6} suggests a maximum number of operations for a single class. This happened because BLUE group introduced new getter and setter methods for the classes.

R_{c16} [page 54] checks two aspects of an attribute, one is the data type of the attribute, and the other is the visibility of an attribute. It is detected five times in the first iteration and three times in the second iteration. Attributes without data types and visibility are hard to understand. The problem may become severe, when code is generated automatically or manually from the diagrams. R_{c16} is violated because students did not follow the tool to add data types for the attributes. The Magic draw UML tool either produced empty space for the data type or the data types were not standard data types of any object oriented programming language.

R_{c18} [page 54] is related to the number of parameters of an operation. The threshold value was four. R_{c18} was violated in the first iteration five times. However, in second iteration they managed to resolve this violation. Hence our quality assessment tool helped to improve the quality of the model.

R_{c21} [page 54] was violated 142 times in first iteration. The BLUE group defined methods without providing return types. In the first iteration, they had elements existing in the model containment tree but not in the diagram pane. Our quality assessment tool detected the violations of R_{c21} for those elements.

R_{c22} [page 54] is related to the data type of a parameter passed to the operation. This was violated 10 times in their first iteration but was resolved in the second iteration.

R_{c31} [page 53] is related to the number of entry and exit points in an activity diagram. R_{c31} restricts them to a single entry and single exit node. R_{c31} was violated in two activity diagrams. However, in the second iteration all R_{c31} violations were removed.

R_{c33} [page 54] is related to activities in an activity diagram that do not have any incoming or outgoing transitions. These activities have no use in the activity diagram and are considered to be dead activities, therefore should be removed. Only two dead activities are identified in their first iteration and in second iteration all dead elements were removed.

R.No	Name of the violated rule	No. of times the rule was violated (Iteration 1)	No. of times the rule was violated (Iteration 2)
R_{c1}	Every class should have attributes	10	2
R_{c2}	Every class should have operations	12	1
R_{c6}	Each class should have a maximum of 10 operations	10	10
R_{c16}	Each attribute must have a data type and must be private	5	3
R_{c18}	Each operation should have a maximum of four parameters	5	0
R_{c21}	Each operation must have a return type	142	0
R_{c22}	Each parameter must have a data type	10	0
R_{c31}	Each activity diagram should contain one initial node and one exit point	2	0
R_{c33}	Dead activities must not exist in an activity diagram	2	0

Table 6.6: BLUE Group's Violations of Rules for Understandability of Complete Model Type

Some of the violations listed in Table 6.5 and Table 6.6 are exemplified in Figure 6.6 for violations in class diagrams, in Figure 6.7 for violations in a sequence diagram, and in Figure 6.8 for violations in an activity diagram.

In Figure 6.6 two smelly classes are shown, where R_{c1}, R_{c6}, R_{c18} and R_{c21} are the violations for class *Backverwaltung* and R_{c2} and R_{c16} are the violations detected for the *Backwaremenge*.

R_{c1} [page 53] and R_{c2} [page 53] are violated in these classes. The class attributes are missing for the *Backverwaltung* class and operations are missing for the *Backwaremenge* class.

R_{c6} violation is related to the size of the class as shown in Figure 6.6, in which a maximum of ten operations were allowed, the class *Backverwaltung* contains more than 10 operations.

R_{c16} [page 54] is a critical violation, where attributes do not have any data type. Furthermore, a R_{c16} violation is highlighted in Figure 6.6 for class *Backwaremenge*, where data types are not given for the class attributes.

R_{c21} [page 54] is related to a class operation, that does not have a return type. Class *Backverwaltung* in Figure 6.6 shows the violations of R_{c21} and R_{c18} [page 54]. R_{c18} is highlighted for operation *setBackwarenData* that contains more than four parameters.

Figure 6.7 shows a sequence diagram, where most critical violations are highlighted for the *analyzability* quality attribute. R_{c24} [page 51] is the critical violation, i.e., related to the traceability between a class and sequence diagrams. R_{c25} [page 51] is related to the traceability between methods used in a sequence diagrams and the corresponding operation in a class. Figure 6.7 shows that the methods are defined on the messages but their names and parameters are not described, and the class in a class diagram does not contain that operation. The reason is that the BLUE group did not know, how to call the constructor, and they therefore made violations of R_{c24}.

Figure 6.8 shows the violations in an activity diagram in the *complete model*. R_{c31} [page 53] and R_{c33} [page 54] violations are highlighted in the activity diagram. R_{c31} should have exactly one entry and one exit node in the activity diagram. This violation is visible in the diagram, whereby two exit nodes are detected within the activity diagram.

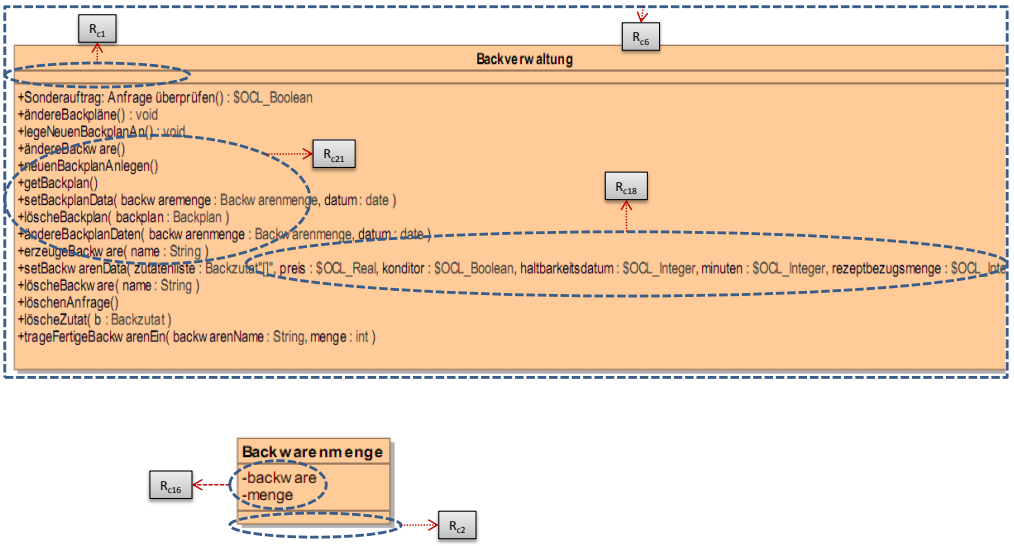


Figure 6.6: BLUE Group's Extracted Classes of Complete Model Type

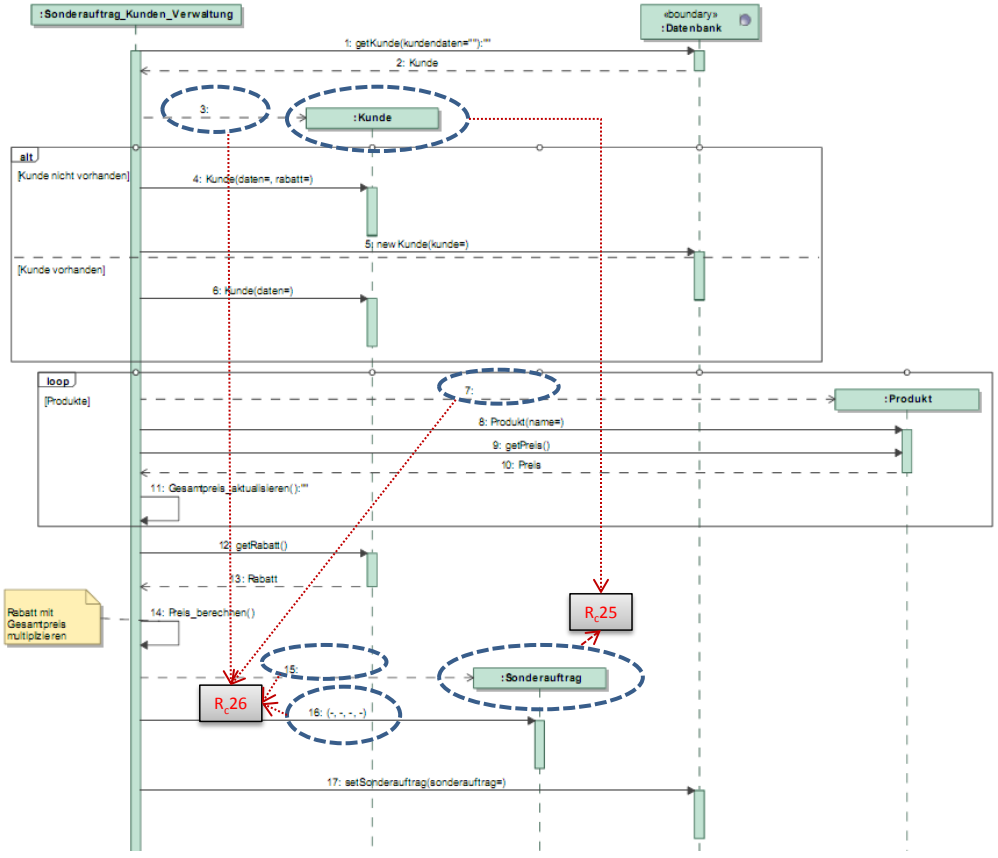


Figure 6.7: BLUE Group's Sequence Diagram of Complete Model Type

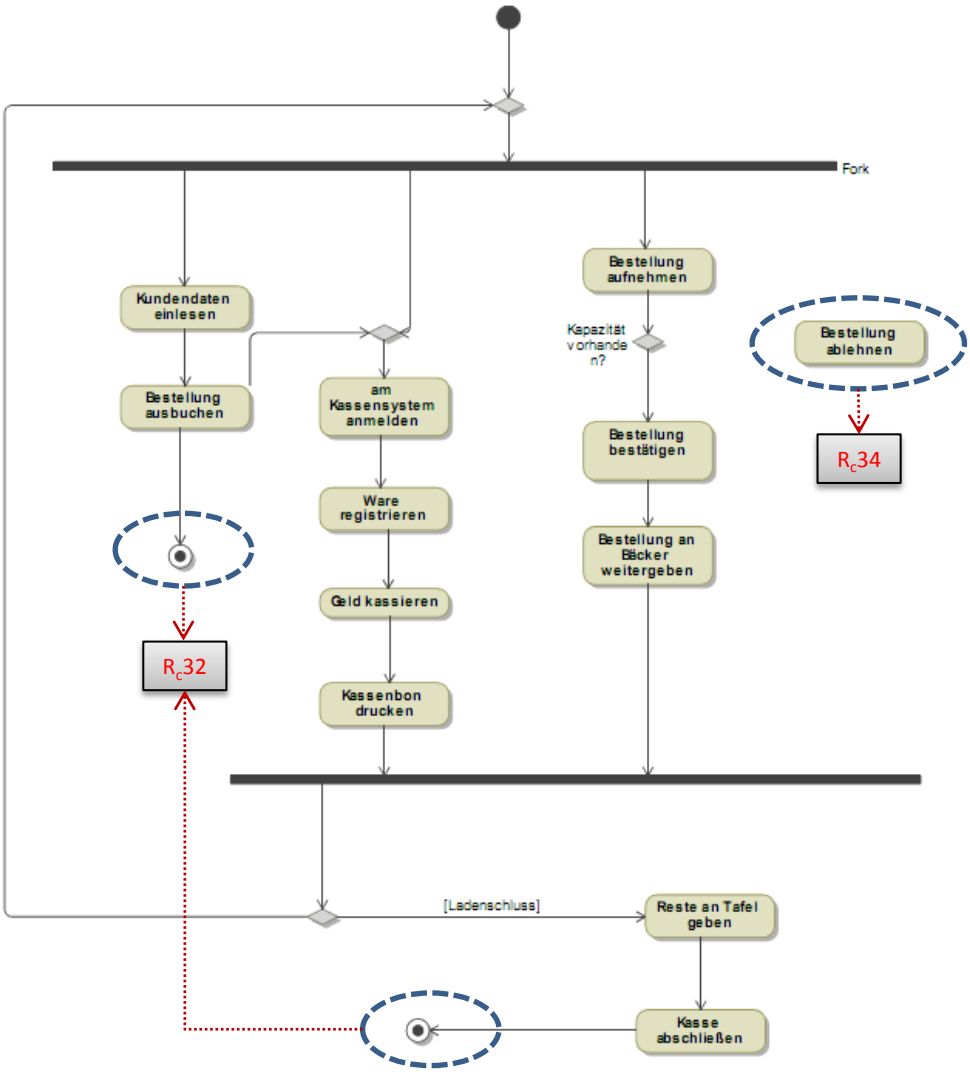


Figure 6.8: BLUE Group's Activity Diagram of Complete Model Type

6.2.2.2 RED Group Complete Model

The results of the violation of the rules by the RED group are shown in Table 6.7 for *analyzability* and Table 6.8 for *understandability* quality attributes respectively.

6.2.2.2.1 Analyzability for Complete Model

R_{c25} [page 51] is a traceability violation between a class diagram and a sequence diagram, in which messages that represent the method can be traced back to their corresponding classes. This rule was violated 37 times in the first iteration and ten times in the second iteration (Table 6.7). The reason is the same as described in the violations of R_{c25} for the BLUE group [page 80], in which a decomposed module was merged to form a single project, thus introducing inconsistencies between a class diagram and a sequence diagram.

R.No	Name of the violated rule	No. of times the rule was violated (Iteration 1)	No. of times the rule was violated (Iteration 2)
R_{c25}	Every call message received by a lifeline should have a corresponding operation in the class	37	10

Table 6.7: RED Group's Violations of Rules for Analyzability of Complete Model Type

6.2.2.2.2 Understandability for Complete Model

The value of R_{c1} [page 53] and R_{c2} [page 53] violations are high in their first and second iteration (Table 6.8). Both groups did not take care, when creating a new classes or deleting any empty class from their models. They deleted classes from a diagram pane but not from the model containment tree and our tool detected it as a violation of R_{c1} and R_{c2}.

R_{c5} [page 55] is related to a naming convention in that class names should start with a capital letter. This was violated by the RED group more than in their first iteration. The value increased in the second iteration because the RED group deleted or modified about more than 20 classes. When they modified the classes, they did not respect R_{c5}, and our prototype tool detected violations of R_{c5}.

R_{c6} [page 54] and R_{c13} [page 54] are related to the number of operations for a single class and the number of classes for a single package respectively. R_{c6} violations increased by one and become seven in their second iteration. They improved the model with respect to the violations of R_{c13}.

R_{c17} [page 55] is related to the composition relationship, in which multiplicity values for complete models were recommended on the side of the owner. R_{c17} was violated once in the first iteration and in the second iteration the problem was resolved.

R_{c18} [page 54] is related to the number of parameters, in which a maximum of four parameters per operation is defined by R_{c18}. This was violated by two times in the first iteration, but still the same violations were present in the model in the second iteration. This happened because students did not take care of R_{c18} and hence had the same violations in their second iteration.

R_{c19} [page 55] is related to getter and setter methods for the <<entity>> classes. This was violated 12 times in the first iteration and still two violations were present in the second iteration. This is because new <<entity>> classes were added to their class diagrams without getter and setter methods.

R_{c21} [page 54] was violated 25 times in the first iteration and 23 of them were removed in the second iteration leaving two still in existence. The number of operations was increased from 174 to 277 in their second iteration as a part of the introduction of the new violations of R_{c21}.

R_{c22} [page 54] was violated 42 times in their first iteration, which is a quite high value, and in the second iteration, they still had five violations. This is due to the increased number of parameters per operations in the second iteration. In addition, the RED group also had some dead parameters introduced when they merged their decomposed model into one single project.

Figure 6.9 shows some of the violations and these are R_{c1} [page 53], R_{c17} [page 55], R_{c18} [page 54] and R_{c21} [page 54] as highlighted in the Figure 6.9.

R_{c13} is related to size of the package, in which R_{c13} allows 20 classes per package, However, this is not shown in Figure 6.9 due to limited space.

R_{c17} is related to a violation in which the multiplicity of composition association type should be one at the side of the owner. One of the sample violations of the R_{c17} is shown in Figure 6.9.

R_{c18} is related to the size, in which a maximum of four parameters per operation is recommended. The operation *KundendatenSpeichern* has more than four parameters as shown in Figure 6.9.

R.No	Name of the violated rule	No. of times the rule was violated (Iteration 1)	No. of times the rule was violated (Iteration 2)
R_{c1}	Every class should have attributes	32	21
R_{c2}	Every class should have operations	30	25
R_{c5}	Each class should start with a capital letter and should be one word	8	15
R_{c6}	Each class should have a maximum of 10 operations	6	7
R_{c13}	Each package should contain a maximum of 20 classes	1	0
R_{c17}	If class has a composition relationship then multiplicity should be one at the side of owner	1	0
R_{c18}	Each operation should have a maximum four parameters	2	2
R_{c19}	An <<entity>> class should have getters and setters	12	2
R_{c21}	Each operation must have a return type	25	2
R_{c22}	Each parameter must have a data type	42	5

Table 6.8: RED Group's Violations of Rules for Understandability of Complete Model Type

R_{c21} refers to a missing return type of an operation. The sample R_{c21} violation is highlighted in the class *Verkauf* for the operation *KundenLaden()* as shown in Figure 6.9.

Figure 6.10 highlights some violations in a sequence diagram. R_{c25} [page 51] is related to missing traceability of methods in a sequence diagram to the class operation.

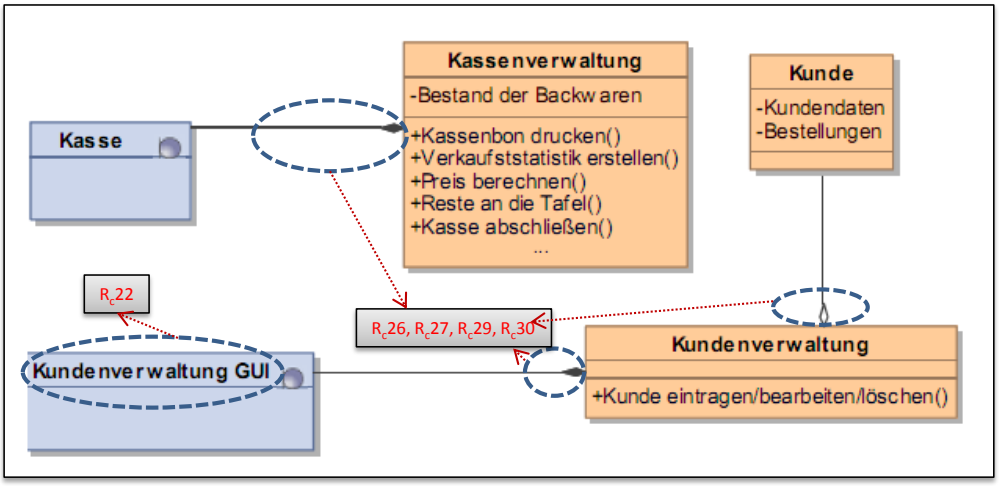


Figure 6.9: RED Group's Class Diagram of Complete Model Type

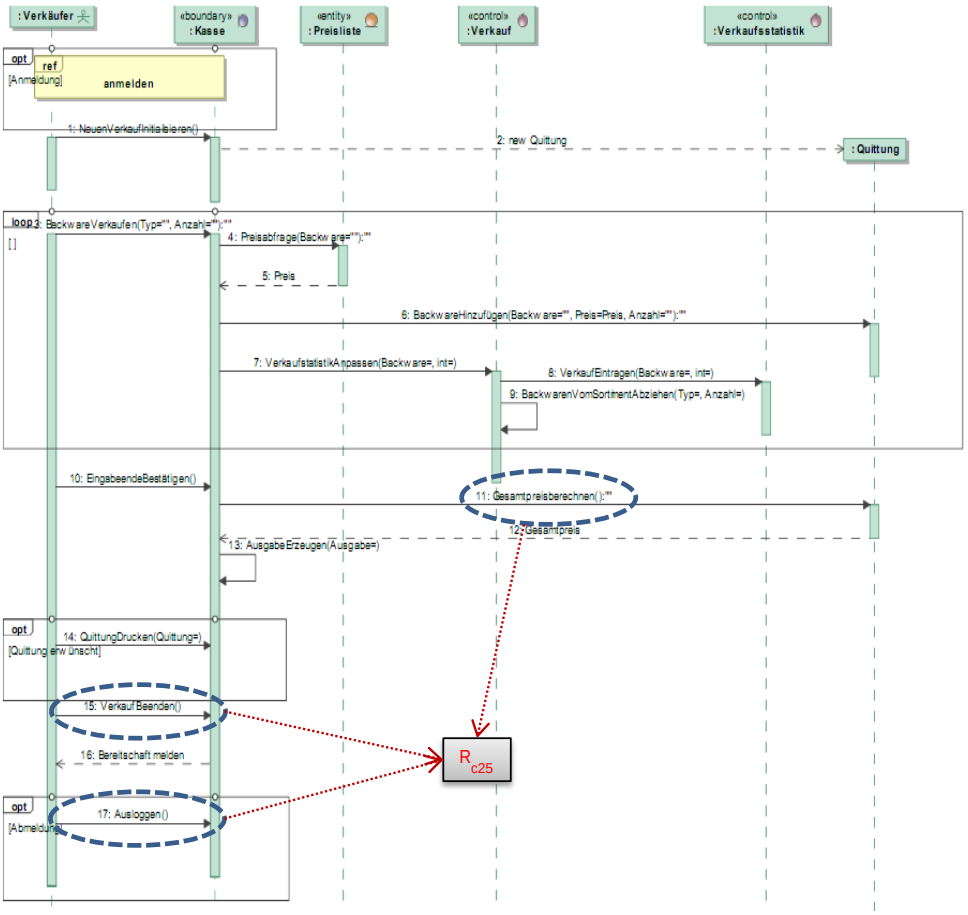


Figure 6.10: RED Group's Sequence Diagram of Complete Model Type

6.3 Size and Ratio Metrics

This section presents size and ratio metrics for the quality assessment of the bakery system. The ratio metrics provides faults per element. The metrics are defined for *analyzability* and *understandability* quality attributes. Each metric is defined for the violated rule or the guideline described in the previous Section 6.2 for *incomplete* and *complete* models.

6.3.1 Size Metrics for Incomplete Model

Before discussing the ratio metrics, let us look into the size of the *incomplete model* developed by the BLUE and RED groups. Table 6.9 illustrates the size of the *incomplete model* before refactoring (i.e., iteration 1) and after refactoring (i.e., iteration 2) of the models. From the results, the size of the model varies in the second iteration i.e., after refactoring the model. For example, RED group has 40 diagrams before refactoring the model and 30 after refactoring the model. This is because they had removed duplicate elements in the model. Likewise, the total number of activity diagrams were 12 in the first iteration and in the second iteration they reduced to three.

	Group	Iteration 1	Iteration 2
Total Number of Diagrams	BLUE	35	35
	RED	40	30
Total Number of Use Cases	BLUE	50	42
	RED	24	25
Total Number of Actors	BLUE	11	11
	RED	6	6
Total Number of Subsystem	BLUE	3	3
	RED	3	3
Total Number of Classes	BLUE	31	31
	RED	33	30
Total Number of Associations	BLUE	52	36
	RED	33	33
Total Number of Operations	BLUE	35	37
	RED	59	61
Total Number of Sequence Diagrams	BLUE	30	30
	RED	13	13
Total Number of Messages	BLUE	230	221
	RED	223	223
Total Number of Activity Diagrams	BLUE	3	3
	RED	12	3
Total Number of Activity States in Activity Diagram	BLUE	31	32
	RED	45	20

Table 6.9: Size Metrics for Incomplete Model Type of BLUE and RED Group

6.3.1.1 Ratio Metrics for Analyzability of the Incomplete Model

Table 6.10 show absolute and ratio metrics for the *analyzability* quality attribute for BLUE and RED group. The metric value zero indicates that there are no violations for the corresponding rule and metric value 1.0 indicates that all available elements in the model have violated the corresponding rule.

Metric M_{i16} counts the violations of R_{i16} for BLUE group [Table 6.1, page 72] and for RED group [Table 6.3, page 77] in the case of *incomplete models*. The unchanged metric value 1.0 indicates that there was no improvement made by the BLUE group in the second iteration. For RED group, the metric value 0.0 indicates that there was no violation in the first iteration while, the metric value increased in the second iteration, which decreased the quality of the model.

The metric value M_{i17} counts the violations of R_{i17} for BLUE group [Table 6.1, page 72] and for RED group [Table 6.3, page 77] in the case of *incomplete models*. The metric value decreased in the second iteration due to reduction in the number of violations for the BLUE group. This indicates that quality of the model improved in the second iteration. The RED group's model do not show any reasonable variation in their metric value because they still had a quite high value for the violations of R_{i17}.

The metric value M_{i18} counts the violations of R_{i18} for BLUE group [Table 6.1, page 72] and for RED group [Table 6.3, page 77] in the case of *incomplete models*. The metric value is the same for BLUE and RED groups in their first iteration, which shows that they have violated R_{i18} for all three subsystems. In the second iteration both groups improved their models, and hence metric value becomes zero. The metric M_{i18} measures the consistency between a use case diagram and a class diagram.

The metric value M_{i34} counts the violations of R_{i34} for BLUE group [Table 6.1, page 72]. The quality of the model was better in first iteration than in second iteration for BLUE group. In second iteration, the violations increased, which led to an increase in the metric value and a decrease in the quality of the model. RED group did not violate R_{i34} in both iterations, hence the metric value is zero, and the model is stable in both iterations.

Metric = No. of violations of the context element / Total No. of context Element	Group	No. of violations of the context element		Total No. of context element		Metrics value	
		Iteration 1	Iteration 2	Iteration 1	Iteration 2	Iteration 1	Iteration 2
M_i16= No. of subsystems that do not refined by activity diagrams / Total number of subsystems	BLUE	3	3	3	3	1.0	1.0
	RED	0	2	3	3	0.0	0.67
M_i17= No. of activities that do not refer to use case / Total number of activities in Activity diagrams	BLUE	20	1	31	32	0.65	0.03
	RED	31	13	45	20	0.69	0.65
M_i18= No. of subsytems that do not refer to class package / Total number of subsystems	BLUE	3	0	3	3	1.0	0.0
	RED	3	0	3	3	1.0	0.0
M_i34= No. of messages that do not refer to class operations / Total number of messages	BLUE	9	48	230	221	0.04	0.22
	RED	0	0	223	223	0.0	0.0

Table 6.10: Analyzability Ratio Metrics for BLUE and RED Group of Incomplete Model Type

6.3.1.2 Ratio Metrics for Understandability of Incomplete Model

Table 6.11 lists the *understandability* absolute and ratio metrics for BLUE and RED groups.

The metric M_{i1} counts the violations of R_{i1} [Table 6.2, page 73]. The metric value zero in the second iteration of BLUE group indicates that the quality of the model was improved. RED group's model did not violate R_{i1} in neither iterations.

The metric M_{i5} and M_{i6} are only counted for the BLUE group because RED group did not violate R_{i5} and R_{i6}. The metric value for M_{i5} and M_{i6} indicate that the quality of the BLUE group's model improved in the second iteration.

The metric value M_{i7} counts the violations of R_{i7} for BLUE group [Table 6.2, page 73] and for RED group [Table 6.4, page 78] in the case of *incomplete models*. The metric value is same in first and second iteration for both groups. The metric shows that there is no quality improvement in second iteration of the models.

The metric M_{i8} is only counted for the RED group because BLUE group did not violate R_{i8}. The metric counts the violations of R_{i8} for RED group [Table 6.4, page 78]. In first iteration, the RED group model has violations and this decreases the quality of the model. In second iteration, they managed to improve the quality of the model.

The metric M_{i22} counts the violations of R_{i22} for BLUE group [Table 6.2, page 73] and for RED group [Table 6.4, page 78] in the case of *incomplete models*. For BLUE and RED group models in first iteration, the

Metric = No. of violations of the context element / Total No. of context Element	Group	No. of violations of the context element		Total No. of context element		Metrics value	
		Iteration 1	Iteration 2	Iteration 1	Iteration 2	Iteration 1	Iteration 2
M_{i1}= No. of use cases do not placed inside the subsystem / Total no. of use cases	BLUE	1	0	50	42	0.02	0.0
	RED	0	0	24	25	0.0	0.0
M_{i5}= No. of use cases do not communicate more than three actors / Total no. of use cases	BLUE	7	0	50	42	0.14	0.0
	RED	0	0	24	25	0.0	0.0
M_{i6}= No. of use case name contains more than four words/ Total no. of use cases	BLUE	3	0	50	42	0.06	0.0
	RED	0	0	24	25	0.0	0.0
M_{i7}= No. of subsystems that contain more than five use cases / Total no. of subsystems	BLUE	3	3	3	3	1.0	1.0
	RED	3	3	3	3	1.0	1.0
M_{i8}= No. of subsystem name that do not start with a capital letter or have more than two words in a name / Total no. of subsystems	BLUE	0	0	3	3	0.0	0.0
	RED	2	0	3	3	0.67	0.0
M_{i22}= No. of classes that do not start with a capital letter or do not consist of one word / Total no. of classes	BLUE	7	0	31	31	0.23	0.0
	RED	6	0	33	30	0.18	0.0
M_{i26}= No. of associations that do not have a name / Total no. of associations	BLUE	1	0	52	36	0.02	0.0
	RED	1	0	33	33	0.03	0.0
M_{i27}= No. of association name that do not have multiplicity value at both ends / Total no. of associations	BLUE	1	0	52	36	0.02	0.0
	RED	1	0	33	33	0.03	0.0
M_{i29}= No. of assocition name that do not start with lower case letter / Total no. of associations	BLUE	1	0	52	36	0.02	0.0
	RED	1	0	33	33	0.03	0.0
M_{i30}= No. of classes that have composition or aggregation type of association / Total no. of classes	BLUE	17	1	31	31	0.55	0.03
	RED	0	0	33	30	0.0	0.0

Table 6.11: Understandability Ratio Metrics for BLUE and RED Group of Incomplete Model Type

M_{i22} measure indicates that there are some violations existing in the model which decreases the quality of the models. In second iteration, both groups managed to improve the quality of their models and metric value decreases to zero.

R_{i26} is violated once for BLUE and RED groups in the first iteration. The metric value indicates that both groups have increased the quality of the models by decreasing the violations in the second iteration.

R_{i27} and R_{i29} are violated once for both BLUE and RED groups in the first iteration. The metric value for the second iteration shows that the quality was improved by both groups.

The metric M_{i30} exhibits the violations of R_{i30} only for BLUE group [Table 6.2, page 73]. There are no R_{i30} violations detected for the RED group. The metric value slightly decreased in second iteration, showing that the model developed in second iteration has better quality than that one developed in the first iteration.

6.3.2 Size Metrics for Complete Models

The size metric for *complete models* for BLUE and RED groups are shown in Table 6.12. The size metrics for both iteration showed the variation in the models developed before and after refactoring.

	Group	Iteration 1	Iteration 2
Total Number of Diagrams	BLUE	44	44
	RED	47	46
Total Number of Classes	BLUE	51	51
	RED	70	48
Total Number of Attributes	BLUE	97	95
	RED	121	126
Total Number of Associations	BLUE	65	65
	RED	53	52
Total Number of Entitty Classes	BLUE	3	3
	RED	12	14
Total Number of Operations	BLUE	255	220
	RED	177	274
Total Number of Parameters	BLUE	190	190
	RED	61	80
Total number of Packages	BLUE	12	12
	RED	10	10
Total Number of Sequence Diagrams	BLUE	38	38
	RED	26	18
Total Number of Messages	BLUE	548	403
	RED	455	402
Total Number of Lifelines	BLUE	254	180
	RED	220	135
Total Number of Activity Diagrams	BLUE	4	4
	RED	4	13
Total Number of Activity States in Activity Diagrams	BLUE	50	50
	RED	10	58
Total Number of Objects in activity diagrams	BLUE	0	0
	RED	10	6
Total Number of State Machine Diagrams	BLUE	2	2
	RED	6	5
Total Number of States in State Machine Diagrams	BLUE	20	15
	RED	60	60

Table 6.12: Size Metrics for Complete Model Type of BLUE and RED Group

6.3.2.1 Ratio Metrics for Analyzability of Complete Model

Table 6.13 shows the metrics for *analyzability* quality attribute for BLUE and RED group. The metric value zero indicates that there are no violations present in their model, while a metric value 1.0 indicates that all elements present in the model have violated by the corresponding rule.

The metric M_{c24} counts the violations of R_{c24} for BLUE group [Table 6.5, page 81] for the *complete model*. The metric value decreases in the second iteration, which increased the quality of the model for the BLUE group. The RED group did not violate R_{c24} in both iterations, which indicates that RED group's model is more stable in both iterations.

The metric M_{c25} counts the violations of R_{c25} for BLUE group [Table 6.5, page 81] and for RED group [Table 6.7, page 86] for the *complete models*. The metric value decreased in the second iteration for the BLUE group model but still violations exists in the second iteration. However, the quality of the model is better in the second iteration than in the first iteration. The metric value for RED group in the second iteration decreased, and indicates that the quality of the model increased in the second iteration.

Metric = No. of violations of the context element / Total No. of context Element	Group	No. of violations of the context element		Total No. of context element		Metrics value	
		Iteration 1	Iteration 2	Iteration 1	Iteration 2	Iteration 1	Iteration 2
M_{c24}= No. of objects in a sequence diagram do not refer to class in a class diagram / Total no. of objects in a sequence diagram	BLUE	33	15	254	180	0.13	0.08
	RED	0	0	220	135	0.0	0.0
M_{c25}= No. of messages do not refer to a class operation / Total no. of messages	BLUE	37	10	548	403	0.07	0.02
	RED	7	0	455	402	0.02	0.0
M_{c32}= No. of activity diagrams do not refer to a single class operation / Total no. of activity diagrams	BLUE	0	0	4	4	0.0	0.0
	RED	0	0	4	13	0.0	0.0

Table 6.13: Analyzability Ratio Metrics for BLUE and RED Group of Complete Model Type

6.3.2.2 Ratio Metric for Understandability of Complete Model

Table 6.14 shows the ratio metrics for *understandability* quality attribute for the *complete model* of BLUE and RED group.

The metric M_{c1} counts the violations of R_{c1} of the *complete model* for BLUE group [Table 6.6, page 82] and for RED group [Table 6.8, page 88]. The metric M_{c1} measure is related to the class which does not contain any attribute. The metric value for both groups is slightly decreased in second iteration due to the presence of violations in second iteration. This decrease in the value of the metric shows some improvement in second iteration.

The metric M_{c2} counts the violations of R_{c2} in the case of the complete model for the BLUE group [Table 6.6, page 82] and for the RED group [Table 6.8, page 88]. The metric M_{c2} measure is related to the classes which do not have an operation or classes defined without any operation. The metric value for the BLUE group in the second iteration decreased, which shows that the quality improved for the model. The metric value for RED group increased in the second iteration showing that the quality of the model has decreased.

M_{c5}, M_{c6} and M_{c13} are only violated for RED group [Table 6.8, page 88]. M_{c5} is the metric to count violations for R_{c5}, M_{c6} counts violations for R_{c6} and M_{c13} counts violations for R_{c13}. The M_{c5} and M_{c6} values increased, thus decreasing the quality of the RED group's model in the second iteration. M_{c13} value decreased in the second iteration indicating that the quality of the RED group's model have increased.

M_{c16} is only violated by the BLUE group [Table 6.6, page 82] and the metric value in the second iteration decreased, which indicates that the quality of the model has increased.

The metric M_{c17} shows the violation of R_{c17} [Table 6.8, page 88] for RED group and there are no violations for BLUE group of R_{c17}. The metric values are 0.01 and zero in first and second iteration respectively. The metric M_{c17} is related to the multiplicity of values at the ends of associations.

The metric M_{c18} counts the violations of R_{c18} for BLUE group [Table 6.6, page 82] and for RED group [Table 6.8, page 88]. The metric value in second iteration decreased showing that the quality of the model has improved.

The metric M_{c19} reveals the violation of R_{c19} for RED group [Table 6.8, page 88] and there are no R_{c19} violations detected for BLUE group. In first iteration, the

Metric = No. of violations of the context element / Total No. of context Element	Group	No. of violations of the context element		Total No. of context element		Metrics value	
		Iteration 1	Iteration 2	Iteration 1	Iteration 2	Iteration 1	Iteration 2
M_{C1}= No. of classes that do not have an attribute / Total no. of classes in class diagram	BLUE	10	2	51	51	0.20	0.04
	RED	32	21	70	48	0.46	0.44
M_{C2}= No. of classes that do not have an operation / Total no. of classes	BLUE	12	1	51	51	0.24	0.02
	RED	30	25	70	48	0.43	0.52
M_{C5}= No. of classes that do not start with capital letter or not have one single word in a name / Total no. of classes in a class diagram	BLUE	0	0	51	51	0.0	0.0
	RED	8	15	70	48	0.11	0.31
M_{C6}= No. of classes that contain more than 10 operations / Total no. of classes in a class diagram	BLUE	0	0	51	51	0.0	0.0
	RED	6	7	70	48	0.09	0.15
M_{C13}= No. of packages that contains more than 20 classes / Total no. of packages	BLUE	0	0	12	12	0.0	0.0
	RED	1	0	10	10	0.10	0.0
M_{C16}= No. of attributes that do not have a data type / Total no. of attributes in a class diagram	BLUE	5	3	97	95	0.05	0.03
	RED	0	0	121	126	0.0	0.0
M_{C17}= No. of classes that do not have composition relationship and multiplicity is not one at owners end / Total no. of classes in a class diagram	BLUE	0	0	51	51	0.0	0.0
	RED	1	0	70	48	0.01	0.0
M_{C18} = No. of operations that contains more than 4 parameters / Total no. of operations	BLUE	5	0	255	220	0.02	0.0
	RED	2	2	177	274	0.01	0.01
M_{C19} = No. of entity classes that do not have getters or setters / Total no. of entity classes in a class diagram	BLUE	0	0	3	3	0.0	0.0
	RED	12	2	12	14	1.0	0.14
M_{C21} = No. of operations that do not have return type / Total no. of operations	BLUE	142	0	255	220	0.56	0.0
	RED	25	2	177	274	0.14	0.01
M_{C22} = No. of parameters that do not have data type / Total no. of parameters	BLUE	10	0	190	190	0.05	0.0
	RED	42	5	61	80	0.69	0.06
M_{C31}= No of activity diagrams that have more than one intitial or exit nodes / Total no. of activity diagrams	BLUE	0	0	4	4	0.0	0.0
	RED	2	0	4	13	0.50	0.0
M_{C33}= No. of dead activities in an activity diagram / Total no. of activities in an activity diagram	BLUE	2	0	50	50	0.04	0.0
	RED	0	0	10	69	0.0	0.0

Table 6.14: Understandability Ratio Metrics for BLUE and RED Group of Complete Model Type

metric value indicates that all elements have violated R_{c19} and in the second iteration the metric value slightly decreased. Hence, we can say that the quality of the model, developed in the second iteration was better than that of the model developed in first iteration.

M_{c22} counts the violations of R_{c22} for BLUE group [Table 6.6, page 82] and for RED group [Table 6.8, page 88]. The metric value for the BLUE group in the second iteration shows that the quality of the model increased and there are no violations existing. The metric value for RED group in the second iteration decreased to 0.06, and that indicates that violations still exist in the model. For RED group's model, we can say that the model developed in second iteration was better than the one developed in first iteration.

The metric M_{c31} counts the violations of R_{c31} for RED group [Table 6.8, page 88]. There are no R_{c31} violations detected for the BLUE group. The metric M_{c31} measures the presence of more than one initial and exit nodes in an activity diagrams. The metric values decrease to zero in second iteration indicating that the quality of the model has improved.

M_{c33} detects the dead activities in the activity diagram. This metric count the violations of R_{c33} for the BLUE group [Table 6.6, page 82]. There are no R_{c33} violations detected for the RED group. The metric value zero in the second iteration of the BLUE group indicating that the quality of the model has improved.

6.4 Student Feedback and Problems Faced by the Students

The students of the UML course was asked to provide their feedback about the approach. Three types of questions were asked related to our quality assurance approach as described in Table 6.15.

Question No	Question	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Q1	Was the quality assurance approach helpful to you?	0	0	1	0	1	3	1	3	4	0	1
Q2	Was it easy for you to understand the quality assessment results?	0	0	0	0	0	4	1	3	2	4	0
Q3	Was it easy for you to understand the quality criteria ?	0	0	0	1	2	2	1	4	1	2	1

Table 6.15: Student Feedback

The total number of students who participated in the UML practical course was 14. They all provided their feedback for the course. Each response to the question is evaluated in the scale ranging from [0% to 100%], where 0% indicates that the student was not satisfied at all and 100% indicates that the student was fully satisfied with our approach.

Question 1 was asked to provide feedback on how much the quality assurance approach was helpful to them. Only two students had a response less than 50% and 12 students responded between 50% to 100%, which shows that our quality assurance approach was helpful to them.

Question 2 was related to the understanding of the quality assessment results. For this all 14 students responded between 50% to 100%, that is there was no response less than 50%. This shows that students have no difficulty in understanding the quality assessment results.

Question 3 required the student to provide feedback about the understandability of the selected quality criteria. Only three students responded to less than 50% and 11 students responded between 50% to 100%, which shows that the selected quality attributes were comprehensive enough for them. The students were more interested in having more quality criteria to be considered for the quality assessment of their models. Due to time limitations, we only selected three types of quality attributes, which were simple and easy to understand for the students. They violated rules related to *analyzability* and *understandability* quality attributes and there were no violations detected for the *changeability* quality attribute. This is why no violations are reported in the quality assessment results described in Chapter 6.

The UML models were developed using the Magic Draw [63] UML tool. During the development of the models, some tool related problems were faced by the students.

In the first week, both groups (i.e., RED and BLUE) had no experience with the UML modeling language and UML tool. There were lots of the inconsistencies detected in their first iteration of the *incomplete model*. In the second week, they were more confident with the tool. They violated fewer rules than in the first week. In the following paragraph, more details of the problems faced by the students are discussed.

First, the students had difficulties with modeling the behavior of the elements, for example, how to create a behavioral state machine for a class or how to create an activity diagram for an operation. With the help of the instructors, they managed to handle these types of problems in the second iterations.

The second major problem identified in their projects is the orphan proxy problem. This was due to the decomposition of large projects into smaller projects, which helped in organization of the large groups into smaller groups. The ”orphaned proxy” is an indication of a dangling reference. Appearance of the proxy is an

indication that some other elements, for example, from outside the module (i.e., elements in the main project or other modules) reference to the element in the module that was previously there but no longer exists. That means the element was deleted, removed or somehow made unavailable in the module. In such cases MagicDraw creates so called ”orphan proxy” in place of the missing element - a surrogate or non real element (with an exclamation mark - ”!”) in the place where the real element once existed [63].

6.5 Concluding Remarks

A bakery system has been selected as an example used in the UML practical course, and it was also used for the evaluation of our proposed approach. There were two reasons behind the choice of the bakery system as a problem statement for the UML course. First, students can understand the problem statement quite easily. Secondly, the same example has been used in the UML course for two years now, and that gives us more confidence in defining appropriate rules and guidelines for the prototypical instantiation of our proposed quality model.

The findings concerning the bakery system models for both groups showed that students introduced new issues into the models while refactoring their models. This was highly evident in their *incomplete models*.

The BLUE group's *incomplete model* results show that their model is not very stable and has many issues in their first and second iterations. However, their *complete model* is more stable in the second iteration of the model. This shows an increased in the competence while working with the tool, as they were developing the design models.

The RED group's *incomplete model* results show that their model is more stable in the first and second iterations. However, their *complete model* is not stable and has many quality issues in the second iteration. This was because each group was further sub-divided into smaller groups and some of the students in the smaller group introduced some big issues into their design models, thus decreasing the quality of their *complete model*.

From the quality assessment results, we can conclude that all of the issues detected were either *analyzability* or *understandability* problems and there was no violation detected for *changeability*, which was the third quality attribute used for the prototypical instantiation of the quality model described in Chapter 4, Section 4.5.2 for the *incomplete model* and Section 4.5.5 for the *complete model*.

From the implementation point of view, the execution of the Xtend language is slow, which was already discussed by Schubert in his Masters thesis [81], where he compared different M2M transformation languages. The Xtend execution becomes even slower, when we use OCL in the Xtend language.

The case study was performed successfully, and the student's feedback was encouraging. They were more interested in having more quality criteria being considered for quality assessment.

Chapter 7

Conclusion

In this chapter, we summarize our research work and its contributions, and suggest some future research topics to extend and refine the methods presented in this thesis.

7.1 Summary

The main objective of this research was to develop a method for the quality assessment and improvement of UML models. For this purpose, we first defined a quality model for UML models, which is based on an inclusion relationship for three types of the model completeness types, these are: incomplete, complete and executable models. The quality characteristics for each model completeness type are adopted from the generic quality model as defined by the ISO/IEC 9126 quality model.

Our proposed quality model takes into account the different model completeness types used in the software development phase in which a UML model is developed. The purpose of the quality model is to provide a way to the modeler to select appropriate methods for continuous quality assessment and improvement of UML models.

An instantiation of our quality model for a concrete case study was described for three main quality characteristics. These are analyzability, changeability, and understandability, for *incomplete* and *complete* models.

To assess the quality of UML models, we used a GQM based approach to select appropriate rules and guidelines. These rules and guidelines are described in the formal language OCL. The violation of rules or guidelines was considered to be a

smell. Our approach was applied in the UML course offered to B.Sc. and M.Sc. students, in which a bakery system is used as a problem statement. In the UML practical course, students were divided into two groups BLUE and RED in order to get two versions of the same model. The developed models went through our quality assessment prototype tool, and the feedback was provided to the students in the form of quality assessment results. The results were presented in a way that students could easily trace back detected issues to their actual models. Hence, the result contains the name of the violated rule and the location of the problematic element in their models. After getting feedback, the students refactored their models. The refactored model was subjected to the quality assessment tool.

The research focused on the design and an evaluation of the continuous quality assessment and improvement approach for UML models. In this thesis, we have shown that our approach is practically feasible to assess and improve the quality of models in a continuous way.

7.2 Outlook

The proposed quality model for UML is based on our experience, review of existing literature and a series of discussions with experts in software quality and software testing. A possible direction for future research related to the quality model is further validation in the context of Executable UML (xUML) models [54].

Further case studies can be evaluated by considering more quality attributes of the proposed quality model. Additionally, models of different developers could be studied in more detail by considering factors, for example, tooling, expertise, and skill of the developer.

It has been observed that manual refactoring of models introduces new issues in the model. Our two refactorings Rename and Pull up refactorings described in Section 5.2.3.1.1 and Section 5.2.3.1.2 show the application of the automated refactoring of the UML models. The next big step of this research would be to provide an automated tool support for the refactorings of the UML models.

The UML models are visualized graphically with graphical notations of UML elements, which provides partial views of the UML model. These partial views of the diagrams are some time hard to read and understand. The work on the layout of the UML diagram is rarely as described in our literature review in Chapter 3. Hence, the next extension of this research could focus on the layout issues of UML models.

Bibliography

[1] Abstratt Technologies. TextUML Toolkit. http://www.abstratt.com, Last Visited February, 2011. [cited at p. 7]

[2] S. Ambler. *The Elements of UML 2.0 Style*. Cambridge University Press, 2005. [cited at p. 30, 31, 33]

[3] AndroMDA. Code Generator Tool. http://www.andromda.org/, Last Visited February, 2011. [cited at p. 1]

[4] ArgoUML Project. ArgoUML. http://argouml.tigris.org, Last Visited February, 2011. [cited at p. 34]

[5] J. Arlow and I. Neustadt. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley Professional, 2nd edition, 2005. [cited at p. 52]

[6] D. Astels. Refactoring with UML. In *Proceedings of the 3rd International Conference on eXtreme Programming and Flexible Processes in Software Engineering (XP2002)*, 2002. [cited at p. 31, 32]

[7] A. Baroni, S. Braz, and F. B. e Abreu. Using OCL to Formalize Object-Oriented Design Metrics Definitions. In *Proceedings of ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Spain*. Springer, 2002. [cited at p. 29, 35]

[8] V. Basili, G. Caldiera, and H. Rombach. The Goal Question Metric Paradigm. *Encyclopedia of Software Engineering*, 2:528--532, 1994. [cited at p. 45]

[9] V. R. Basili and D. M. Weiss. A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, 10(6):728--738, 1984. [cited at p. 20]

[10] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing - Algorithms for the Visualization of Graphs*. Prentice-Hall, 1998. [cited at p. 33]

[11] B. Berenbach and G. Borotto. Metrics for Model Driven Requirements Development. In *Proceeding of the 28th International Conference on Software Engineering*. ACM Press, 2006. [cited at p. 29]

[12] E. Biermann, C. Ermel, and G. Taentzer. Precise Semantics of EMF Model Transformations by Graph Transformation. In *Model Driven Engineering Languages and Systems*, volume 5301 of *Lecture Notes in Computer Science*. Springer, 2008. [cited at p. 33]

[13] R. Binder. Design for testability in object-oriented systems. *Communication ACM*, 37:87--101, September 1994. [cited at p. 28]

[14] B.Marín, G. Giachetti, O. Pastor, and A. Abran. A Quality Model for Conceptual Models of MDD Environments. *Advances in Software Engineering*, 2010, 2010. [cited at p. 27]

[15] M. Boger, T. Sturm, and P. Fragemann. Refactoring Browser for UML. In *Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, volume 2591 of *Lecture Notes in Computer Science*. Springer, 2003. [cited at p. 34]

[16] Borland. Borland Together. http://www.borland.com/us/products/together, Last Visited February, 2011. [cited at p. 34]

[17] R. Castello, R. Mili, and I. Tollis. Automatic Layout of Statecharts. *Software --- Practice & Experience*, 32:25--55, 2002. [cited at p. 34]

[18] S. R. Chidamber and C. Kemerer. A Metric Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476--493, 1994. [cited at p. 29]

[19] Ł. Dobrzański. UML Model Refactoring- Support for Maintenance of Executable UML Models. Master's thesis, Blekinge Institute of Technology, School of Engineering, Ronneby, Sweden, 2005. [cited at p. 33, 34]

[20] Eclipse. Modeling project. http://http://www.eclipse.org/modeling/, Last Visited February, 2011. [cited at p. 1]

[21] Eclipse. Xpand Model to Model Transformation Project. http://wiki.eclipse.org/Xpand, Last Visited February, 2011. [cited at p. 1, 9, 35]

[22] Eclipse Foundation. Eclipse Model Development Tools (MDT) OCL. http://www.eclipse.org/modeling/mdt/?project=ocl, Last visited February, 2011. [cited at p. 62]

[23] H. Eichelberger and J. W. von Gudenberg. UML Class Diagrams - State of the Art in Layout Techniques. In *Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis, Amsterdam*, 2003. [cited at p. 30, 33]

[24] M. El-Wakil, A. El-Bastawisi, M. B. Riad, and A. A. Fahmy. A Novel Approach to Formalize Object-Oriented Design Metrics. In *Proceedings of the 9th International Conference on Empirical Assessment in Software Engineering*, 2005. [cited at p. 29]

[25] EMFRefactor-Team. The EMF Refactor Component Proposal Eclipse based Project. http://www.eclipse.org/proposals/emf-refactor/, Last Visited February, 2011. [cited at p. 34]

[26] EMPANADA. MetricView Tool. http://www.win.tue.nl/empanada/metricview/, Last Visited February, 2011. [cited at p. 34]

[27] N. Fenton and S. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing, Boston, 1997. [cited at p. 18, 20, 55]

[28] A. Folli and T. Mens. Refactoring of UML models using AGG. In *Proceedings of the 3rd International ERCIM Symposium on Software Evolution*, 2007. [cited at p. 33]

[29] M. Folwer and K. Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language- second Edition*. Addison-Wesley Professional, 1999. [cited at p. 5]

[30] M. Fowler. Refactorings in Alphabetical Order. http://www.refactoring.com/catalog/index.html, Last Visited February, 2011. [cited at p. 21]

[31] M. Fowler. *Refactoring -- Improving the Design of Existing Code*. Addison-Wesley, Boston, 1999. [cited at p. 20, 21, 31]

[32] R. France and J. Bieman. Multi-View Software Evolution --- A UML-based Framework for Evolving Object-Oriented Software. In *Proceedings of 17th IEEE International Conference on Software Maintenance (ICSM 2001)*. IEEE, 2001. [cited at p. 32]

[33] D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons, 2003. [cited at p. 41]

[34] P. Gorp, H. Stenten, T. Mens, and S. Demeyer. Towards Automating Source-Consistent UML Refactorings. In *UML 2003 -- Modeling Languages and Applications*, volume 2863 of *Lecture Notes in Computer Science*. Springer, 2003. [cited at p. 34]

[35] R. Gronback. Model Validation: Applying Audits and Metrics to UML Models, 2004. http://conferences.codegear.com/jp/article/32089, Last Visited February, 2011. [cited at p. 30]

[36] IBM. IBM Rational Systems Developer. http://www.ibm.com/software/awdtools/developer/systemsdeveloper, Last Visited February, 2011. [cited at p. 34]

[37] INRIA. ATLAS Transformation Language. http://www.eclipse.org/m2m/atl, Last Visited February, 2011. [cited at p. 34]

[38] INRIA. The MoDisco Eclipse Project. http://www.eclipse.org/MoDisco/, Last Visited February, 2011. [cited at p. 34]

[39] International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC). ISO/IEC Standard No. 9126. Software Engineering- Product Quality; Part 1-4, 2001-2004. [cited at p. 19, 39]

[40] ItemisAG. The Xpand Eclipse Modeling Project. http://www.eclipse.org/modeling/m2t/?project=xpand, Last Visited February, 2011. [cited at p. 34]

[41] A. Jalbani, J. Grabowski, H. Neukirchen, and B. Zeiß. Towards an Integrated Quality Assessment and Improvement Approach for UML Models. In *14th System Design Languages Forum (SDL Forum 2009), 22-24 Sep 2009, Ruhr-University of Bochum, Germany*, sep 2009. [cited at p. 25]

[42] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez. ATL: A QVT-Like Transformation Language. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*. ACM, 2006. [cited at p. 34]

[43] M. Kiewkanya and P. Muenchaisri. Measuring Maintainability in Early Phase using Aesthetic Metrics. In *Proceedings of the 4th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems*, 2005. [cited at p. 30]

[44] H. Kim and C. Boldyreff. Developing Software Metrics Applicable to UML Models. In *Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Engineering, Malaga, Spain*, 2002. [cited at p. 29]

[45] R. Kollmann and M. Gogolla. Metric-Based Selective Representation of UML Diagrams. *Software Maintenance and Reengineering, European Conference on*, 2002. [cited at p. 29]

[46] P. Kruchten. *The Rational Unified Process: An Introduction.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003. [cited at p. 9]

[47] C. Lange. Improving the Quality of UML Models in Practice. In *Proceedings of 28th International Conference on Software Engineering (ICSE 2006)*. ACM, 2006. [cited at p. 29]

[48] C. Lange. *Assessing and Improving the Quality of Modeling.* PhD thesis, Technische Universiteit Eindhoven, Netherland, 2007. [cited at p. v, 25, 26, 31]

[49] C. Lange and R. Chaudron. Empanada: Empirical analysis of architecture and design quality. `http://www.win.tue.nl/empanada/tools.htm`, Last Visited February, 2011. [cited at p. 34]

[50] C. Lange and R. Chaudron. Managing Model Quality in UML-Based Software Development. In *Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering in Practice (STEP 2005)*. IEEE, 2005. [cited at p. 25]

[51] H. Ma, W. Shao, L.Zhang, Z.Ma, and Y.Jiang. Applying OO Metrics to Assess UML Meta-models. In *Proceedings of MODELS/UML'2004*, 2004. [cited at p. 29]

[52] J. McCall, P. Richards, and G. Walters. Factors in Software Quality. Technical Report RADC TR-77-369, US Rome Air Development Center, 1977. [cited at p. 19]

[53] J. McQuillan and J. Power. A Metamodel for the Measurement of Object-Oriented Systems: An Analysis using Alloy. In *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST 2008)*. IEEE, 2008. [cited at p. 29]

[54] S. Mellor and M. Balcer. *Executable UML: A Foundation for Model-Driven Architecture.* Addison-Wesley, 2002. [cited at p. 33, 41, 104]

[55] ModelwareProject. D2.2 MDD Engineering Metrics Definition. Technical report, Framework Programme Information Society Technologies, 2006. [cited at p. 30]

[56] ModelwareProject. D2.5 MDD Engineering Metrics Baseline. Technical report, Framework Programme Information Society Technologies, 2006. [cited at p. 30]

[57] P. Mohagheghi and J. Aagedal. Evaluating Quality in Model-Driven Engineering. In *Proceedings of the International Workshop on Modeling in Software Engineering (MISE 2007)*. IEEE, 2007. [cited at p. 27]

[58] P. Mohagheghi, V. Dehlen, and T. Neple. Definitions and approaches to model quality in model-based software development - A review of literature. *Information and Software Technology*, 51(12):1646 -- 1669, 2009. [cited at p. 27]

[59] H. Neukirchen and M. Bisanz. Utilising Code Smells to Detect Quality Problems in TTCN-3 Test Suites. In *Proceedings of the 19th IFIP International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2007)*, number 4581 in Lecture Notes in Computer Science (LNCS), 2007. [cited at p. 2]

[60] H. Neukirchen, B. Zeiß, and J. Grabowski. An Approach to Quality Engineering of TTCN-3 Test Specifications. *International Journal on Software Tools for Technology Transfer (STTT), Volume 10, Issue 4. (ISSN 1433-2779) DOI: 10.1007/s10009-008-0075-0*, pages 309--326, Aug. 2008. [cited at p. 2]

[61] H. Neukirchen, B. Zeiss, and J. Grabowski. An Approach to Quality Engineering of TTCN-3 Test Specifications. *International Journal on Software Tools for Technology Transfer (STTT)*, 105(4):309--326, 2008. [cited at p. 2]

[62] J. Nödler, H. Neukirchen, and J. Grabowski. A Flexible Framework for Quality Assurance of Software Artefacts --With Applications to Java, UML, and TTCN-3 Test Specifications. In *2nd International Conference on Software Testing, Verification, and Validation (ICST 2009)*. IEEE, 2009. [cited at p. 2]

[63] NoMagic. Magic Draw UML Tool. http://www.magicdraw.com. [cited at p. 99, 100]

[64] R. Norlund. Integrating the Rational Unified Process with Managing Successful Programmes. http://www.ibm.com/developerworks/rational/library/jun05/norlund/, Last Visited February, 2011. [cited at p. v, 9, 10]

[65] oAW. openArchitectureWare Tool. http://www.openarchitectureware.org, Last Visited February, 2011. [cited at p. 58]

[66] Object Management Group (OMG). *MDA Guide Version 1.0.1*, June 2003. Available online at http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf. Last Visited February, 2011. [cited at p. v, 1, 10, 11]

[67] Object Management Group (OMG). MOF 2.0/XMI Mapping, Version 2.1.1, formal/2007-12-01, 2007. [cited at p. 7]

[68] Object Management Group (OMG). Meta Object Facility (MOF) 2.0 Query/View/-Transformation Specification, formal/08-04-03, 2009. [cited at p. 34]

[69] Object Management Group (OMG). Meta Object Facility (MOF) Core Specification, Version 2.0, formal/2006-01-01, 2009. [cited at p. 6]

[70] Object Management Group (OMG). OCL Core Specification version 2.0, formal/2006-05-01, 2009. [cited at p. 7, 40]

[71] Object Management Group (OMG). UML Infrastructure Specification, Version 2.2, formal/2009-02-04, 2009. [cited at p. 6]

[72] Object Management Group (OMG). UML Superstructure Specification, Version 2.2, formal/2009-02-02, 2009. [cited at p. 1, 5, 6, 45]

[73] OMG. Object Management Group. http://www.omg.org/, Last Visited February, 2011. [cited at p. 7]

[74] OMG. UML 2.3. http://www.omg.org/spec/UML/, Last Visited February, 2011. [cited at p. 5]

[75] OMG. UML Profile for CORBA. http://www.omg.org/technology/documents/profile_catalog.htm, Last Visited February, 2011. [cited at p. 11]

[76] L. Pareto and U. Boquist. A Quality Model for Design Documentation in Model-Centric Projects. In *Proceedings of the 3rd International Workshop on Software Quality Assurance (SOQUA 2006)*. ACM, 2006. [cited at p. 27]

[77] D. Parnas. Software Aging. In *Proceedings of the 16th International Conference on Software Engineering (ICSE), Sorrento, Italy*, pages 279--287. IEEE/ACM, 1994. [cited at p. 21]

[78] I. Porres. Model Refactorings as Rule-Based Update Transformations. In *UML 2003 - The Unified Modeling Language*, volume 2863 of *Lecture Notes in Computer Science*. Springer, 2003. [cited at p. 32]

[79] H. Purchase, J. Allder, and D. Carrington. Graph Layout Aesthetics in UML Diagrams: User Preferences. *Journal of Graph Algorithms and Applications*, 6(3):255--279, 2002. [cited at p. 30, 32, 33]

[80] J. Rech and C. Bunse. *Model-Driven Software Development: Integrating Quality Assurance*. Idea Group Publishing, 2008. [cited at p. 2]

[81] L. Schubert. An Evaluation of Model Transformation Languages for UML Quality Engineering. Master's thesis, Georg-August-Universität Göttingen, 2010. [cited at p. 101]

[82] SDMetrics. The Software Design Metrics tool for the UML. http://www.sdmetrics.com, Last Visited February, 2011. [cited at p. 34]

[83] P. Seuring. Design and Implementation of a UML Model Refactoring Tool. Master's thesis, Hasso-Plattner-Institute for Software Systems Engineering at the Univesity of Potsdam, 2005. [cited at p. 34]

[84] R. Soley. Model-driven Architecture Targets Middleware Interoperability Challenges. http://www.ibm.com/developerworks/rational/library/403.html, Last Visited February, 2011. [cited at p. 10]

[85] T. Stahl, M. Völter, S. Efftinge, and A. Haase. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt, Heidelberg, 2. edition, 2007. [cited at p. 1]

[86] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF -- Eclipse Modeling Framework, Second Edition*. Addison-Wesley, 2009. [cited at p. 1]

[87] G. Sunyé, D. Pollet, Y. Traon, and J. Jézéquel. Refactoring UML Models. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, volume 2185 of *Lecture Notes in Computer Science*. Springer, 2001. [cited at p. 32]

[88] G. Taentzer. A Graph Transformation Environment for Modeling and Validation of Software. In *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lecture Notes in Computer Science*, pages 446--453. Springer, 2004. [cited at p. 33]

[89] B. Unhelkar. *Verification and Validation For Quality Of UML 2.0 Models*. Wiley Interscience, 2005. [cited at p. 9, 14]

[90] H. Voigt, B. Güldali, and G. Engels. Quality Plans for Measuring the Testability of Models. In S. G. I. Schieferdecker, editor, *Proceedings of the 11th International Conference on Quality Engineering in Software Technology (CONQUEST 2008), Potsdam (Germany)*, pages 353 -- 370. dpunkt.verlag, 2008. [cited at p. v, 28]

[91] E. Werner and J. Grabowski. UML-Praktikum. http://www.swe.informatik.uni-goettingen.de/edu/notes/index.php?vorl_nr=70, Last Visited February, 2011. [cited at p. 115]

[92] Wikipedia. Eclipse Modeling Framework. http://en.wikipedia.org/wiki/Eclipse_Modeling_Framework, Last Visited February, 2011. [cited at p. 58]

[93] B. Zeiß. *Quality Assurance of Test Specifications for Reactive Systems*. PhD thesis, Dissertation, Universität Göttingen, Juni 2010 (electronically published on http://webdoc.sub.gwdg.de/diss/2010/zeiss/)., 2010. [cited at p. 2]

Appendices

Appendix A

Description of the Bakery System used in the UML Practical Course

The following description of the bakery system is used with permission of the author (Dr. Edith Werner). The information about the course and related material can be found on the course web page [91].

A.1 Das Bäckerei-System

Die Bäckerei Stegbeck ist ein fränkisches Traditionsunternehmen, das nun im Zuge der Modernisierung ein Softwaresystem erhalten soll.

Neben der Unterstützung der alltäglichen Bäckereiarbeit soll das neue System auch eine bessere Integration an externe Systeme, z.B. eine automatische Warenlieferung, bieten.

Die Bäckerei stellt jeden Tag mehrere Sorten Brot und Brötchen sowie Kuchen und Gebäck für den freien Verkauf her. Da am Wochenende mehr Leute Zeit zum gemütlichen Frühstück und Kaffeetrinken haben, werden am Samstag mehr Brötchen und Gebäck benötigt. Kuchen werden nach Saison angeboten und wechseln regelmäßig. z.B. im Frühsommer Erdbeerkuchen und im Herbst Apfeltaschen. Zusätzlich gibt es spezielle Festgebäcke, die nur zu bestimmten Zeiten angeboten werden (Osterlämmer in der Karwoche, Kniekühle zur Kirchweih, Martinswecken am 11.11., Lebkuchen im Dezember, ...)

A.1.1 Verkauf

Im Verkaufsraum werden die fertigen Gebäcke ausgestellt. Das System soll erfassen, wann und wieviele Waren aus der Backstube in den Verkauf gelangen und wann sie verkauft werden. Zur Optimierung des Angebots soll monatlich eine Verkaufsstatistik erstellt werden.

Am Ende des Tages werden nicht verkaufte Waren der örtlichen Tafel gespendet. Das System muss das entsprechend erfassen. Darüber hinaus nimmt die Bäckerei Sonderaufträge an Z.B. um ein Schulfest mit Brötchen und Brezen zu beliefern oder eine Hochzeitstorte zu backen.

Bei Sonderaufträgen muss die Leistungsfähigkeit der Mitarbeiter beachtet werden, das System soll diese Kapazität automatisch berechnen und anzeigen. Ein Konditor kann nur eine große Hochzeitstorte pro Tag herstellen (mehr kann die Bäckerei auch gar nicht aufbewahren).

Sonderaufträge können auch über ein Online-Portal angefragt werden. In diesem Fall muss ein Mitarbeiter den Sonderauftrag bestätigen und den Kunden benachrichtigen. Für jedes Gebäck ist im System ein Preis gespeichert. Die Kasse kann dann aus der Menge der gekauften Gebäcke und dem Preis den Gesamtpreis berechnen. Selbstverständlich werden auch Kassenbons erstellt!

Da die Kasse sicherheitskritisch ist (Geld, öffentlicher Raum), müssen sich Mitarbeiter bei jedem Verkaufsvorgang mit einer Kennzahl anmelden. So ist es auch möglich, dass mehrere Verkäufer an derselben Kasse unterschiedliche Verkäufe abwickeln.

Bei Sonderaufträgen gibt es für Großkunden und Stammkunden Rabatte. Dazu muss das Verkaufssystem auch die Daten der Kunden erfassen.

A.1.2 Personalverwaltung

Die Personalverwaltung soll Namen und Anschriften der Mitarbeiter erfassen, sowie die jeweiligen Gehälter und die Dauer der Betriebszugehörigkeit. In der Bäckerei arbeiten Verkäufer und Bäcker, zusätzlich gibt es immer mindestens einen Konditor im Betrieb.

Sowohl im Verkauf als auch in der Bäckerei wird ausgebildet. Auszubildende haben an drei Tagen pro Woche Berufsschule und stehen daher nur an den anderen drei Tagen zur Verfügung.

Im ersten und zweiten Lehrjahr benötigen Auszubildende noch viel Betreuung durch erfahrene Mitarbeiter, das muss im Schichtplan berücksichtigt werden. Im dritten Lehrjahr können Auszubildende schon selbständige Tätigkeiten übernehmen und werden daher wie volle Mitarbeiter verplant.

Die wichtigste Funktion der Personalverwaltung ist die Erstellung der Schichtpläne. Schichtpläne werden monatlich jeweils für die Backstube und den Verkaufsraum erstellt: Dabei müssen wechselnde Öffnungszeiten, Urlaubszeiten, Krankentage und Teilzeit berücksichtigt werden, In der Backstube gibt es nur festangestellte Mitarbeiter und Auszubildende, im Verkauf werden zusätzlich auch Aushilfen mit unregelmäßigen Arbeitszeiten beschäftigt, Mitarbeiter können Schichten tauschen, solange die Aufträge dann noch erfüllt werden können.

Die tatsächlichen Arbeitszeiten werden über ein digitales Stechkartensystem erfasst und mit den Schichtplänen abgeglichen. Anhand der geleisteten Arbeitszeiten wird am Ende des Monats der Arbeitslohn berechnet und überwiesen.

A.1.3 Lagerverwaltung

In der Lagerverwaltung werden die Zutaten überwacht. Da in der Bäckerei mit Lebensmitteln gearbeitet wird, muss die Lagerverwaltung für jede Zutat das Mindesthaltbarkeitsdatum erfassen. Auch Wareneingang und das Datum des Verbrauchs werden notiert.

Bei Lebensmitteln, die gekühlt werden müssen (wie Milch, Butter, Eier) muss zusätzlich die Einhaltung der Kühlkette dokumentiert werden (das Lebensmittel darf während Transport und Lagerung 8° Celsius nicht überschreiten).

Die Höchstlagermenge für verderbliche Lebensmittel darf nur überschritten werden, wenn diese kurzfristig für Sonderaufträge benötigt werden. Auch bei Sonderaufträgen darf die Menge der kühlpflichtigen Lebensmittel die Kühlkapazität nicht überschreiten.

Ein Lebensmittel, das sein Mindesthaltbarkeitsdatum überschritten hat oder das nicht korrekt gelagert wurde darf nicht mehr in den Verkauf gelangen und muss entsorgt werden. Da die Entsorgung von Lebensmitteln teuer ist, soll derartige Verschwendung mithilfe des Lagersystems vermieden werden.

Gerade bei frischen Zutaten kann es Schäden geben, außerdem gibt es bei Mehl, Zucker und ähnlichen Trockenzutaten auch Schwund durch ungenaues Wiegen. Daher soll regelmäßig der Lagerbestand geprüft werden. Prüfung:

Langfristig lagernde Lebensmittel: alle 3 Monate, Kühlräume: monatlich, Schnell verderbliche Ware (z.B. frisches Obst): wöchentlich. In allen Fällen wird der Bestand ggf. korrigiert und verdorbene Ware entsorgt.

Zutaten, die täglich benötigt werden (z.B. Mehl, Zucker, Eier, Hefe, Butter) müssen immer in ausreichender Menge vorrätig sein, ansonsten wird automatisch nachbestellt. Saisonale Zutaten, z.B. Erdbeeren im Frühjahr und Zwetschgen im Herbst, werden auf dem Großmarkt eingekauft, aber ebenfalls im System erfasst.

Neben den Standard-Zutaten werden für Sonderaufträge auch außergewöhnliche Zutaten verarbeitet. Bei der Bestellung von Sonderzutaten muss darauf geachtet werden, dass diese rechtzeitig für die Ausführung des Sonderauftrags geliefert werden.

Da die benötigten Zutaten von den zubereiteten Backwaren abhängen, muss das Lagersystem für jedes Gebäck eine Zutatenliste verwalten. Die Bäckerei legt in einem Backplan fest, welche Gebäcke zubereitet werden. Kuchensorten variieren nach Saison, Brot wird täglich zubereitet, Samstags werden mehr Brötchen gebacken.

Der Backplan beruht auf langjähriger Erfahrung und wird im System verwaltet. Ein Mitarbeiter pflegt den Plan, z.B. um saisonales Obstangebot zu berücksichtigen.Sonderaufträge werden kurzfristig erfasst. Bei Engpässen (z.B. Mitarbeiterausfall wegen Krankheit) wird der Plan kurzfristig durch einen Mitarbeiter angepasst.

Auf Basis des Backplans soll das Lagersystem die Standardzutaten automatisch bestellen und außerdem wöchentlich eine Einkaufsliste für den Großmarkt erstellen. Zutaten können bei Lieferung mithilfe eines Scanners erfasst werden oder über ein Formular: Das Lagersystem soll auf kritische Punkte hinweisen (Hinweise zur korrekten Lagerung, Kühlpflicht, Probleme, ...), Sonderzutaten sollen einem Sonderauftrag zugewiesen werden können, damit sie nicht versehentlich anderweitig verwendet werden.

Die Entnahme erfolgt Rezept-gebunden: ein Mitarbeiter ruft ein Rezept auf und entnimmt die entsprechenden Waren aus dem Lager. Zusätzlich soll es auch möglich sein zu experimentieren, so dass beliebige Waren entnommen werden können.

Das System soll dann aber warnen, wenn z.B. Zutaten für einen Sonderauftrag entnommen werden sollen oder die Standardzutaten durch das Experiment die Mindestlagermenge unterschreiten.

Appendix B

Rules and Guidelines

This appendix contains the list of rules for incomplete and complete model types.

B.1 Rules and Guidelines for Incomplete Models

R_{i1}: Each use case must be inside the subsystem.	R_{i2}: Each use case must be associated with an actor.
R_{i3}: The generalization between use cases must not be present in a use case diagram.	R_{i4}: Each use case must be refined in a sequence diagram.
R_{i5}: A use case should not contain more than three actors.	R_{i6}: A use case should contain 1 to 4 words.
R_{i7}: Each subsystem should contain minimum 3 and maximum 5 use cases.	R_{i8}: A subsystem name should start with a capital letter and should be consisting of one to two words.
R_{i9}: Each actor name should start with a capital letter.	R_{i10}: The depth of generalization of an actor should not exceed to one.
R_{i11}: Each system name should start with a capital letter and contain one to two words.	R_{i12}: An Actor must be placed outside the system.
R_{i13}: A use case diagram should not contain more than 20 use cases.	R_{i14}: The depth of include chain of a use case should not exceed to one.
R_{i15}: The depth of extend chain of a use case should not exceed to one.	R_{i16}: Each subsystem should be refined by one activity diagram.
R_{i17}: Each Activity in activity diagram should refers to a use case in a use case diagram.	R_{i18}: Each subsystem of a use case diagram should be represented as a package in a class diagram.

R_{i19}: Each Package should not contain more than 20 classes.	R_{i20}: The depth of inheritance tree should not exceed to 2.
R_{i21}: Multiple inheritance must not exists.	R_{i22}: Each class name should start with a capital letter and should be one word.
R_{i23}: An <<entity>> class should contain at least 3 attributes.	R_{i24}: A <<control>> class should contain 2-5 Operations.
R_{i25}: If class is empty class than class must be the <<boundary>> class.	R_{i26}: Each association must have name.
R_{i27}: Each association must specify multiplicity values at both ends.	R_{i28}: Each class should have 1-5 association.
R_{i29}: Each association name should start with a lower case letter.	R_{i30}: Class should not be linked with composition or aggregation association type.
R_{i31}: The links to classes belonging to another package must be unidirectional.	R_{i32}: Each Sequence diagram should have at least one actor on a lifeline.
R_{i33}: Each object or lifeline in a sequence diagram must have corresponding class in a class diagram.	R_{i34}: Every call message received by the lifeline must have corresponding operation in the class.
R_{i35}: If there is a message call between two lifeline than there must be an association between corresponding classes.	R_{i36}: Each message must be labeled.

Table B.1: Rules and Guidelines for Incomplete Models

B.2 Rules and Guidelines for Complete Models

R_{c1}: Every class should have attributes.	R_{c2}: Every class should have operations.
R_{c3}: The depth of inheritance level should be less than 4.	R_{c4}: same as R_{i21}.
R_{c5}: same as R_{i22}.	R_{c6}: Each class should have a maximum of 10 operations.
R_{c7}: same as R_{i26}.	R_{c8}: same as R_{i28}.
R_{c9}: same as R_{i29}.	R_{c10}: Each association must have a direction.
R_{c11}: Each association must specify multiplicity and it must be n to 1.	R_{c12}: Association classes must not present in a model.
R_{c13}: Each package should have maximum 20 classes.	R_{c14}: same as R_{i31}.
R_{c15}: The maximum package nesting level should be 2.	R_{c16}: Each attribute must have data type and must be private.
R_{c17}: If class has composition relationship than multiplicity must be 1.	R_{c18}: Each operation should have a maximum of four parameters.
R_{c19}: An <<entity>> class should have getters and setters.	R_{c20}: Abstract class should have abstract operations.
R_{c21}: Each operation must have a return type.	R_{c22}: Each parameter must have a data type.
R_{c23}: same as R_{i32}.	R_{c24}: same as R_{i33}.
R_{c25}: same as R_{i34}.	R_{c26}: same as R_{i35}.
R_{c27}: If a message is empty then it must be a return type message.	R_{c28}: One activity diagram should reference to one class operation.
R_{c29}: The maximum number of decision point in an activity diagram should be 12.	R_{c30}: Each activity diagram should contain 0 to 3 swimlane.
R_{c31}: Each activity diagram should contain one initial node and one exit point.	R_{c32}: Each activity in an activity diagram should reference to a class operation.
R_{c33}: Dead activities must not present in an activity diagram.	R_{c34}: Each objects of an activity diagram should have corresponding class in a class diagram.

R_{c35}: Dead state must not be present in a state machine diagram.	R_{c36}: State names must be unique.
R_{c37}: All states except root state and initial state should have one incoming transition.	

Table B.2: Rules and Guidelines for Complete Models

Appendix C

Case Study Model

This section contains Bakery system models designed by the groups BLUE and RED. We show here only partial model (does not contain all diagrams) for *incomplete* and *complete* model for iteration 1.

C.1 Incomplete Model of Group BLUE for Iteration 1

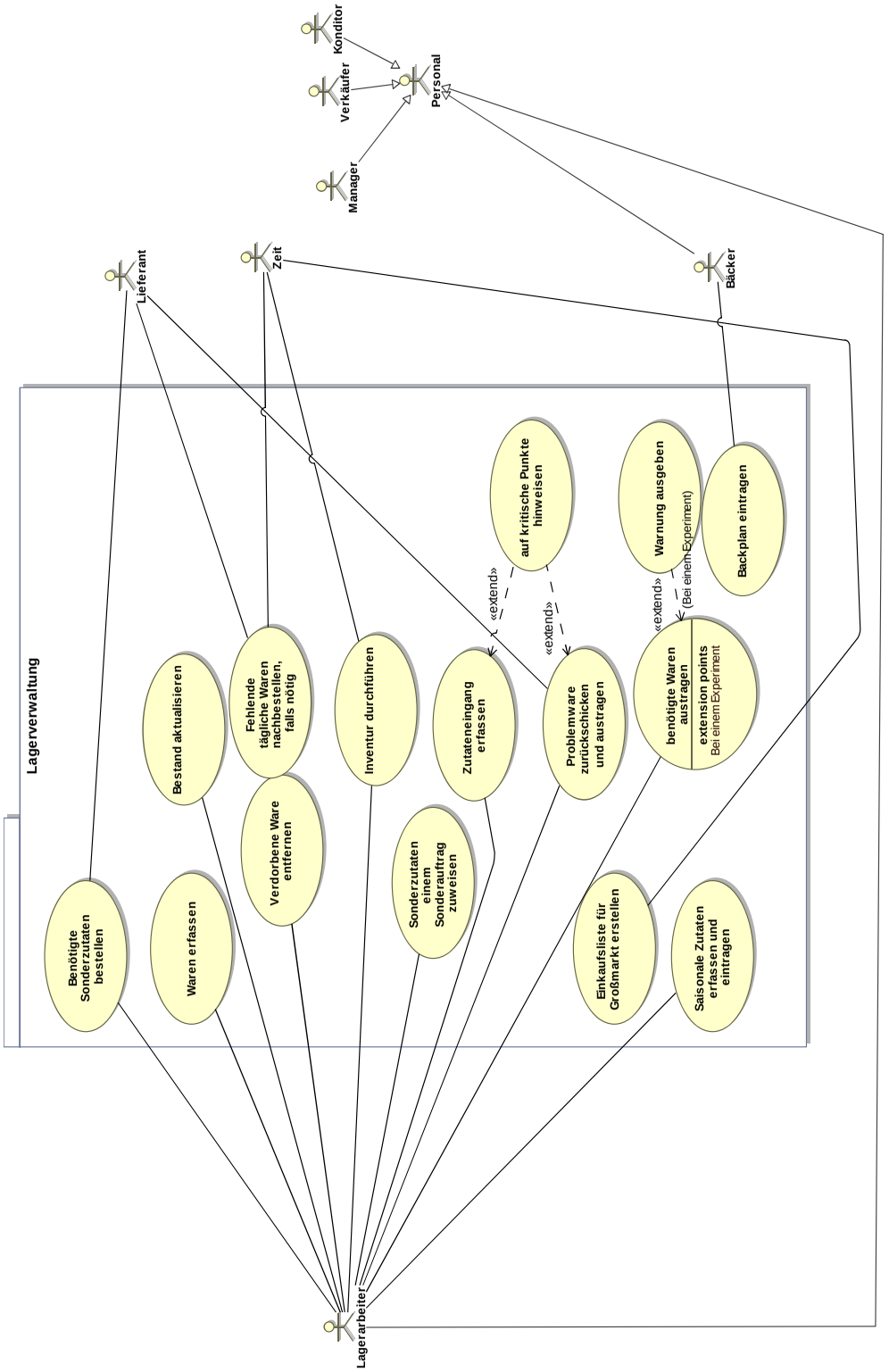


Figure C.1: BLUE Group's Verkauf Subsystem for the Bakery System

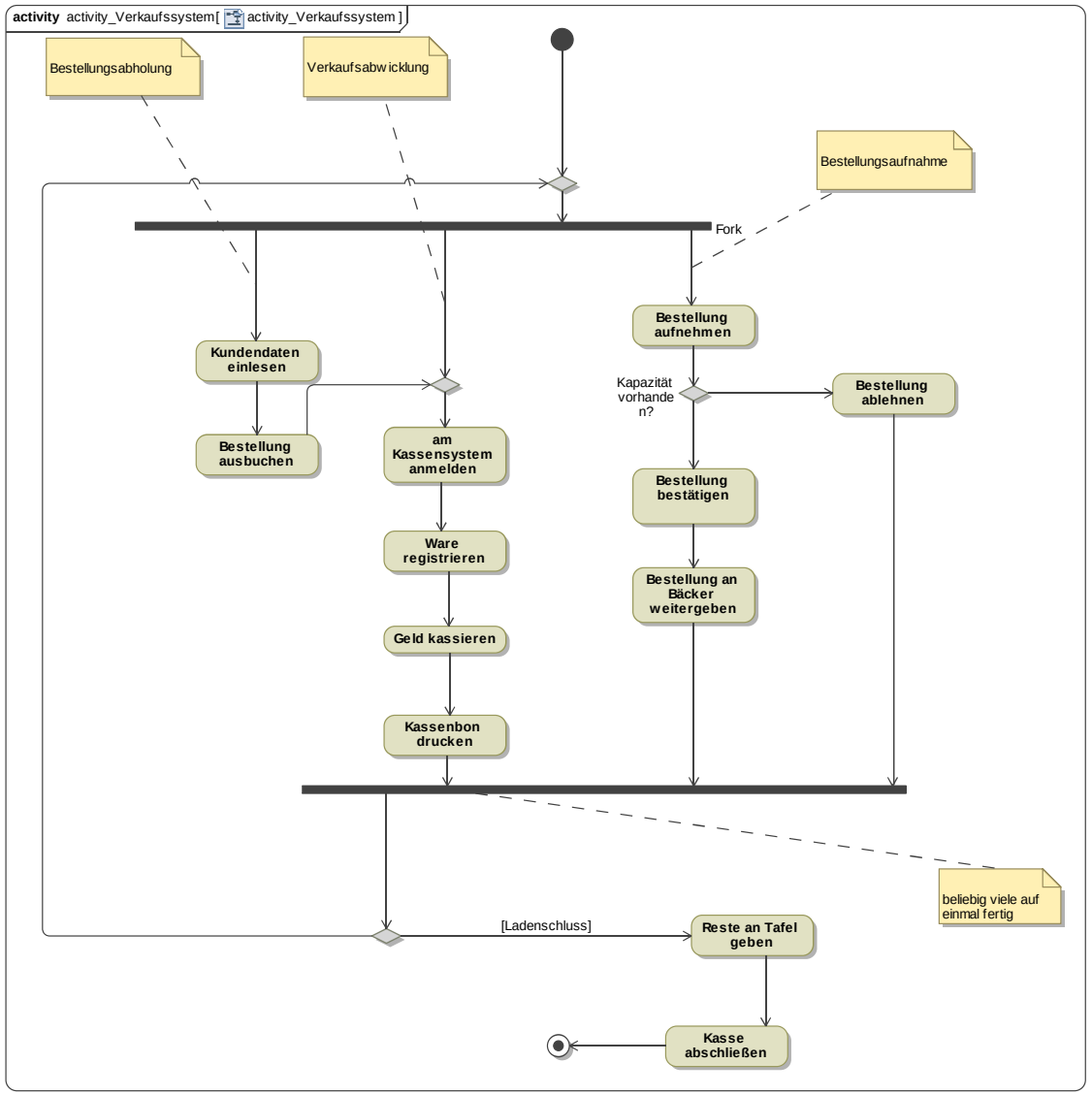


Figure C.2: BLUE Group's Activity Diagram for Verkaufsubsystem the Bakery System

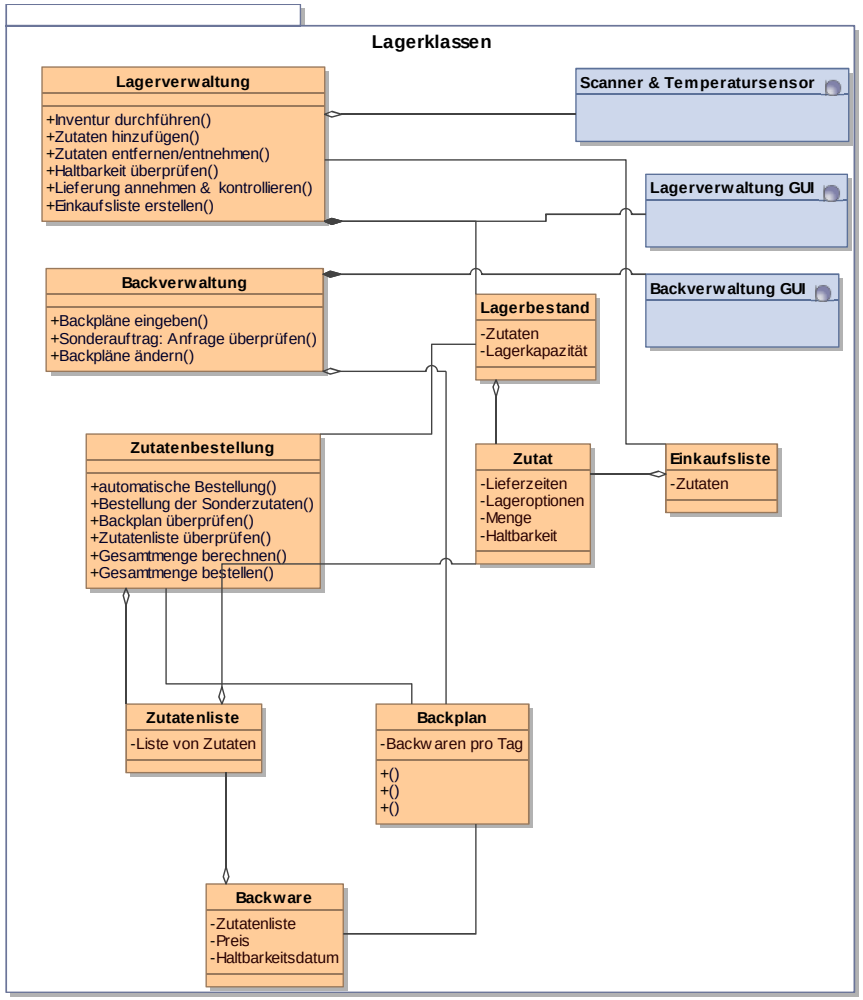


Figure C.3: BLUE Group's LagerKlassen Package for the Bakery System

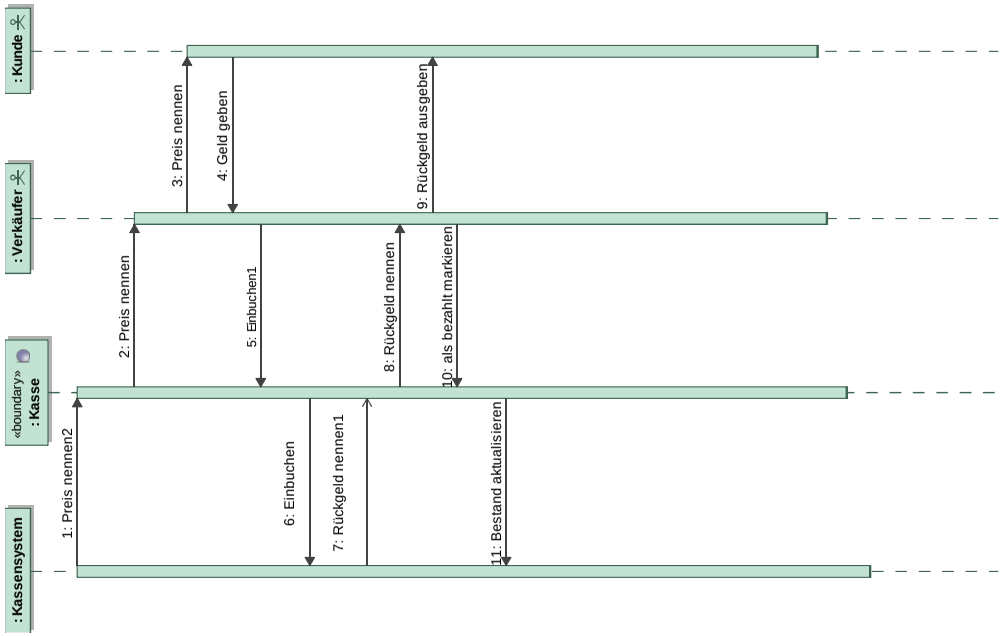


Figure C.4: BLUE Group's Sequence Diagram for Use Case Geld Kassieren for the Bakery System

C.2 Incomplete Model of Group RED for Iteration 1

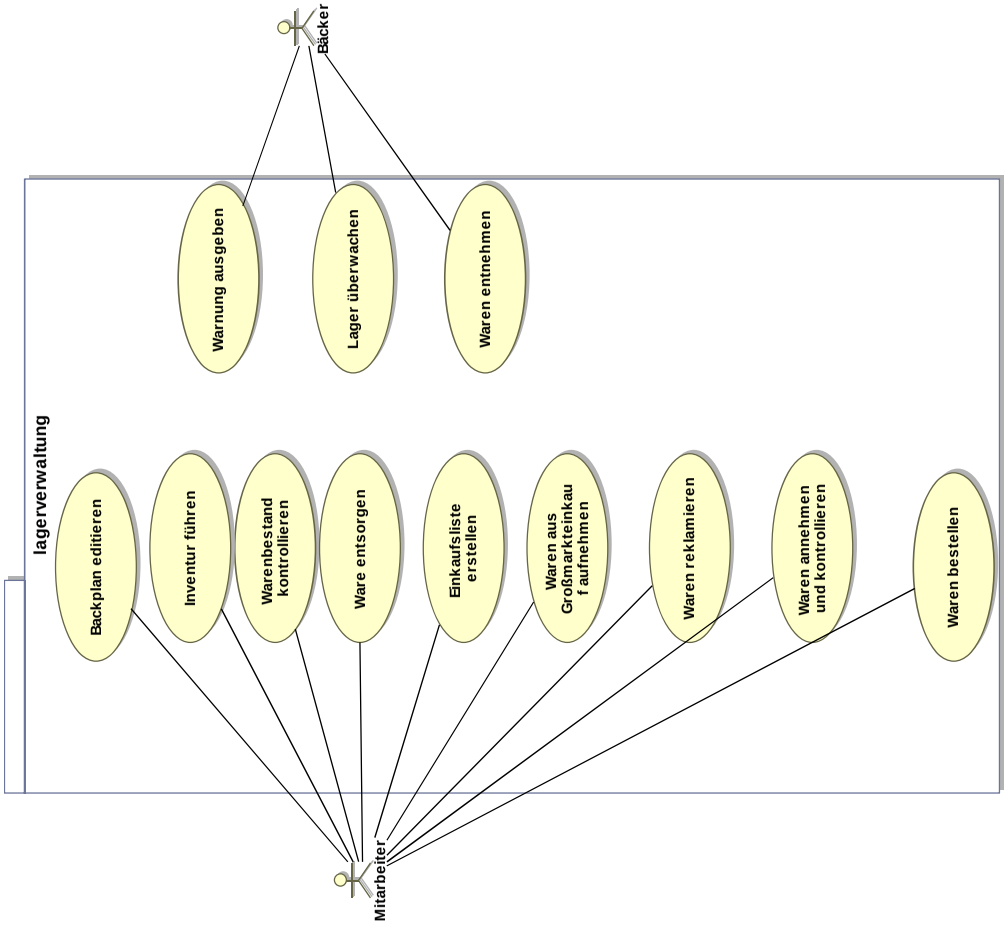


Figure C.5: RED Group's Lagerverwaltung Subsystem for the Bakery System

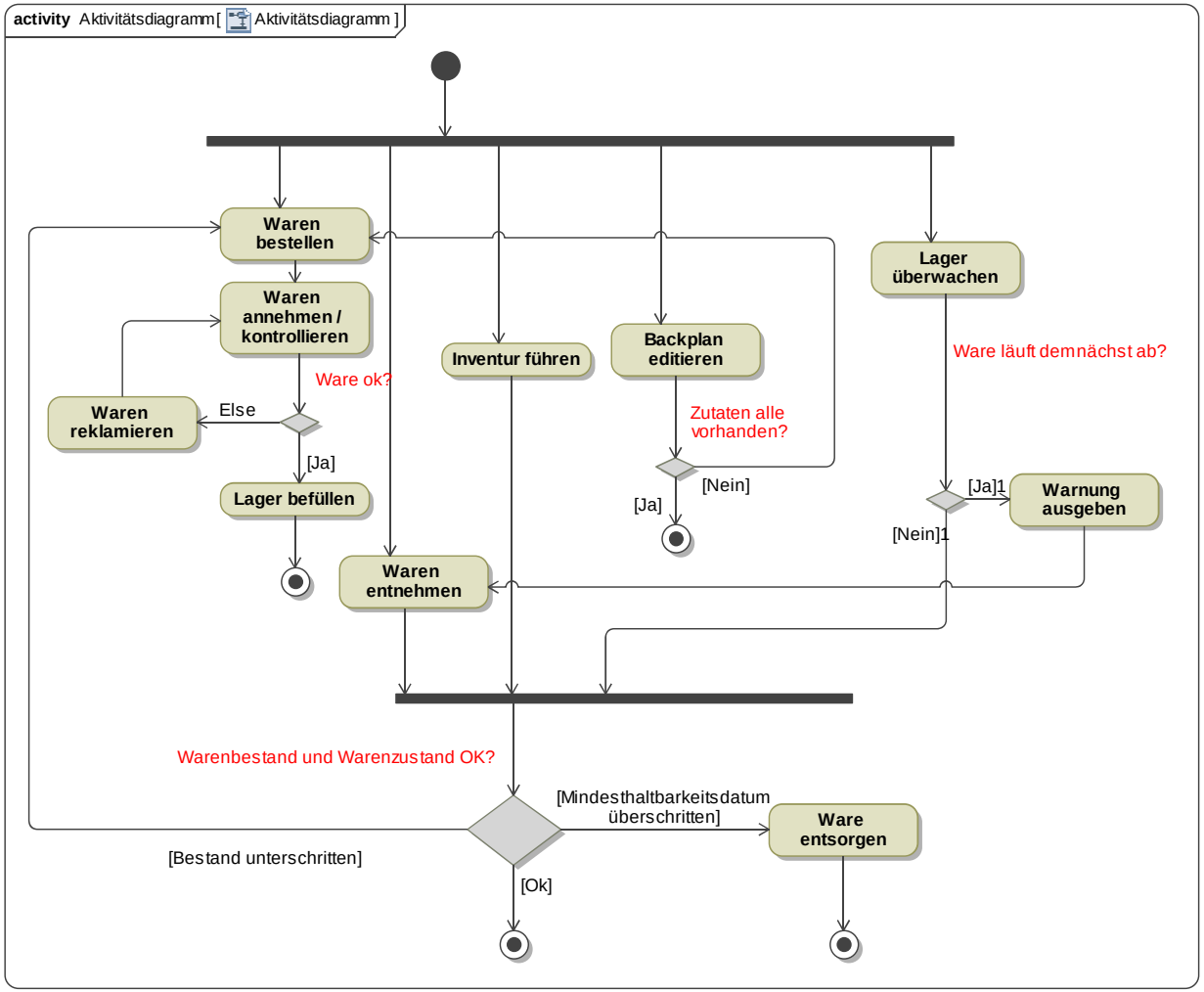


Figure C.6: RED Group's Activity Diagram for the Bakery System

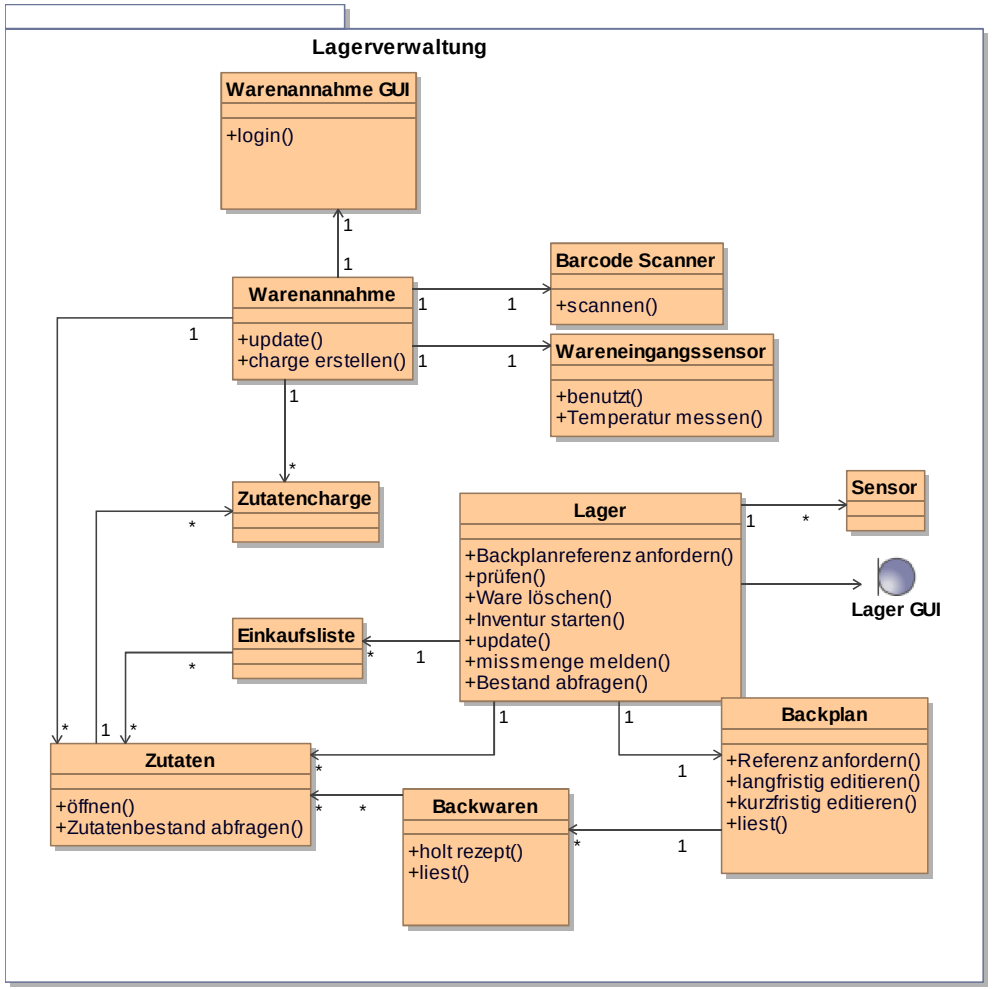


Figure C.7: RED Group's Lagerverwaltung Package for the Bakery System

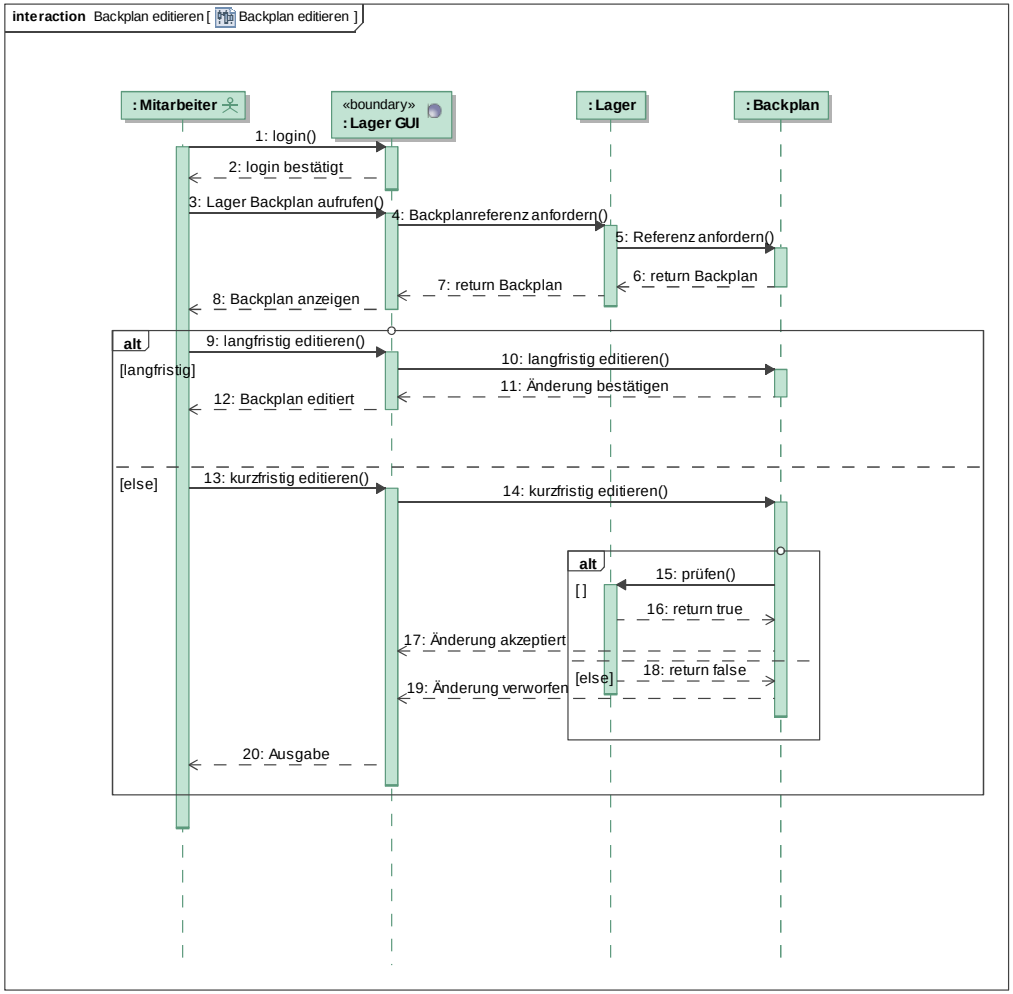


Figure C.8: RED Group's Sequence Diagram for Use Case Backplan Editieren for the Bakery System

C.3 Complete Model of Group BLUE for Iteration 1

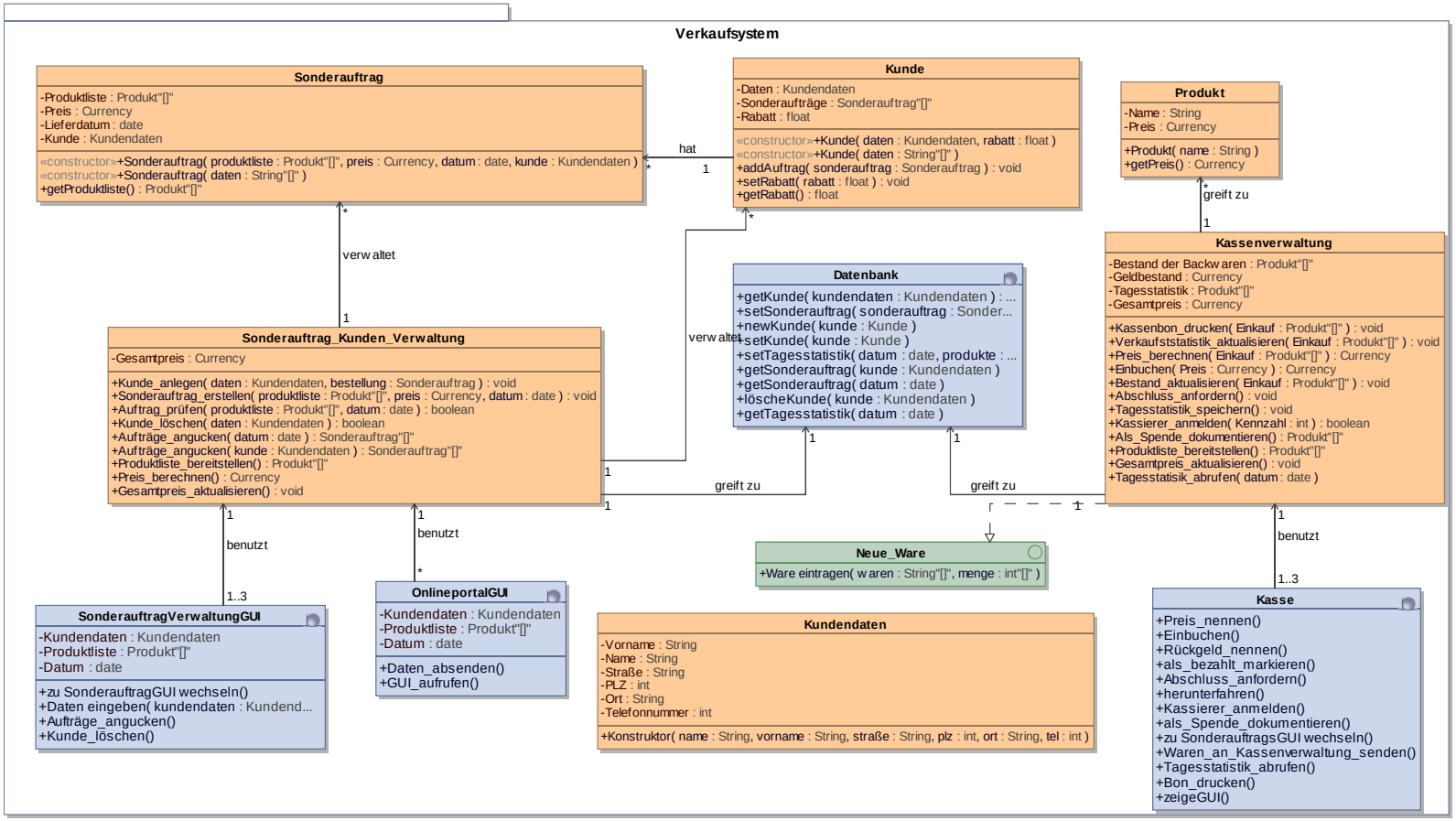


Figure C.9: BLUE Group's Verkaufsystem Package for the Bakery System

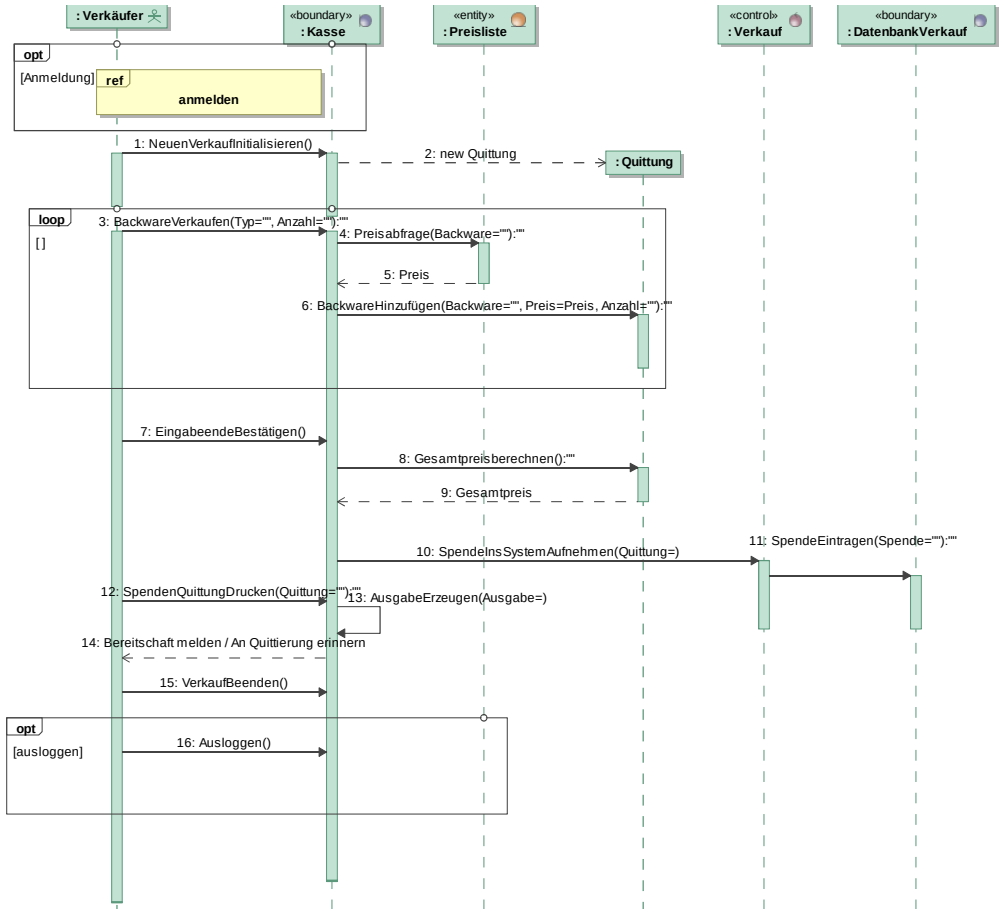


Figure C.10: BLUE Group's Sequence Diagram for BackwareDatenEingeben for the Bakery System

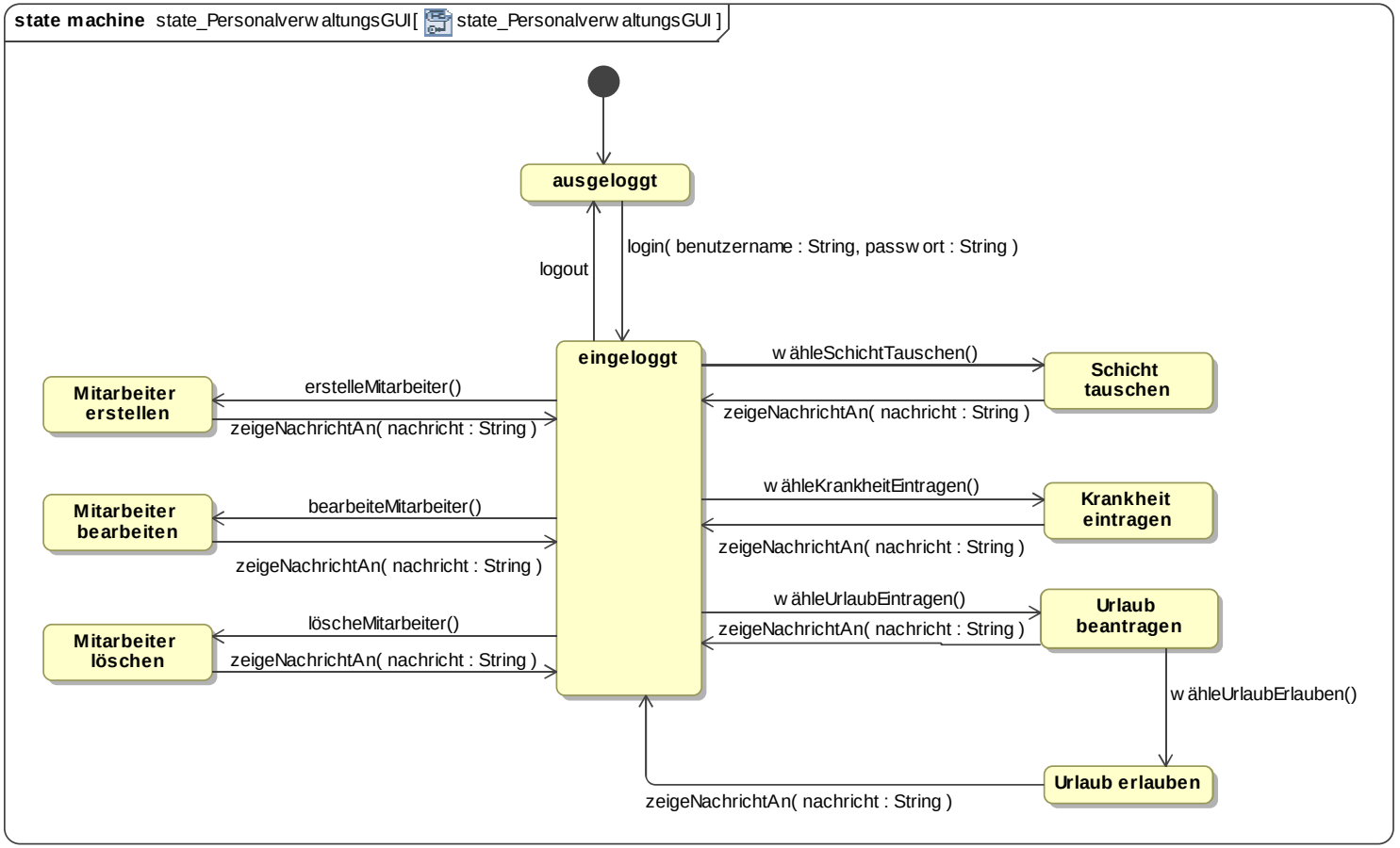


Figure C.11: BLUE Group's State Machine Diagram for PersonalverwaltungsGUI for the Bakery System

C.4 Complete Model of Group RED for Iteration 1

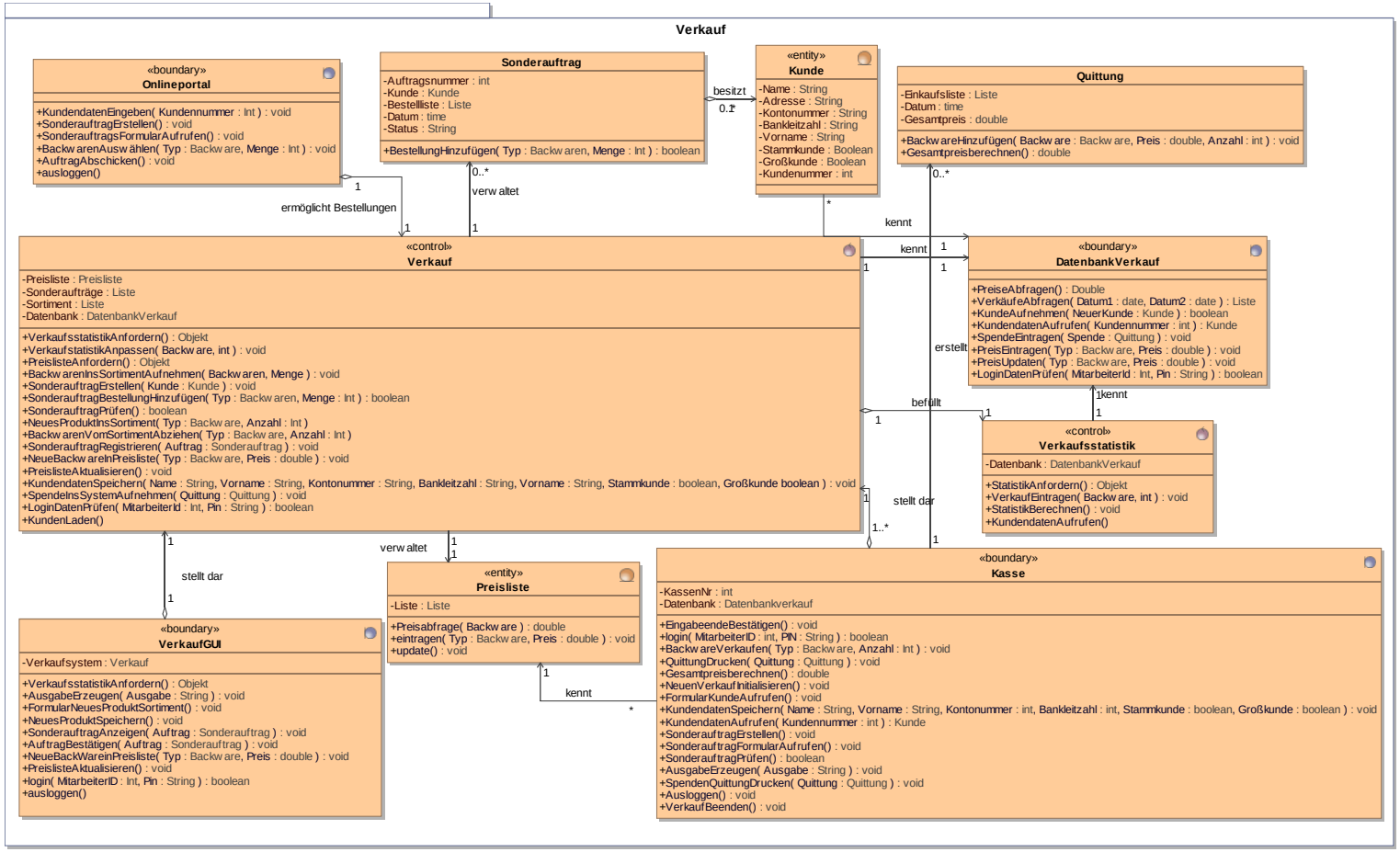


Figure C.12: RED Group's Verkauf Package for the Bakery System

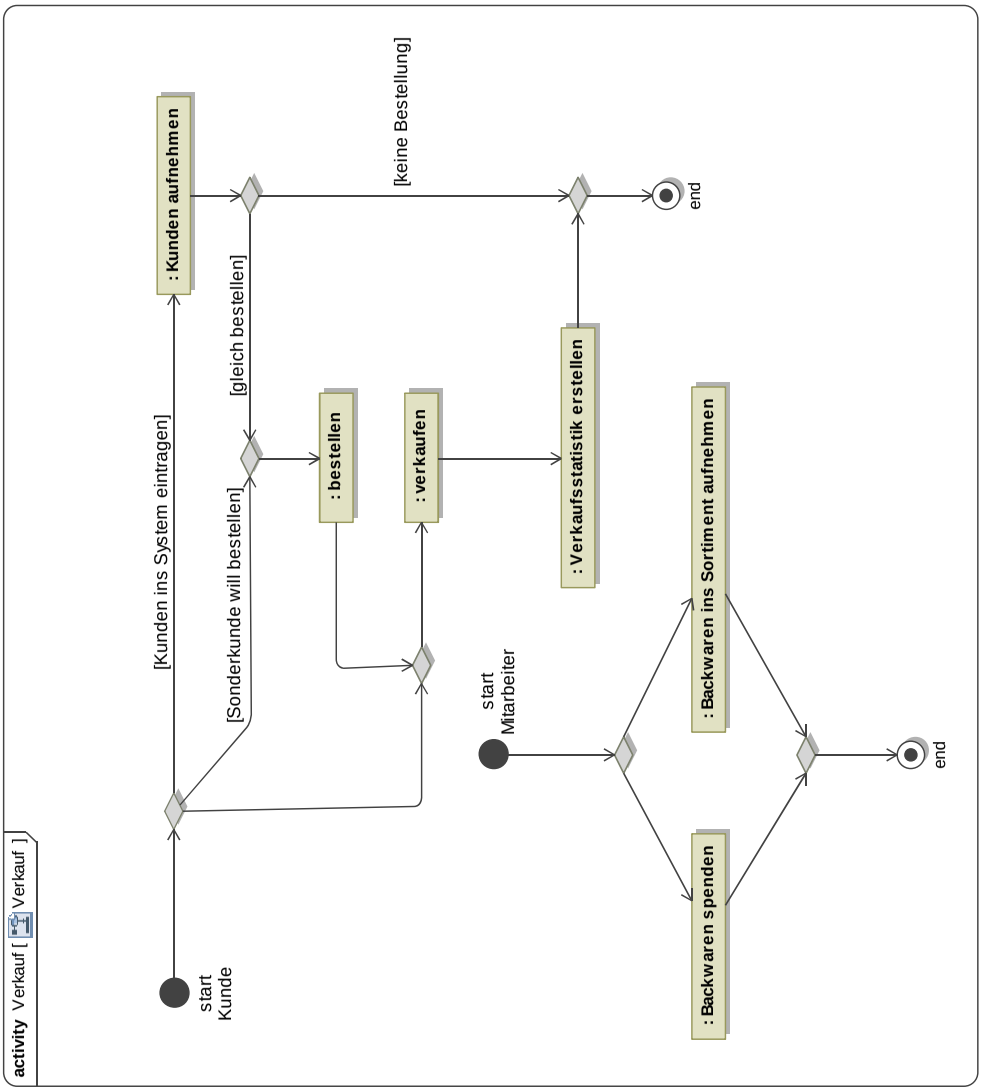


Figure C.13: RED Group's Activity Diagram for Verkauf for the Bakery System

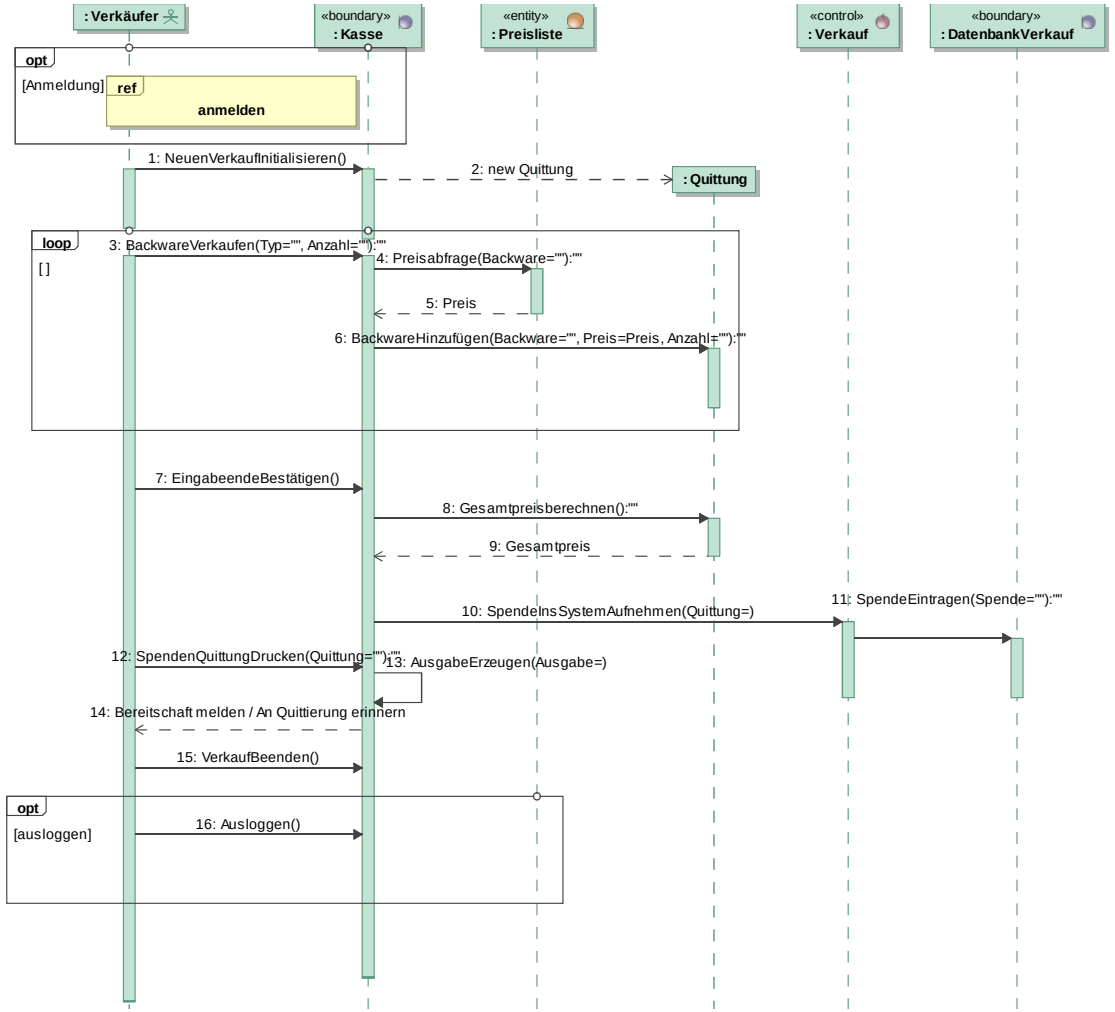


Figure C.14: RED Group's Sequence Diagram for Backware Spenden for the Bakery System

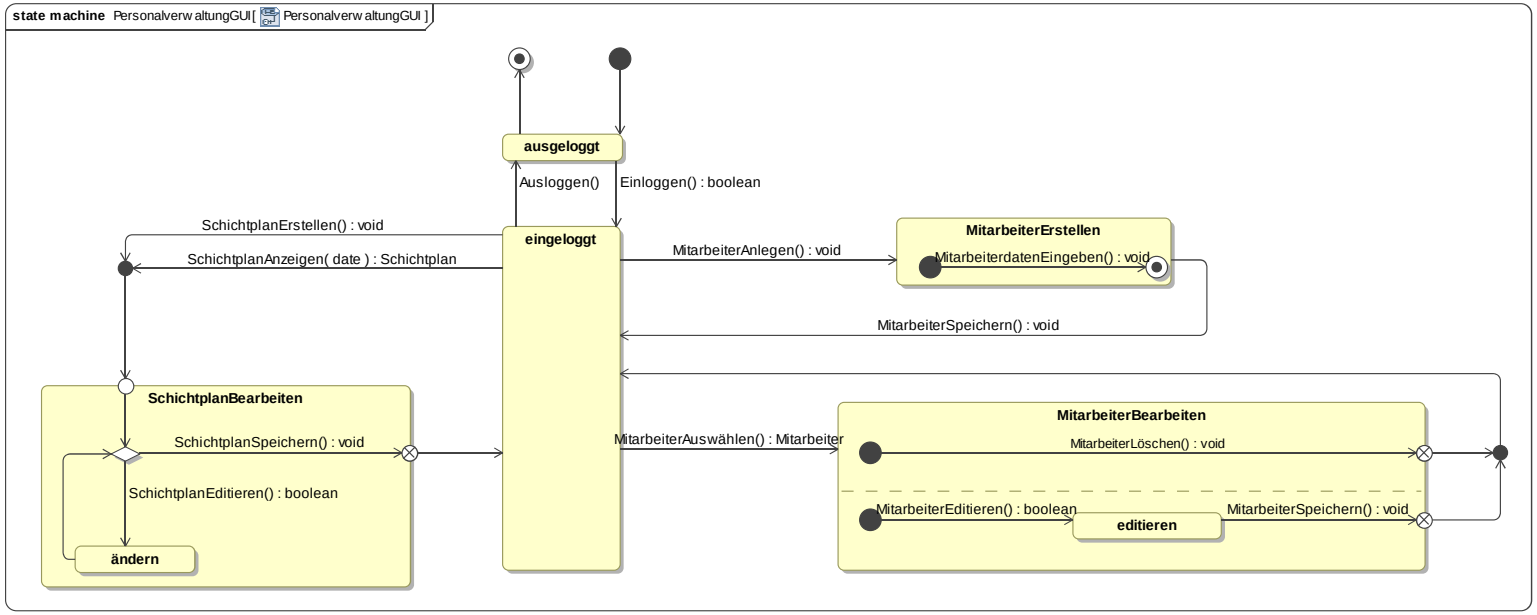


Figure C.15: RED Group's State Machine Diagram for PersonalverwaltungsGUI for the Bakery System

Appendix D

OCL Component

```
1  /*
2  This file contains java helper Class and OCLEvaluator Class.
3  Xtend provides strong connectivity with Java Extensions.
4  Author: Akhtar Ali Jalbani
5  Date: 18-09-2010
6  */
7  import uml;
8  cached Void dump(String s) :
9    JAVA helper.Helper.dump(java.lang.String);
10
11 cached Void evaluateOCL(String ms, uml::Element model) :
12   JAVA helper.OCLEvaluator.evaluateOCL(java.lang.String, org.eclipse.uml2.uml.Element );
13
14 Void delete (emf::EObject e):
15   JAVA org.eclipse.emf.ecore.util.EcoreUtil.delete(org.eclipse.emf.ecore.EObject);
```

Listing D.1: Helper File to use Java in Xtend

```
1  /*This is the main OCL Class which is used to execute OCL checks on UML models
2   * This has been modified from OCLInterpreter Example from eclipse, open source project.
3   * Supported files are added as a workflow component.
4   * Author: Akhtar Ali Jalbani
5   * Date: 18-09-2010
6   * */
7  package helper;
8
9  import org.eclipse.core.runtime.IAdaptable;
10 import org.eclipse.emf.ecore.EObject;
11 import org.eclipse.ocl.OCL;
12 import org.eclipse.ocl.ParserException;
13 import org.eclipse.ocl.expressions.OCLExpression;
14 import org.eclipse.ocl.helper.OCLHelper;
15
16 import workflow.componenets.IOCLFactory;
17 import workflow.componenets.ModelingLevel;
18 import workflow.componenets.UMLOCLFactory;
19
```

```
20 public class OCLEvaluator {
21   public static Object evaluateOCL(String expression, org.eclipse.uml2.uml.Element elem) {
22     EObject context = null;
23
24     if (elem instanceof EObject) {
25       context = (EObject) elem;
26     } else if (elem instanceof IAdaptable) {
27       context = (EObject) ((IAdaptable) elem)
28           .getAdapter(EObject.class);
29     }
30     if (context == null) {
31       return -1111;
32     }
33     IOCLFactory<Object> oclFactory = new UMLOCLFactory(context);
34     ModelingLevel modelingLevel = ModelingLevel.M2;
35     OCL<?, Object, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?> ocl;
36     ocl = oclFactory.createOCL(modelingLevel);
37     OCLHelper<Object, ?, ?, ?> helper = ocl.createOCLHelper();
38     try {
39       modelingLevel.setContext(helper, context, oclFactory);
40       switch (modelingLevel) {
41       case M2:
42         OCLExpression<Object> parsed = helper
43             .createQuery(expression);
44         Object results = ocl.evaluate(context, parsed);
45         return results;
46       case M1:
47         break;
48       }
49     } catch (ParserException e) {
50       return 0;
51     }
52     return -1111;
53   }
54 }
```

Listing D.2: EvaluateOCL.java Class from Interpreter eclipse OCL Example

Appendix E

Rules for Incomplete Models

```
1  // OCL Rules, with Xtend Language for Analysis Model ( Incomplete UML Models)
2  // Author: Akhtar Ali Jalbani
3  // Date:15.09.2010
4  import uml;
5  extension templates::helper;
6  extension org::eclipse::xtend::util::stdlib::io reexport;
7
8  //=========================================
9  //  Rules for UseCase Diagram
10 //=========================================
11 //Ri1: Each use case must be inside the Package (subsystem)
12 cached Boolean isUseCase_InsideSubSystem(uml::UseCase uc):
13   let query = "self.owner->exists(oclIsTypeOf(Package))":
14   query.evaluateOCL(uc);
15
16 //Ri2: Each usecase must be associated with an actor
17 cached Boolean isUseCase_Connected(uml::UseCase uc):
18 uc.getRelationships().reject(e|e.eAllContents.typeSelect(Extend))->
19   uc.getRelationships().reject(e|e.eAllContents.typeSelect(Include))->
20   uc.getRelationships().relatedElement.typeSelect(Actor).size > 0;
21
22 //Ri3: The generalization between usecase must not be present in a use case diagram
23 cached Boolean isUseCase_Inherited(uml::UseCase uc):
24   let query = "self.parents()->size()=0":
25   query.evaluateOCL(uc);
26 //Ri4: Each use case must be refined in a sequence diagram
27 cached Boolean isUseCase_RefinedinSD(uml::UseCase uc):
28   let query = "ownedBehavior->select(b | b.oclIsKindOf(Interaction) and
29 b.oclIsTypeOf(Interaction)->size() > 0)->size() > 0":
30   query.evaluateOCL(uc);
31 //Ri5: Use case should not linked to more than three actors
32 cached Boolean actorToUseCaseRatio(uml::UseCase uc):
33   let query = "self.extend->isEmpty() and
34 getRelationships().relatedElement->select(
35 oclIsTypeOf(Actor))->size()<=3":
36   query.evaluateOCL(uc)->
37   uc.getRelationships().reject(e|e.eAllContents.typeSelect(Extend))->
```

```
38   uc.getRelationships().reject(e|e.eAllContents.typeSelect(Include))->
39   uc.getRelationships().relatedElement.typeSelect(Actor).size!=0 &&
40  uc.getRelationships().relatedElement.typeSelect(Actor).size<=3;
41 //Ri6: Each use case name be 1 to 4 words.
42 cached Boolean hasUseCaseNameLength(uml::UseCase uc):
43   let query = "name.size() = 0 or ( let idx:Sequence(Integer) =
44 Sequence{1..name.size()} in idx->select(i| name.substring(i, i) = ' ')->
45 size()+1 <=4)":
46   query.evaluateOCL(uc);
47
48 //Ri7 : Each subsystem contain minimum 3 and maximum 5 use cases i.e., UC= 3-5
49 cached Boolean isSubsystemBig(uml::Package pkg):
50   let query = "(self.allOwnedElements()->select(
51 oclIsTypeOf(UseCase))->size()>=3) and
52 (self.allOwnedElements()->select(oclIsTypeOf(UseCase))->size()<=5)":
53   query.evaluateOCL(pkg);
54
55 //Ri8: A subsystem name should start with a capital letter
56 //and should be consisting of one to two words
57 cached Boolean isSubsystemNameCapital(uml::Package pkg):
58   let query = "(let startsWith:String=name.substring(1,1) in
59 startsWith.toUpper()=startsWith) and (name.size()=0 or
60 ( let idx:Sequence(Integer) = Sequence{1..name.size()} in
61 idx->select(i| name.substring(i, i) = ' ')->size()+1 <=2))":
62   query.evaluateOCL(pkg);
63 //Ri9: Actor name should start with a capital letter
64 cached Boolean isActorNameCapital(uml::Actor ac):
65   let query = "(let startsWith:String = name.substring(1,1) in
66 startsWith.toUpper() = startsWith) and (name.size() = 0 or
67 (let idx:Sequence(Integer) = Sequence{1..name.size()} in
68 idx->forAll(i| name.substring(i, i) <> ' ')))":
69   query.evaluateOCL(ac);
70
71 //Ri10: The depth of generalization of an actor should not exceed to one
72 cached Boolean isActorDit(uml::Actor ac):
73   let query = "self.parents().parents()->size()=0":
74   query.evaluateOCL(ac);
75
76 //Ri11: Each system name should start with a capital letter
77 //and contain one to two words
78 cached Boolean isSystemNameCapital(uml::Model ac):
79   let query = "(let startsWith:String = name.substring(1,1) in
80 startsWith.toUpper() = startsWith) and (name.size() = 0 or
81 ( let idx:Sequence(Integer) = Sequence{1..name.size()} in
82 idx->select(i| name.substring(i, i) = ' ')->size()+1 <=2))":
83   query.evaluateOCL(ac);
84
85 //Ri12: Actor must be outside the system
86 cached Boolean isActorOutsideFromSystem(uml::Model ac):
87   let query = "self.allOwnedElements()->exists(oclIsTypeOf(Actor))":
88   query.evaluateOCL(ac);
89
90 // Ri13: A use case diagram should not contain more than 20 use cases
91 cached Boolean isPackageBig(uml::Model model):
92   let query = "UseCase.allInstances()->exists(uc|uc->size()<=20)":
93   query.evaluateOCL(model);
94
95 //Ri14: The depth of include should not exceed one.
96 cached Boolean isIncludeUseCaseDit(uml::Include inc):
```

```
97    let query = "self.source->includes(self)->size()<2":
98    query.evaluateOCL(inc);
99
100 //Ri15: The depth of extend use case should not exceed one.
101 cached Boolean isExtendUseCaseDit(uml::Extend inc):
102   let query = "self.source->includes(self)->size()<2":
103   query.evaluateOCL(inc);
104
105
106
107 //=============================================
108 //  Rules for Activity Diagram
109 //=============================================
110
111 //Ri16: Each subsystem should be refined by one activity diagram
112 cached Boolean isUseCaseRefinedByActivityDiagram(uml::Package pkg):
113   let query = "Activity.allInstances().name->includes(self.name)":
114   query.evaluateOCL(pkg);
115
116 //Ri17: Each Activity in activity diagram should refers to usecase
117 cached Boolean isActivityReferenceToUseCase(uml::CallBehaviorAction acty):
118   let query = "UseCase.allInstances().name->includes(self.name)":
119   query.evaluateOCL(acty);
120
121
122 //=============================================
123 //  Rules for Class Diagram
124 //=============================================
125 //Ri18: Each subsystem of use case diagram should be represented as
126 //a package in a class diagram
127 cached Boolean isUCSubSystemRepByCSubsystem(uml::Package sub):
128   let query = "Package.allInstances()->includes(self.name)":
129   query.evaluateOCL(sub);
130
131 //Ri19: Each subsystem should not contain more than 20 classes
132 cached Boolean hasClasses(uml::Package pkg):
133   let query = "self.allOwnedElements()->select(oclIsTypeOf(Class))->size() < 20":
134     query.evaluateOCL(pkg);
135
136 //Ri20: The depth of inheritance tree should not exceed 2
137 cached Boolean isDit(uml::Class cs):
138   let query = "self.superClass.superClass.superClass->size()=0":
139     query.evaluateOCL(cs);
140
141 //Ri21: Multiple Inheritance must not exists
142 cached Boolean isMultipleInheritance(uml::Class cs):
143   let query = "self.general->select(oclAsType(Class))->size()<2":
144     query.evaluateOCL(cs);
145
146 //Ri22: Each class name should start with a capital letter and should be one word
147 cached Boolean isClassNameCapital(uml::Class cs):
148   let query = "(let startsWith:String = name.substring(1,1) in
149 startsWith.toUpper() = startsWith) and (name.size() = 0 or
150 ( let idx:Sequence(Integer) = Sequence{1..name.size()}in
151 idx->forAll(i| name.substring(i, i) <> ' ')))":
152     query.evaluateOCL(cs);
153 // Ri23: <<entity>> should contain at least 3 attributes
154 cached Boolean isEntityClassValid(uml::Class cs):
155   let query = "self.attribute->size()>=3 and
```

```
156 self.getAppliedStereotypes().name->
157 includes('entity')":
158     query.evaluateOCL(cs);
159
160 //Ri24: A <<control>> class should contain 2-5 Operations
161 cached Boolean isControlClassValid(uml::Class cs):
162   let query = "(self.ownedOperation->size()>=2 or
163 self.ownedOperation->size() <= 5) and
164 self.getAppliedStereotypes().name->includes('control')":
165     query.evaluateOCL(cs);
166
167 //Ri25: If class is empty class than class must be the <<boundary>> class
168 cached Boolean isEmptyClassValid(uml::Class cs):
169   let query = "self.allOwnedElements()->size() = 0 and
170 self.getAppliedStereotypes().name->includes('boundary')":
171     query.evaluateOCL(cs);
172
173 //Ri26: Each association must have name
174 cached Boolean isAssociationhasName(uml::Association as):
175   let query = "self.name <> ' '":
176     query.evaluateOCL(as);
177
178
179 //Ri27: Each association must specify multiplicity values at both ends.
180 cached Boolean isMultiplicityValue(uml::Association as):
181   let query = "self.memberEnd ->forAll ( n |
182 (not n.lowerValue.oclIsUndefined()) or
183 (not n.upperValue.oclIsUndefined()))":
184     query.evaluateOCL(as);
185
186 //Ri28: Each class should have 1 to 5 association (1-5)
187 cached Boolean isAssociationPerClass(uml::Class cs):
188   let query = "self.attribute.association->size()>0 ||
189 self.attribute.association->size()< 6":
190     query.evaluateOCL(cs);
191
192 //Ri29: Each association name should be start with a lower case letter
193 cached Boolean isAssociationNameLower(uml::Association as):
194     let query = "let startsWith:String = name.substring(1,1) in
195 startsWith.toLower() = startsWith":
196     query.evaluateOCL(as);
197
198 //Ri30: Classes should not be linked with composition or
199 //aggregation type of association.
200 cached Boolean hasAggregationOrComposition(uml::Property p):
201   let query = "let opposite:Property = self.opposite.
202 association.memberEnd->any(e|e<>self) in (opposite.aggregation<>
203 AggregationKind::shared) and (not(opposite.isComposite))":
204     query.evaluateOCL(p);
205
206 //Ri31: The links to classes belonging to another package must be uni-directional
207 cached Boolean isClassUniDirectional(uml::Association as):
208   let query = "self.memberEnd.isNavigable()->includes(false) and
209  self.getEndTypes()->select(oclIsTypeOf(Class))->
210 exists(e1,e2|e1.owner <> e2.owner)":
211     query.evaluateOCL(as);
212 //=============================================
213 //    Rules for Sequence Diagram
214 //=============================================
```

```
215
216 //Ri32: Each Sequence diagram have atleast one actor on a lifeline
217 cached Boolean isActorInSequenceDiagram(uml::Interaction inaction):
218   let query = "self.lifeline.represents.type->exists(oclIsTypeOf(Actor))->size() >0":
219     query.evaluateOCL(inaction);
220
221 //Ri33: Each objector lifeline in a sequence diagram must have corresponding
222 //class/actor in a class diagram
223 cached Boolean isObjectReferesToClass(uml::Lifeline line):
224   let query = "self.represents.type->exists(oclIsTypeOf(Class)) or
225  self.represents.type->exists(oclIsTypeOf(Actor)) or
226 self.represents.type->exists(oclIsTypeOf(Interface))":
227     query.evaluateOCL(line);
228
229 //Ri34: Every call message received by the lifeline should have
230 //corresponding operation in a class diagram
231 cached Boolean isMessageReferesToOperation(uml::Message msg):
232   let query = "((not receiveEvent.oclAsType(MessageOccurrenceSpecification).
233 event.oclIsUndefined())
234 and (receiveEvent.oclAsType(MessageOccurrenceSpecification).
235 event.oclIsTypeOf(CallEvent))) implies not
236 receiveEvent.oclAsType(MessageOccurrenceSpecification).
237 event.oclAsType(CallEvent).operation.oclIsUndefined()":
238     query.evaluateOCL(msg);
239 //Ri35: If there is a message call between two lifelines then
240 //there must be an association between two classes
241 cached Boolean hasMessageCallRelationToClassAssocaition(uml::Lifeline ll):
242   let query = "(MessageOccurrenceSpecification.allInstances().covered->
243 includes(self)) and (Association.allInstances().getEndTypes()->select(oclIsTypeOf(Class))->
244 asSet()->includes(self.represents.type))":
245     query.evaluateOCL(ll);
246
247 //Ri36: Each message must be labeled.
248 cached Boolean isMessageLabeled(uml::Message msg):
249   let query = "self.name <> ' '":
250     query.evaluateOCL(msg);
251
252 ///=========================================================
```

Listing E.1: Rules for Incomplete Model

Appendix F

Rules for Complete Models

```
1  // OCL Rules with xtend language for Design Model (Complete UML Models)
2  // Author: Akhtar Ali Jalbani
3  // Date:16.09.2010
4  import uml;
5  extension templates::helper;
6  extension Rules::designModel;
7  extension org::eclipse::xtend::util::stdlib::io reexport;
8
9  //Rc1: Every class should have attributes
10 cached Boolean isClasshasAttributes(uml::Class cs):
11   let query =”self.ownedAttribute->size()>0”:
12     query.evaluateOCL(cs);
13
14 //Rc2: Every class should have operations
15 cached Boolean isClasshasOperations(uml::Class cs):
16   let query =”self.getAllOperations()->size()>0”:
17     query.evaluateOCL(cs);
18
19 //Rc3: The depth of inheritance tree should be less than 4
20 cached Boolean isClassDit(uml::Class cs):
21   let query =”self.superClass.superClass.superClass.superClass->size() = 0”:
22     query.evaluateOCL(cs);
23 //Rc4: Multiple Inheritance must not exists
24   //===same as Ri21
25
26 //Rc5: same as Ri22
27
28 // Rc6: Each class should have maximum 10 operations.
29 cached Boolean hasToomanyOperations(uml::Class cs):
30   let query =”self.ownedOperation->size() <= 10”:
31     query.evaluateOCL(cs);
32
33 //==class Association rules
34
35 //Rc7}: Each association must have name
36   //===same as Ri26
37
```

```
38
39 //Rc8: Each class should have 1−5 associations
40   //===Same as Ri28
41
42
43 //Rc9: Each association name should start with a lower case letter.
44   //=== same as Ri29
45
46
47 //Rc10: Each association must have direction
48 cached Boolean isDirectedAssociation(uml::Association as):
49   let query = "(self.memberEnd.isNavigable()−>includes(false))":
50     query.evaluateOCL(as);
51
52
53 //Rc11: Each association must specify multiplicity and it must be n to 1.
54   cached Boolean hasMultiplicityDefined(uml::Association as):
55   let query = "let opposite:Property = association.memberEnd−>any(e|e <> self)
56 in (not opposite.oclIsUndefined() and not upperValue.oclIsUndefined())
57 implies (upper = 1)":
58     query.evaluateOCL(as);
59
60 //Rc12: Association class must not be present in a design model.
61 cached Boolean hasAssociationClass(uml::Model model):
62   let query ="AssociationClass.allInstances()−>size()=0":
63     query.evaluateOCL(model);
64
65 //Rc13: Each package should have maximum 20 classes.
66 cached Boolean hasTooManyClasses(uml::Package pkg):
67   let query ="(self.allOwnedElements()−>select(oclIsTypeOf(Class))
68 −>size()) <= 20":
69     query.evaluateOCL(pkg);
70
71 //Rc14: The links to classes belonging to another package must be uni−directional.
72   // Same as Ri31
73
74
75 //Rc15: The maximum package nesting level should be 2.
76 cached Boolean ispackageNestingLvel(uml::Package pkg):
77   let query ="(self.owner.owner.owner−>size())=0":
78     query.evaluateOCL(pkg);
79
80 //Rc16: Each attribute must have data type and should be private.
81 // association will also be considered in this query.
82 cached Boolean isAttributeDataType(uml::Property prop):
83   let query ="self.visibility = VisibilityKind::private":
84     query.evaluateOCL(prop);
85
86 //Rc17: If Class has composition relationship than multiplicity must be 1.
87 cached Boolean isCompositionRelationship(uml::Property prop):
88   let query ="let opposite:Property = association.memberEnd−>
89 any(e|e <> self) in (not opposite.oclIsUndefined() and
90 opposite.isComposite and not upperValue.oclIsUndefined())
91 implies (upper = 1)":
92     query.evaluateOCL(prop);
93
94
95 //Rc18: Each operation should have maximum four parameters.
96 cached Boolean hasParameters(uml::Operation op):
```

```
97    let query =”self.ownedParameter->reject(e|e.direction =
98   ParameterDirectionKind::return)->size()< 5”:
99       query.evaluateOCL(op);
100
101  //Rc19: Entity class should have getters and setters.
102  cached Boolean hasGettersAndSetters(uml::Class cs):
103    let query =”self.ownedOperation->exists(name.substring(1, 3 ) =
104  'set' or name.substring(1, 3 ) = 'get')”:
105       query.evaluateOCL(cs);
106  //Rc20: Abstract class should have abstract operations.
107  cached Boolean isAbsractOperationInAbsractClass(uml::Operation op):
108    let query =”(self.isAbstract implies self.owner->exists(isAbstract))”:
109       query.evaluateOCL(op);
110
111
112  //Rc21: Each operation must have return type.
113  cached Boolean hasReturnType(uml::Operation par):
114    let query =”self.ownedParameter->exists(e|e.direction =
115  ParameterDirectionKind::return)”:
116       query.evaluateOCL(par);
117
118  //Rc22: Each parameter must have data type.
119  cached Boolean hasParameterType(uml::Parameter par):
120    let query =”self.type->notEmpty()”:
121       query.evaluateOCL(par);
122
123  //Rc23: Each Sequence diagram have at least one actor on a lifeline
124    // Same as Ri32
125
126  //==Lifeline Rules
127  //Rc24: Each object or lifeline in sequence diagram must have corresponding
128  //class in a class diagram
129    //Same as Ri33
130
131  //Rc25: Every call message received by the lifeline should have corresponding
132  //operation in a class
133    // Same as Ri34
134
135  //Rc26: If there is a message call between two lifelines then there must
136  //be an association between corresponding classes
137    // Same as Ri35
138
139
140  //Rc27: If message is empty then it must be a return type message.
141  cached Boolean isReturnMessage(uml::Message msg):
142    let query =”self.name =' ' implies self.messageSort=MessageSort::reply”:
143       query.evaluateOCL(msg);
144
145  //Rc28: One activity diagram should reference to one class operation.
146  //( Activity diagram per operation should be one).
147  cached Boolean hasReferenceClass(uml::Activity ac):
148    let query =”self.specification->select(oclIsTypeOf(Operation))->size()=1”:
149       query.evaluateOCL(ac);
150
151  //Rc29: The maximum decision point should be 12 in activity diagram.
152  cached Boolean hasTooManyDecisionPoints(uml::Activity act):
153    let query =”self.allOwnedElements()->select(oclIsTypeOf(DecisionNode))->size() <13”:
154       query.evaluateOCL(act);
155
```

```
156 //Rc30: Each activity diagram should contain 0 to 3 swim lane.
157 cached Boolean hasTooManySwimlanes(uml::Activity act):
158   let query =”self.allOwnedElements()−>select(oclIsTypeOf(ActivityPartition))−>size() <4”:
159     query.evaluateOCL(act);
160
161 //Rc31: Each activity diagram should contain one initial node and one exit node.
162 cached Boolean hasTooManyIntialAndExitNodes(uml::Activity act):
163   let query =”self.allOwnedElements()−>select(oclIsTypeOf(InitialNode))−>size() = 1
164 and self.allOwnedElements()−>select(oclIsTypeOf(ActivityFinalNode))−>size() = 1”:
165     query.evaluateOCL(act);
166 //==rules for CallOperation
167 //Rc32: Activity in activity diagram should reference to a class operations.
168 cached Boolean hasActivityReferenceToClass(uml::Action coa):
169   let query =”Class.allInstances().ownedOperation−>exists(e|e.name = self.name)”:
170     query.evaluateOCL(coa);
171
172 //Rc33: Dead activity must not present in activity diagram
173 cached Boolean isDeadActivity(uml::Action coa):
174   let query =”self.incoming −>size()<> 0 and self.outgoing−>size() <> 0”:
175     query.evaluateOCL(coa);
176
177 //==
178 //Rc34: Each objects of activity diagram should have corresponding
179 //class in a class diagram.
180 cached Boolean hasObjectReferenceToClass(uml::CentralBufferNode cbn):
181   let query =”self.type−>exists(oclIsTypeOf(Class))”:
182     query.evaluateOCL(cbn);
183
184
185 //Rc35: Dead state must not present in a state machine.
186 cached Boolean isDeadState(uml::State s):
187   let query =” self.incoming−>size()<> 0 and self.outgoing−>size()<> 0”:
188     query.evaluateOCL(s);
189
190 //Rc36: State names must be unique.
191 cached Boolean isStateUnique(uml::State sm):
192   let query =”State.allInstances()−>forAll (p,q|p.name<>q.name implies p=q )”:
193     query.evaluateOCL(sm);
194
195 //Rc37: All states except root state and initial state should have one incoming transition.
196 cached Boolean hasTooManyTransitions(uml::StateMachine sm):
197   let query =”PseudoState.allInstances()−>select(PseudoState=PseudoStateKind::initial))−>
198 size()>=1”:
199     query.evaluateOCL(sm);
```

Listing F.1: Rules for Complete Model

Appendix G

Html Report Generation in Xpand for Incomplete Model

```
1  «REM»
2  This is root file to generate report for incomplete and complete models by expanding
3  the corresponding templates.
4
5  Author: Akhtar Ali Jalbani
6  Date: 18−09−2010
7
8  «ENDREM»
9  «IMPORT uml»
10
11 «REM»This template is a root template, that can be used to create html
12 report for incomplete and complete model. «ENDREM»
13 «DEFINE main FOR Model»
14   «REM»Report for incomplete model
15   «EXPAND templates::redGroup_analysisModel::analysisModel»
16   «REM»«EXPAND templates::bluegroup_analysisModel::analysisModel»«ENDREM»
17   «REM»Report for complete model
18   «EXPAND templates_designModel::redGroup_designModel::designModel»
19   «ENDREM»
20   «EXPAND templates_designModel::blueGroup_designModel::designModel»
21 «ENDDEFINE»
```

Listing G.1: Xpand Main File

```
1  «REM»
2  This file contains html report for the incomplete models at analysis phase.
3  Author: Akhtar Ali Jalbani
4  Date: 18−09−2010
5
6  «ENDREM»
7  «IMPORT uml»
8  «EXTENSION templates::helper»
9  «EXTENSION Rules::analysisModel»
10
```

```
11 «DEFINE analysisModel FOR Model»
12 «FILE "bluegroup_analysisModel.html"»
13 <HTML>
14 <HEAD>
15
16 </HEAD>
17 <Title>Report for Incomplete Models at Analysis Phase .....<Title>
18 <BODY >
19     <H1><a name="PM">Report for Analysis model:
20 ( Bakery system) − Blue Group.</a></H1>
21
22     <TABLE BORDER="1">
23         <TR>
24           <th width="60%">Rule</th>
25           <th width="40%">Qualified Name</th>
26         </TR><TR>
27        «REM»UseCase Rules
28        <TH Colspan=2 align= left>UseCase Rules</TH>
29
30         «EXPAND rule1 FOREACH eAllContents.typeSelect(UseCase)»
31
32         «EXPAND rule2 FOREACH eAllContents.typeSelect(uml::UseCase).
33             reject(e|!e.extend.isEmpty || !e.include.isEmpty)»
34
35         «EXPAND rule3 FOREACH eAllContents.typeSelect(UseCase)»
36         «EXPAND rule4 FOREACH
37         eAllContents.typeSelect(UseCase).select(e|e.ownedBehavior==uml::Interaction)»
38
39         «EXPAND rule5 FOREACH eAllContents.typeSelect(UseCase).
40         reject(e|!e.extend.isEmpty || !e.include.isEmpty)»
41
42         «EXPAND rule6 FOREACH eAllContents.typeSelect(UseCase)»
43         </TR><TR>
44         <TH Colspan=2 align= left>Subsystem Rules ( Package)</TH>
45
46         «EXPAND rule7 FOREACH eAllContents.typeSelect(Package).
47         reject(e|e.qualifiedName == "Data::Bäckerei" ||
48         e.qualifiedName == "Data::Actors" ||
49         e.qualifiedName == "Data::UML Standard Profile" ||
50           e.qualifiedName == "Data::Bäckerei::Lagerverwaltung::Lagerklassen" ||
51         e.qualifiedName == "Data::Bäckerei::Personalverwaltung::Personalklassen" ||
52         e.qualifiedName == "Data::Bäckerei::Verkaufssystem::Verkaufsklassen")»
53
54         «EXPAND rule8 FOREACH eAllContents.typeSelect(Package).
55         reject(e|e.qualifiedName =="Data::Bäckerei" ||
56         e.qualifiedName == "Data::Actors" ||
57         e.qualifiedName == "Data::UML Standard Profile" ||
58         e.qualifiedName == "Data::Bäckerei::Lagerverwaltung::Lagerklassen" ||
59         e.qualifiedName == "Data::Bäckerei::Personalverwaltung::Personalklassen" ||
60         e.qualifiedName == "Data::Bäckerei::Verkaufssystem::Verkaufsklassen")»
61         </TR><TR>
62         <TH Colspan=2 align= left>Actor Rules</TH>
63
64         «EXPAND rule9 FOREACH eAllContents.typeSelect(Actor)»
65
66         «EXPAND rule10 FOREACH eAllContents.typeSelect(Actor)»
67         </TR<TR>
68         <TH Colspan=2 align= left>System/Model Rules</TH>
69
```

```
70        «EXPAND rule11 FOREACH eAllContents.typeSelect(Model)»
71
72        «EXPAND rule12 FOREACH eAllContents.typeSelect(Model)»
73        </TR><TR>
74        <TH Colspan=3 align= left>UseCase Package Rules</TH>
75
76        «EXPAND rule13 FOREACH eAllContents.typeSelect(Model)»
77        </TR><TR>
78        <TH Colspan=3 align= left>UseCaseInclude and Extend Rules</TH>
79
80        «EXPAND rule14 FOREACH eAllContents.typeSelect(Include)»
81
82        «EXPAND rule15 FOREACH eAllContents.typeSelect(Extend)»
83        </TR><TR>
84        «REM»−−−−−Activity Diagram Rules −−−−−«ENDREM»
85        <TH Colspan=2 align= left>Activity traceability to usecase Rules</TH>
86
87        «EXPAND rule16 FOREACH eAllContents.typeSelect(Package).
88        reject(e|e.qualifiedName == ”Data::Bäckerei” || e.qualifiedName ==
89        ”Data::Actors” ||
90        e.qualifiedName == ”Data::UML Standard Profile” ||
91        e.qualifiedName == ”Data::Bäckerei::Lagerverwaltung::Lagerklassen” ||
92        e.qualifiedName == ”Data::Bäckerei::Personalverwaltung::Personalklassen” ||
93        e.qualifiedName == ”Data::Bäckerei::Verkaufssystem::Verkaufsklassen”)»
94
95        «EXPAND rule17 FOREACH eAllContents.typeSelect(CallBehaviorAction)»
96        </TR>
97        «REM»−−−−−Class Diagram Rules −−−−−−−«ENDREM»
98        <TR>
99        <TH Colspan=2 align= left>Class Package Rules</TH>
100       «EXPAND rule18 FOREACH eAllContents.typeSelect(Package).
101       reject(e|e.qualifiedName == ”Data::Bäckerei::Personalverwaltung” ||
102       e.qualifiedName ==”Data::Bäckerei::Verkaufssystem” ||
103       e.qualifiedName == ”Data::Bäckerei::Lagerverwaltung” ||
104       e.qualifiedName == ”Data::Bäckerei” ||
105       e.qualifiedName == ”Data::Actors” ||
106       e.qualifiedName == ”Data::UML Standard Profile”)»
107
108       «EXPAND rule19 FOREACH eAllContents.typeSelect(Package).
109       reject(e|e.qualifiedName == ”Data::Bäckerei::Personalverwaltung” ||
110       e.qualifiedName ==”Data::Bäckerei::Verkaufssystem” ||
111         e.qualifiedName == ”Data::Bäckerei::Lagerverwaltung” ||
112       e.qualifiedName == ”Data::Bäckerei” || e.qualifiedName == ”Data::Actors” ||
113       e.qualifiedName == ”Data::UML Standard Profile”)»
114       </TR><TR>
115       <TH Colspan=3 align= left>Class Rules</TH>
116     «EXPAND rule20 FOREACH
117       eAllContents.typeSelect(uml::Class).
118       reject(e|Interaction.isInstance(e)|| Activity.isInstance(e))»
119
120     «EXPAND rule21 FOREACH
121       eAllContents.typeSelect(uml::Class).
122       reject(e|Interaction.isInstance(e)|| Activity.isInstance(e))»
123
124     «EXPAND rule22 FOREACH
125       eAllContents.typeSelect(uml::Class).
126       reject(e|Interaction.isInstance(e)|| Activity.isInstance(e))»
127
128     «EXPAND rule23 FOREACH
```

```
129         eAllContents.typeSelect(uml::Class).
130         reject(e|Interaction.isInstance(e)||Activity.isInstance(e)).
131         select(e|e.getApplicableStereotypes().first().name==”entity”)»
132
133       «EXPAND rule24 FOREACH
134         eAllContents.typeSelect(uml::Class).
135         reject(e|Interaction.isInstance(e)|| Activity.isInstance(e)).
136         select(e|e.getApplicableStereotypes().first().name==”control”)»
137
138       «EXPAND rule25 FOREACH
139         eAllContents.typeSelect(uml::Class).
140         reject(e|Interaction.isInstance(e)|| Activity.isInstance(e)).
141         select(e|e.getApplicableStereotypes().first().name==”boundary”)»
142         </TR><TR>
143         <TH Colspan=2 align= left>Class Association Rules</TH>
144         «EXPAND rule26 FOREACH
145         eAllContents.typeSelect(Association).
146         select(e|e.getEndTypes() == uml::Class)»
147
148         «EXPAND rule27 FOREACH
149         eAllContents.typeSelect(Association).
150         select(e|e.getEndTypes() == uml::Class)»
151
152         «EXPAND rule28 FOREACH
153         eAllContents.typeSelect(Class).
154         reject(e|Interaction.isInstance(e)|| Activity.isInstance(e))»
155
156         «EXPAND rule29 FOREACH
157         eAllContents.typeSelect(Association).
158         select(e|e.getEndTypes() == uml::Class)»
159
160         «EXPAND rule30 FOREACH eAllContents.typeSelect(Property)»
161
162         «EXPAND rule31 FOREACH eAllContents.typeSelect(Association)»
163         </TR><TR>
164         «REM»−−−−−Sequence Diagram Rules −−−−−−−−«ENDREM»
165
166         <TH Colspan=2 align= left>Sequence Diagram Rules</TH>
167
168         «EXPAND rule32 FOREACH eAllContents.typeSelect(Interaction)»
169
170         «EXPAND rule33 FOREACH eAllContents.typeSelect(Lifeline)»
171
172         «EXPAND rule34 FOREACH eAllContents.typeSelect(Message)»
173
174         «EXPAND rule35 FOREACH eAllContents.typeSelect(Lifeline)»
175
176         «EXPAND rule36 FOREACH eAllContents.typeSelect(Message)»
177         </TR>
178
179     </TABLE>
180     <BR><BR><BR>
181
182     <TABLE>
183     <TR>
184     <TD><B>Report By: Akhtar Ali Jalbani</B></TD>
185     </TR><TR><TD><i><a href=”Email: ajalbani@informatik.uni−geottingen.de”>Email:
186 ajalbani@informatik.uni−geottingen.de</a></i></TD>
187     </TR><TR><TD><i><a
```

```
188 href=”http://http://www.swe.informatik.uni−goettingen.de/”>
189 www.swe.informatik.uni−goettingen.de</a></i></TD>
190     </TR><TR><TD><i>Software Engineering and Distributed Systems Group</i></TD>
191     </TR><TR><TD><i>Institute of Computer Science, University of Goettingen</i></TD>
192     </TR><TR><TD><i>Goettingen, Germany</i></TD>
193
194     </TR>
195     </TABLE>
196
197 <script type=”text/javascript”>
198 <!−−
199 var d_names = new Array(”Sunday”, ”Monday”, ”Tuesday”,
200 ”Wednesday”, ”Thursday”, ”Friday”, ”Saturday”);
201
202 var m_names = new Array(”January”, ”February”, ”March”,
203 ”April”, ”May”, ”June”, ”July”, ”August”, ”September”,
204 ”October”, ”November”, ”December”);
205
206
207
208 var d = new Date();
209 var curr_day = d.getDay();
210 var curr_date = d.getDate();
211 var sup = ””;
212 if (curr_date == 1 || curr_date == 21 || curr_date ==31)
213     {
214     sup = ”st”;
215     }
216 else if (curr_date == 2 || curr_date == 22)
217     {
218     sup = ”nd”;
219     }
220 else if (curr_date == 3 || curr_date == 23)
221     {
222     sup = ”rd”;
223     }
224 else
225     {
226     sup = ”th”;
227     }
228 var curr_month = d.getMonth();
229 var curr_year = d.getFullYear();
230
231 var curr_hour = d.getHours();
232 var curr_min = d.getMinutes();
233
234
235 document.write(”Report generated on: ”+d_names[curr_day] + ” ” + curr_date + ”<SUP>”
236 + sup + ”</SUP> ” + m_names[curr_month] + ” ” + curr_year+ ” Time : ”+ curr_hour + ” : ” +
237 curr_min);
238
239 //−−>
240 </script>
241
242 </body>
243 </html>
244 «ENDFILE»
245 «ENDDEFINE»
246
```

```
247
248
249 «DEFINE rule1 FOR uml::UseCase»
250         «IF getQualifiedName()== null»
251           <TR>
252             <TD width=”60%><FONT FACE=”Geneva, Arial” SIZE=2>
253           ”Ri1: Each use case must be inside the subsystem”</TD>
254             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
255             </TD>
256             «ELSEIF isUseCase_InsideSubSystem() == false»
257               <TR>
258               <TD width=”60%><FONT FACE=”Geneva, Arial” SIZE=2>
259             ”Ri1: Each use case must be inside the subsystem”</TD>
260                 <TD>«qualifiedName»</TD>
261           «ENDIF»
262
263 «ENDDEFINE»
264
265 «DEFINE rule2 FOR uml::UseCase»
266
267         «IF getQualifiedName()== null»
268             <TR>
269             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
270           ”Ri2: Each use case must be associated with an actor”</TD>
271             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
272           </TD>
273             «ELSEIF isUseCase_Connected()==false»
274             <TR>
275             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
276           ”Ri2: Each use case must be associated with an actor”</TD>
277               <TD>«qualifiedName»</TD>
278           «ENDIF»
279 «ENDDEFINE»
280 «DEFINE rule3 FOR uml::UseCase»
281     «IF isUseCase_Inherited()==false»
282           <TR>
283             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
284           ”Ri3: The generalization between use case must not present in a use case diagram”</TD>
285               <TD>«qualifiedName»</TD>
286           «ENDIF»
287
288 «ENDDEFINE»
289
290 «DEFINE rule4 FOR uml::UseCase»
291
292         «IF getQualifiedName()== null»
293           <TR>
294             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
295           ”Ri4: Use case must be refined in a sequence diagram”</TD>
296             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
297           </TD>
298           «ELSEIF isUseCase_RefinedinSD()==false»
299               <TR>
300               <TD><FONT FACE=”Geneva, Arial” SIZE=2>
301           ”Ri4: Use case must be refined in a sequence diagram”</TD>
302               <TD>«qualifiedName»</TD>
303           «ENDIF»
304
305 «ENDDEFINE»
```

```
306 «DEFINE rule5 FOR uml::UseCase»
307         «IF getQualifiedName()== null»
308             <TR>
309             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
310           ”Ri5: Each use case should not be linked to more than three actors”</TD>
311             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
312           </TD>
313             «ELSEIF actorToUseCaseRatio()==false»
314               <TR>
315               <TD><FONT FACE=”Geneva, Arial” SIZE=2>
316           ”Ri5: Each use case should not be linked to more than three actors”</TD>
317               <TD>«qualifiedName»</TD>
318           «ENDIF»
319
320 «ENDDEFINE»
321 «DEFINE rule6 FOR uml::UseCase»
322         «IF getQualifiedName()== null»
323             <TR>
324             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
325           ”Ri6: Each usecase name should contain one to four words”</TD>
326             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
327           </TD>
328             «ELSEIF hasUseCaseNameLength()==false»
329               <TR>
330               <TD><FONT FACE=”Geneva, Arial” SIZE=2>
331           ”Ri6: Each usecase name should contain one to four words”</TD>
332               <TD>«qualifiedName»</TD>
333           «ENDIF»
334
335 «ENDDEFINE»
336 «DEFINE rule7 FOR uml::Package»
337
338         «IF getQualifiedName()== null»
339           <TR>
340               <TD><FONT FACE=”Geneva, Arial” SIZE=2>
341             ”Ri7: Each subsystem contain minimum 3 and
342               maximum 5 Use cases i.e UC= 3−5”</TD>
343             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
344           </TD>
345
346           «ELSEIF isSubsystemBig()==false»
347               <TR>
348               <TD><FONT FACE=”Geneva, Arial” SIZE=2>
349           ”Ri7: Each subsystem contain minimum 3 and
350           maximum 5 Use cases i.e UC= 3−5”</TD>
351               <TD>«qualifiedName»</TD>
352           «ENDIF»
353
354 «ENDDEFINE»
355 «DEFINE rule8 FOR uml::Package»
356       «IF getQualifiedName()== null»
357         <TR>
358             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
359           ”Ri8: Each subsystem name should start with
360           a capital letter and contains one to two words”</TD>
361             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
362           </TD>
363             «ELSEIF isSubsystemNameCapital()==false»
364               <TR>
```

```
365                 <TD><FONT FACE="Geneva, Arial" SIZE=2>
366             "Ri8: Each subsystem name should start with
367             capital letter and contains one to two words"</TD>
368                 <TD>«qualifiedName»</TD>
369             «ENDIF»
370
371 «ENDDEFINE»
372 «DEFINE rule9 FOR uml::Actor»
373
374             «IF getQualifiedName()== null»
375             <TR>
376             <TD><FONT FACE="Geneva, Arial" SIZE=2>
377           "Ri9: Each actor name should start with a capital letter"</TD>
378             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
379           </TD>
380
381             «ELSEIF isActorNameCapital()==false»
382               <TR>
383                 <TD><FONT FACE="Geneva, Arial" SIZE=2>
384             "Ri9: Each actor name should start with a capital letter"</TD>
385                 <TD>«qualifiedName»</TD>
386             «ENDIF»
387
388 «ENDDEFINE»
389 «DEFINE rule10 FOR uml::Actor»
390       «IF getQualifiedName()== null»
391             <TR>
392               <TD><FONT FACE="Geneva, Arial" SIZE=2>
393             "Ri10: The depth of generalization of an actor should not exceed one"</TD>
394               <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
395             </TD>
396               «ELSEIF isActorDit()==false»
397                 <TR>
398                 <TD><FONT FACE="Geneva, Arial" SIZE=2>
399             "Ri10: The depth of generalization of an actor should not exceed one"</TD>
400                 <TD>«qualifiedName»</TD>
401             «ENDIF»
402
403 «ENDDEFINE»
404 «DEFINE rule11 FOR uml::Model»
405       «IF getQualifiedName()== null»
406             <TR>
407             <TD><FONT FACE="Geneva, Arial" SIZE=2>
408           "Ri11: Each system name should start with a capital letter and
409           contain one to two words"</TD>
410             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
411           </TD>
412             «ELSEIF isSystemNameCapital()==false»
413                 <TR>
414               <TD><FONT FACE="Geneva, Arial" SIZE=2>
415     "Ri11: Each system name should start with a capital letter and
416           contain one to two words"</TD>
417             <TD>«qualifiedName»</TD>
418             «ENDIF»
419
420 «ENDDEFINE»
421 «DEFINE rule12 FOR uml::Model»
422     «IF getQualifiedName()== null»
423         <TR>
```

```
424             <TD><FONT FACE="Geneva, Arial" SIZE=2>
425           "Ri12: Actor must be outside the system"</TD>
426             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
427           </TD>
428           «ELSEIF isActorOutsideFromSystem()==true»
429           <TR>
430             <TD><FONT FACE="Geneva, Arial" SIZE=2>
431           "Ri12: Actor must be outside the system"</TD>
432             <TD>«qualifiedName»</TD>
433           «ENDIF»
434
435 «ENDDEFINE»
436 «DEFINE rule13 FOR uml::Model»
437         «IF getQualifiedName()== null»
438             <TR>
439             <TD><FONT FACE="Geneva, Arial" SIZE=2>
440           "Ri13: Use case diagram should not contain
441           more than 20 use cases"</TD>
442             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
443           </TD>
444             «ELSEIF isPackageBig()==false»
445               <TR>
446               <TD><FONT FACE="Geneva, Arial" SIZE=2>
447           "Ri13: Use case diagram should not contain
448           more than 20 use cases"</TD>
449               <TD>«qualifiedName»</TD>
450           «ENDIF»
451
452 «ENDDEFINE»
453 «DEFINE rule14 FOR uml::Include»
454           «IF isIncludeUseCaseDit()==false»
455             <TR>
456             <TD><FONT FACE="Geneva, Arial" SIZE=2>
457           "Ri14: The depth of Include use case should not
458           exceed one"</TD>
459             <TD>«qualifiedName»</TD>
460           «ENDIF»
461
462 «ENDDEFINE»
463 «DEFINE rule15 FOR uml::Extend»
464           «IF isExtendUseCaseDit()==false»
465             <TR>
466             <TD><FONT FACE="Geneva, Arial" SIZE=2>
467           "Ri15: The depth of extend use case should not
468           exceed to one"</TD>
469           <TD>«qualifiedName»</TD>
470           «ENDIF»
471
472 «ENDDEFINE»
473 «REM»----Rules for Activity Diagram-----«ENDREM»
474 «DEFINE rule16 FOR uml::Package»
475       «IF getQualifiedName()== null»
476           <TR>
477             <TD><FONT FACE="Geneva, Arial" SIZE=2>
478           "Ri16: Each subsystem of use case should be refined
479           by activity diagram"</TD>
480             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
481           </TD>
482             «ELSEIF isUseCaseRefinedByActivityDiagram()==false»
```

```
483           <TR>
484           <TD><FONT FACE="Geneva, Arial" SIZE=2>
485           "Ri16: Each subsystem of use case should be refined by
486           activity diagram"</TD>
487             <TD>«qualifiedName»</TD>
488           «ENDIF»
489
490 «ENDDEFINE»
491 «DEFINE rule17 FOR uml::CallBehaviorAction»
492       «IF getQualifiedName()== null»
493             <TR>
494             <TD><FONT FACE="Geneva, Arial" SIZE=2>
495           "Ri17: Each Activity in activity diagram should
496           refer to a usecase"</TD>
497             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
498           </TD>
499           «ELSEIF isActivityReferenceToUseCase()==false»
500             <TR>
501             <TD><FONT FACE="Geneva, Arial" SIZE=2>
502           "Ri17: Each Activity in activity diagram should
503           refer to a usecase"</TD>
504            <TD>«qualifiedName»</TD>
505           «ENDIF»
506
507 «ENDDEFINE»
508 «REM»----CLASS DIAGRAM RULES----«ENDREM»
509
510 «DEFINE rule18 FOR uml::Package»
511       «IF getQualifiedName()== null»
512           <TR>
513           <TD><FONT FACE="Geneva, Arial" SIZE=2>
514         "Ri18: Each subsystem should be
515         represented as a package in a class diagram"</TD>
516           <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
517         </TD>
518           «ELSEIF isUCSubSystemRepByCSubsystem()==false»
519             <TR>
520             <TD><FONT FACE="Geneva, Arial" SIZE=2>
521         "Ri18: Each subsystem of use case diagram should be
522         represented as a package in a class diagram"</TD>
523           <TD>«qualifiedName»</TD>
524           «ENDIF»
525 «ENDDEFINE»
526 «DEFINE rule19 FOR uml::Package»
527       «IF getQualifiedName()== null»
528           <TR>
529             <TD><FONT FACE="Geneva, Arial" SIZE=2>
530           "Ri19: Each package should not contain more than 20 classes"</TD>
531             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
532           </TD>
533           «ELSEIF hasClasses()==false»
534             <TR>
535             <TD><FONT FACE="Geneva, Arial" SIZE=2>
536             "Ri19: Each package should not contain more than 20 classes"</TD>
537             <TD>«qualifiedName»</TD>
538           «ENDIF»
539 «ENDDEFINE»
540 «DEFINE rule20 FOR uml::Class»
541           «IF getQualifiedName()== null»
```

```
542             <TR>
543             <TD><FONT FACE="Geneva, Arial" SIZE=2>
544           "Ri20: The DIT should not exceed 2"</TD>
545             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
546           </TD>
547             «ELSEIF isDit()==false»
548            <TR>
549            <TD><FONT FACE="Geneva, Arial" SIZE=2>
550           "Ri20: The DIT should not exceed to 2"</TD>
551            <TD>«qualifiedName»</TD>
552           «ENDIF»
553 «ENDDEFINE»
554 «DEFINE rule21 FOR uml::Class»
555       «IF getQualifiedName()== null»
556             <TR>
557             <TD><FONT FACE="Geneva, Arial" SIZE=2>
558           "Ri21: Multiple Inheritance must not exists"</TD>
559             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
560           </TD>
561             «ELSEIF isMultipleInheritance()==false»
562           <TR>
563           <TD><FONT FACE="Geneva, Arial" SIZE=2>
564           "Ri21: Multiple Inheritance must not exists"</TD>
565               <TD>«qualifiedName»</TD>
566           «ENDIF»
567
568 «ENDDEFINE»
569 «DEFINE rule22 FOR uml::Class»
570       «IF getQualifiedName()== null»
571          <TR>
572             <TD><FONT FACE="Geneva, Arial" SIZE=2>
573           "Ri22: Each class name should start with a capital letter
574           and should be one word."</TD>
575             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
576           </TD>
577           «ELSEIF isClassNameCapital()==false»
578            <TR>
579               <TD><FONT FACE="Geneva, Arial" SIZE=2>
580           "Ri22 Each Class name should start with a capital letter
581            and should be one word."</TD>
582             <TD>«qualifiedName»</TD>
583           «ENDIF»
584
585 «ENDDEFINE»
586 «DEFINE rule23 FOR uml::Class»
587       «IF getQualifiedName()== null»
588           <TR>
589             <TD><FONT FACE="Geneva, Arial" SIZE=2>
590           "Ri23: Entity class should contain at least 3 attributes"</TD>
591             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
592           </TD>
593             «ELSEIF isEntityClassValid()==false»
594             <TR>
595              <TD><FONT FACE="Geneva, Arial" SIZE=2>
596           "Ri23: Entity class should contain at least 3 attributes"</TD>
597             <TD>«qualifiedName»</TD>
598           «ENDIF»
599
600 «ENDDEFINE»
```

```
601 «DEFINE rule24 FOR uml::Class»
602       «IF getQualifiedName()== null»
603             <TR>
604             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
605             ”Ri24: Control class should contain 2−5 Operations”</TD>
606             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
607           </TD>
608             «ELSEIF isControlClassValid()==false»
609           <TR>
610           <TD><FONT FACE=”Geneva, Arial” SIZE=2>
611           ”Ri24: Control class should contain 2−5 Operations”</TD>
612             <TD>«qualifiedName»</TD>
613           «ENDIF»
614 «ENDDEFINE»
615 «DEFINE rule25 FOR uml::Class»
616     «IF getQualifiedName()== null»
617       <TR>
618             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
619           ”Ri25: If class is empty class than class must be
620            a boundary class”</TD>
621             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
622           </TD>
623             «ELSEIF isEmptyClassValid()==false»
624             <TR>
625             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
626           ”Ri25: If class is empty class than class must be
627            a boundary class”</TD>
628             <TD>«qualifiedName»</TD>
629           «ENDIF»
630 «ENDDEFINE»
631 «DEFINE rule26 FOR uml::Association»
632       «LET isAssociationhasName() AS hasName»
633       «IF hasName==false»
634          <TR>
635             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
636           ”Ri26: Each association must have name”</TD>
637             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
638           </TD>
639           «ENDIF»
640       «ENDLET»
641 «ENDDEFINE»
642 «DEFINE rule27 FOR uml::Association»
643       «IF isMultiplicityValue()==false»
644         <TR>
645           <TD><FONT FACE=”Geneva, Arial” SIZE=2>
646           ”Ri27: Each association must specify multiplicity
647           values at both ends.”</TD>
648             <TD>«isMultiplicityValue()»</TD>
649           «ENDIF»
650 «ENDDEFINE»
651
652 «DEFINE rule28 FOR uml::Class»
653     «LET isAssociationPerClass() AS apc»
654       «IF apc==false»
655         <TR>
656             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
657           ”Ri28: Each class have (1−5) associations ”</TD>
658             <TD>«qualifiedName»</TD>
659           «ENDIF»
```

```
660       «ENDLET»
661 «ENDDEFINE»
662 «DEFINE rule29 FOR uml::Association»
663     «LET isAssociationNameLower() AS lowerCaseLetter »
664       «IF qualifiedName==null»
665           <TR>
666             <TD><FONT FACE="Geneva, Arial" SIZE=2>
667             "Ri29: Each association must have name"</TD>
668             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
669           </TD>
670       «ELSEIF lowerCaseLetter==false»
671             <TR>
672             <TD><FONT FACE="Geneva, Arial" SIZE=2>
673           "Ri29: Each association must have name"
674           </TD>
675             <TD>«qualifiedName»</TD>
676           «ENDIF»
677       «ENDLET»
678 «ENDDEFINE»
679 «DEFINE rule30 FOR uml::Property»
680
681       «IF hasAggregationOrComposition()==false»
682       <TR>
683           <TD><FONT FACE="Geneva, Arial" SIZE=2>
684           "Ri30: Classes should not be linked with composition
685           or aggregation type of association"</TD>
686               <TD>«this.association.getEndTypes().typeSelect(Class).
687           getQualifiedName()»</TD>
688           «ENDIF»
689 «ENDDEFINE»
690 «DEFINE rule31 FOR uml::Association»
691       «IF qualifiedName==null»
692           <TR>
693             <TD><FONT FACE="Geneva, Arial" SIZE=2>
694           "R31: The link to classes belonging to another
695           package must be uni-directional"</TD>
696             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
697           </TD>
698       «ELSEIF isClassUniDirectional()==false»
699           <TR>
700               <TD><FONT FACE="Geneva, Arial" SIZE=2>
701           "Ri31 The link to classes belonging to another
702           package must be uni-directional"</TD>
703               <TD>«qualifiedName»</TD>
704           «ENDIF»
705 «ENDDEFINE»
706 «REM»---Sequence Diagram Rules ----«ENDREM»
707 «DEFINE rule32 FOR uml::Interaction»
708       «IF isActorInSequenceDiagram()==false»
709         <TR>
710           <TD><FONT FACE="Geneva, Arial" SIZE=2>
711           "i32: Each Sequence diagram have at least one
712           actor on lifeline"</TD>
713             <TD>«qualifiedName»</TD>
714           «ENDIF»
715 «ENDDEFINE»
716 «DEFINE rule33 FOR uml::Lifeline»
717       «IF isObjectReferesToClass()==false»
718           <TR>
```

```
719                 <TD><FONT FACE="Geneva, Arial" SIZE=2>
720             "Ri33: Each object in sequence diagram must have
721             corresponding class in class diagram"</TD>
722                 <TD>«this.represents.type.qualifiedName»</TD>
723             «ENDIF»
724 «ENDDEFINE»
725 «DEFINE rule34 FOR uml::Message»
726       «IF getQualifiedName()== null»
727             <TR>
728               <TD><FONT FACE="Geneva, Arial" SIZE=2>
729             "Ri34: Every call message received by the lifeline
730             must have corresponding method in class diagram"</TD>
731               <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
732             </TD>
733             «ELSEIF isMessageReferesToOperation()==false»
734             <TR>
735             <TD><FONT FACE="Geneva, Arial" SIZE=2>
736             "Ri34: Every call message received by the lifeline
737             should have corresponding method in class diagram"</TD>
738               <TD>«qualifiedName»</TD>
739             «ENDIF»
740 «ENDDEFINE»
741 «DEFINE rule35 FOR uml::Lifeline»
742       «IF hasMessageCallRelationToClassAssocaition()==false»
743         <TR>
744               <TD><FONT FACE="Geneva, Arial" SIZE=2>
745             "Ri35: If there is a message call between two lifeline than
746             there must be an association between corresponding classes"
747             </TD> <TD>«qualifiedName»</TD>
748             «ENDIF»
749 «ENDDEFINE»
750 «DEFINE rule36 FOR uml::Message»
751       «IF isMessageLabeled()==false»
752           <TR>
753                 <TD><FONT FACE="Geneva, Arial" SIZE=2>
754             "Ri36: Each message must be labeled"</TD>
755               <TD>«qualifiedName»</TD>
756             «ENDIF»
757 «ENDDEFINE»
```

Listing G.2: Xpand Report for Incomplete Model

Appendix H

Html Report Generation in Xpand for Complete Model

```
1  «REM»
2  This file contains html report for the complete models at design phase.
3  Author: Akhtar Ali Jalbani
4  Date: 18−09−2010
5  «ENDREM»
6  «IMPORT uml»
7  «EXTENSION Rules::designModel»
8  «EXTENSION Rules::analysisModel»
9
10 «DEFINE designModel FOR Model»
11 «FILE "blueGroupDesignModel.html"»
12 <HTML>
13 <HEAD>
14 </HEAD>
15 <Title>Report for Complete Models at design Phase for Bakery System.....<Title>
16 <BODY>
17     <H1><a name="PM">Report for Design model:
18 (Bakery System) Group: Blue− UML Course 2010</a></H1>
19     <TABLE BORDER="1">
20         <TR>
21           <th >Rule</th>
22           <th >Qualified Name</th>
23         </TR><TR>
24        «REM»Class Rules  «ENDREM»
25        <TH Colspan=2 align= left>Class Rules</TH>
26         «EXPAND rule1 FOREACH
27          eAllContents.typeSelect(uml::Class).
28         reject(e|Interaction.isInstance(e)|| StateMachine.isInstance(e) ||
29         Interface.isInstance(e) || Activity.isInstance(e))»
30         «EXPAND rule2 FOREACH
31         eAllContents.typeSelect(uml::Class).
32         reject(e|Interaction.isInstance(e)||
33         StateMachine.isInstance(e)|| Activity.isInstance(e))»
34         «EXPAND rule3 FOREACH
```

```
35          eAllContents.typeSelect(uml::Class).reject(e|Interaction.isInstance(e)||
36          StateMachine.isInstance(e)|| Activity.isInstance(e))»
37          «EXPAND rule4 FOREACH
38          eAllContents.typeSelect(uml::Class).reject(e|Interaction.isInstance(e)||
39          StateMachine.isInstance(e)|| Activity.isInstance(e))»
40          «EXPAND rule5 FOREACH
41          eAllContents.typeSelect(uml::Class).reject(e|Interaction.isInstance(e)||
42          StateMachine.isInstance(e)|| Activity.isInstance(e))»
43          «EXPAND rule6 FOREACH
44          eAllContents.typeSelect(uml::Class).reject(e|Interaction.isInstance(e)||
45          StateMachine.isInstance(e)|| Activity.isInstance(e))»  </TR><TR>
46          <TH Colspan=3 align= left>Class Association Rules</TH>
47          «EXPAND rule7 FOREACH eAllContents.typeSelect(Association)»
48          «EXPAND rule8 FOREACH
49           eAllContents.typeSelect(Class).reject(e|Interaction.isInstance(e)||
50          Activity.isInstance(e))»
51          «EXPAND rule9 FOREACH
52          eAllContents.typeSelect(Association).
53          select(e|e.getEndTypes() == uml::Class)»
54          «EXPAND rule10 FOREACH
55          eAllContents.typeSelect(Association).select(e|e.getEndTypes() == uml::Class)»
56        «EXPAND rule11 FOREACH
57        eAllContents.typeSelect(Association).select(e|e.getEndTypes() == uml::Class)»
58         </TR><TR>
59          <TH Colspan=3 align= left>Class Package Rules</TH>
60          «EXPAND rule12 FOREACH eAllContents.typeSelect(Model)»
61          «EXPAND rule13 FOREACH
62          eAllContents.typeSelect(Package).reject(e|
63          e.qualifiedName==”Data::Bäckerei” ||
64          e.qualifiedName == ”Data::Bäckerei::Personalverwaltung” ||
65          e.qualifiedName==”Data::Bäckerei::Backverwaltung” ||
66          e.qualifiedName==”Data::Bäckerei::Lagerverwaltung”||
67          e.qualifiedName==”Data::Bäckerei::Verkaufssystem”)»
68          «EXPAND rule14 FOREACH
69        eAllContents.typeSelect(Association).select(e|e.getEndTypes() == uml::Class)»
70         «EXPAND rule15 FOREACH
71        eAllContents.typeSelect(Package).reject(e|
72        e.qualifiedName==”Data::Bäckerei” ||
73        e.qualifiedName == ”Data::Bäckerei::Personalverwaltung” ||
74        e.qualifiedName==”Data::Bäckerei::Backverwaltung” ||
75        e.qualifiedName==”Data::Bäckerei::Lagerverwaltung”||
76        e.qualifiedName==”Data::Bäckerei::Verkaufssystem”)»
77          </TR> <TR>
78          <TH Colspan=3 align= left>Class Attribute / Property Rules</TH>
79          «EXPAND rule16 FOREACH eAllContents.typeSelect(Property).reject(e|e.association)»
80          «EXPAND rule17 FOREACH eAllContents.typeSelect(Property).reject(e|e.association)»
81          </TR><TR>
82          <TH Colspan=3 align= left>Class Operation and Operation Parameter Rules</TH>
83          «EXPAND rule18 FOREACH eAllContents.typeSelect(Operation)»
84          «EXPAND rule19 FOREACH
85          eAllContents.typeSelect(Class).select(e|e.getAppliedStereotypes().name ==
86          ’entity’)»
87          «EXPAND rule20 FOREACH eAllContents.typeSelect(Operation)»
88          «EXPAND rule21 FOREACH eAllContents.typeSelect(Operation)»
89          «EXPAND rule22 FOREACH
90          eAllContents.typeSelect(Parameter).reject(e|Association.isInstance(e))»
91          </TR> <TR>
92          «REM»−−−−−Sequence Diagram Rules −−−−−−−−«ENDREM»
93          <TH Colspan=3 align= left>Sequence Diagram Rules</TH>
```

```
94          «EXPAND rule23 FOREACH
95        eAllContents.typeSelect(Interaction).reject(e|
96        e.name==”com_Verkauf_Reste_an_Tafel_geben”)»
97          «EXPAND rule24 FOREACH eAllContents.typeSelect(Lifeline)»
98          «EXPAND rule25 FOREACH eAllContents.typeSelect(Message)»
99          «EXPAND rule26 FOREACH eAllContents.typeSelect(Lifeline)»
100         «EXPAND rule27 FOREACH eAllContents.typeSelect(Message)»
101         </TR> «REM»−−−Activity Diagram−−−«REM» <TR>
102         <TH Colspan=3 align= left>Activity Diagram Rules</TH>
103         «EXPAND rule28 FOREACH eAllContents.typeSelect(Activity)»
104         «EXPAND rule29 FOREACH eAllContents.typeSelect(Activity)»
105         «EXPAND rule30 FOREACH eAllContents.typeSelect(Activity)»
106         «EXPAND rule31 FOREACH eAllContents.typeSelect(Activity)»
107         «EXPAND rule32 FOREACH eAllContents.typeSelect(Action)»
108         «EXPAND rule33 FOREACH eAllContents.typeSelect(Action)»
109         «EXPAND rule34 FOREACH eAllContents.typeSelect(CentralBufferNode)»
110         </TR><TR>
111         <TH Colspan=3 align= left>State Machine Diagram Rules</TH>
112         «EXPAND rule35 FOREACH eAllContents.typeSelect(State)»
113         «EXPAND rule36 FOREACH eAllContents.typeSelect(State)»
114         «EXPAND rule37 FOREACH eAllContents.typeSelect(StateMachine)»
115         </TR>
116 </TABLE>
117 <BR><BR><BR>
118 <TABLE>
119     <TR>
120     <TD><B>Report By: Akhtar Ali Jalbani</B></TD>
121     </TR><TR><TD><i><a href=”Email: ajalbani@informatik.uni−geottingen.de”>Email:
122 ajalbani@informatik.uni−geottingen.de</a></i></TD>
123     </TR><TR><TD><i><a
124 href=”http://http://www.swe.informatik.uni−goettingen.de/”>
125 www.swe.informatik.uni−goettingen.de</a></i></TD>
126     </TR><TR><TD><i>Software Engineering and Distributed Systems Group</i></TD>
127     </TR><TR><TD><i>Institute of Computer Science, University of Goettingen</i></TD>
128     </TR><TR><TD><i>Goettingen, Germany</i></TD>
129
130     </TR>
131 </TABLE>
132
133 <script type=”text/javascript”>
134 <!−−
135 var d_names = new Array(”Sunday”, ”Monday”, ”Tuesday”,
136 ”Wednesday”, ”Thursday”, ”Friday”, ”Saturday”);
137 var m_names = new Array(”January”, ”February”, ”March”,
138 ”April”, ”May”, ”June”, ”July”, ”August”, ”September”,
139 ”October”, ”November”, ”December”);
140 var d = new Date();
141 var curr_day = d.getDay();
142 var curr_date = d.getDate();
143 var sup = ””;
144 if (curr_date == 1 || curr_date == 21 || curr_date ==31)
145     {
146     sup = ”st”;
147     }
148 else if (curr_date == 2 || curr_date == 22)
149     {
150     sup = ”nd”;
151     }
152 else if (curr_date == 3 || curr_date == 23)
```

```
153     {
154     sup = ”rd”;
155     }
156 else
157     {
158     sup = ”th”;
159     }
160 var curr_month = d.getMonth();
161 var curr_year = d.getFullYear();
162
163 var curr_hour = d.getHours();
164 var curr_min = d.getMinutes();
165
166
167 document.write(”Report generated on: ”+d_names[curr_day] + ” ” + curr_date + ”<SUP>”
168 + sup + ”</SUP> ” + m_names[curr_month] + ” ” + curr_year+ ” Time : ”+ curr_hour + ” : ” +
169 curr_min);
170
171 //−−>
172 </script>
173
174 </body>
175 </html>
176 «ENDFILE»
177 «ENDDEFINE»
178
179
180 «REM»−−−CLASS DIAGRAM RULES−−−«ENDREM»
181
182 «DEFINE rule1 FOR uml::Class»
183
184       «LET isClasshasAttributes() AS classhasAttributes»
185             «IF qualifiedName==null»
186               <TR>
187               <TD><FONT FACE=”Geneva, Arial” SIZE=2>
188             ”Rc1: Each Class should have attributes”</TD>
189               <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
190             </TD></TR>
191             «ELSEIF classhasAttributes==false»
192           <TR>
193             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
194           ”Rc1: Each Class should have attributes”</TD>
195             <TD>«qualifiedName»</TD>
196             </TR>
197           «ENDIF»
198     «ENDLET»
199 «ENDDEFINE»
200
201 «DEFINE rule2 FOR uml::Class»
202
203       «LET isClasshasOperations() AS classhasOperations»
204           «IF qualifiedName== null»
205             <TR>
206             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
207           ”Rc2: Each class should have operations”</TD>
208             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
209           </TD></TR>
210           «ELSEIF classhasOperations==false»
211             <TR>
```

```
212             <TD><FONT FACE="Geneva, Arial" SIZE=2>
213           "Rc2: Each Class should have Operations"</TD>
214             <TD>«qualifiedName»</TD></TR>
215         «ENDIF»
216
217     «ENDLET»
218 «ENDDEFINE»
219
220 «DEFINE rule3 FOR uml::Class»
221     «LET isClassDit() AS dit»
222           «IF dit==false»
223             <TR>
224           <TD><FONT FACE="Geneva, Arial" SIZE=2>
225           "Rc3: Depth of Inheritance Should be less than 4"</TD>
226             <TD>«qualifiedName»</TD></TR>
227         «ENDIF»
228     «ENDLET»
229 «ENDDEFINE»
230
231 «DEFINE rule4 FOR uml::Class»
232     «LET isMultipleInheritance() AS ismultipileInheriatnce»
233         «IF ismultipileInheriatnce==false»
234             <TR>
235           <TD><FONT FACE="Geneva, Arial" SIZE=2>
236           "Rc4: Multiple Inheritance must not exists"</TD>
237             <TD>«getQualifiedName()»</TD></TR>
238         «ENDIF»
239       «ENDLET»
240 «ENDDEFINE»
241
242 «DEFINE rule5 FOR uml::Class»
243       «LET isClassNameCapital() AS capitalName»
244
245           «IF capitalName==false»
246             <TR>
247           <TD><FONT FACE="Geneva, Arial" SIZE=2>
248           "Rc5: Each class name should start with a capital letter
249           and should be one word"</TD>
250             <TD>«getQualifiedName()»</TD></TR>
251
252         «ENDIF»
253       «ENDLET»
254 «ENDDEFINE»
255
256 «DEFINE rule6 FOR uml::Class»
257     «LET hasToomanyOperations() AS maxOperation»
258           «IF getQualifiedName()== null»
259           <TR>
260           <TD><FONT FACE="Geneva, Arial" SIZE=2>
261           "Rc6: Each class should have maximum 10 operations"</TD>
262             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
263           </TD></TR>
264           «ELSEIF maxOperation==false»
265           <TR>
266           <TD><FONT FACE="Geneva, Arial" SIZE=2>
267           "Rc6: Each class should have maximum 10 operations"</TD>
268             <TD>«getQualifiedName()»</TD></TR>
269         «ENDIF»
270       «ENDLET»
```

```
271 «ENDDEFINE»
272
273
274 «REM»−−−Association Rules −−−«ENDREM»
275
276 «DEFINE rule7 FOR uml::Association»
277       «LET isAssociationhasName() AS hasName»
278       «IF hasName==false»
279          <TR>
280             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
281           ”Rc7 Each association must have name”</TD>
282             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
283         </TD></TR>
284           «ENDIF»
285       «ENDLET»
286 «ENDDEFINE»
287
288 «DEFINE rule8 FOR uml::Class»
289     «LET isAssociationPerClass() AS apc»
290       «IF apc==false»
291         <TR>
292             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
293           ”Rc8: Each class has association (1−5)”</TD>
294             <TD>«qualifiedName»</TD></TR>
295           «ENDIF»
296       «ENDLET»
297 «ENDDEFINE»
298
299 «DEFINE rule9 FOR uml::Association»
300     «LET isAssociationNameLower() AS lowerCaseLetter »
301       «IF qualifiedName==null»
302           <TR>
303             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
304           ”Rc9: Each association must have name”</TD>
305             <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
306           </TD></TR>
307       «ELSEIF lowerCaseLetter==false»
308             <TR>
309             <TD><FONT FACE=”Geneva, Arial” SIZE=2>
310           ”Rc9: Each association name should be in lower case letter”</TD>
311             <TD>«qualifiedName»</TD></TR>
312           «ENDIF»
313       «ENDLET»
314 «ENDDEFINE»
315
316 «DEFINE rule10 FOR uml::Association»
317   «LET isDirectedAssociation() AS hasDirectionalAssociation»
318
319         «IF getQualifiedName()== null»
320         <TR>
321         <TD><FONT FACE=”Geneva, Arial” SIZE=2>
322         ”Rc10: Each association must have direction”</TD>
323         <TD><FONT Color = RED FACE=”Geneva, Arial” SIZE=2> No Name</FONT>
324         </TD></TR>
325           «ELSEIF hasDirectionalAssociation==false»
326             <TR>
327           <TD><FONT FACE=”Geneva, Arial” SIZE=2>
328           ”Rc10: Each association must have direction”</TD>
329               <TD>«getQualifiedName()»</TD></TR>
```

```
330         «ENDIF»
331   «ENDLET»
332 «ENDDEFINE»
333
334 «DEFINE rule11 FOR uml::Association»
335       «LET hasMultiplicityDefined() AS mulValue»
336         «IF getQualifiedName()== null»
337         <TR>
338         <TD><FONT FACE="Geneva, Arial" SIZE=2>
339         "Rc11: Each association must specify multiplicity and it
340         must be n to 1"</TD>
341           <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
342           </TD></TR>
343           «ELSEIF mulValue==false»
344         <TR>
345         <TD><FONT FACE="Geneva, Arial" SIZE=2>
346           "Rc11: Each Association must specify multiplicity and it
347             must be n to 1"</TD>
348             <TD>«getQualifiedName()»</TD></TR>
349         «ENDIF»
350     «ENDLET»
351 «ENDDEFINE»
352
353 «REM»−−−Package Rules−−«ENDREM»
354
355 «DEFINE rule12 FOR uml::Model»
356       «LET hasAssociationClass() AS associationclass»
357         «IF associationclass==false»
358         <TR>
359         <TD><FONT FACE="Geneva, Arial" SIZE=2>
360         "Rc12: Association class must not exists in design model"</TD>
361           <TD>«getQualifiedName()»</TD></TR>
362         «ENDIF»
363     «ENDLET»
364 «ENDDEFINE»
365
366 «DEFINE rule13 FOR uml::Package»
367     «LET hasTooManyClasses() AS tooManyClasses»
368         «IF tooManyClasses==false»
369       <TR>
370       <TD><FONT FACE="Geneva, Arial" SIZE=2>
371       "Rc13: Each package should have maximum 20 classes."</TD>
372         <TD>«getQualifiedName()»</TD></TR>
373         «ENDIF»
374
375     «ENDLET»
376 «ENDDEFINE»
377
378 «DEFINE rule14 FOR uml::Association»
379       «IF qualifiedName==null»
380           <TR>
381             <TD><FONT FACE="Geneva, Arial" SIZE=2>
382           "Rc14: The link to classes belonging to another
383           package must be uni−directional"</TD>
384             <TD><FONT Color = RED FACE="Geneva, Arial" SIZE=2> No Name</FONT>
385         </TD></TR>
386       «ELSEIF isClassUniDirectional()==false»
387           <TR>
388               <TD><FONT FACE="Geneva, Arial" SIZE=2>
```

```
389             "Rc14: The link to classes belonging to another
390             package must be uni−directional"</TD>
391                 <TD>«qualifiedName»</TD> </TR>
392             «ENDIF»
393 «ENDDEFINE»
394
395 «DEFINE rule15 FOR uml::Package»
396
397     «LET ispackageNestingLvel() AS nestingLevel»
398       «IF nestingLevel==true»
399             <TR>
400           <TD><FONT FACE="Geneva, Arial" SIZE=2>
401           "Rc15: Package nesting level should be maximum 2."</TD>
402               <TD>«getQualifiedName()»</TD></TR>
403           «ENDIF»
404       «ENDLET»
405 «ENDDEFINE»
406
407
408 «REM»Class Attribute / Property rules «ENDREM»
409
410 «DEFINE rule16 FOR uml::Property»
411     «LET isAttributeDataType() AS attributeDataType»
412        «IF attributeDataType==false»
413             <TR>
414             <TD><FONT FACE="Geneva, Arial" SIZE=2>
415             "Rc16: Each attribute must have data type and
416               should be private."</TD>
417               <TD>«getQualifiedName()»</TD></TR>
418          «ENDIF»
419     «ENDLET»
420 «ENDDEFINE»
421
422 «DEFINE rule17 FOR uml::Property»
423     «LET isCompositionRelationship() AS multiplicityValue»
424       «IF multiplicityValue==false»
425             <TR>
426             <TD><FONT FACE="Geneva, Arial" SIZE=2>
427             "Rc17: If Class has composition relationship than
428             multiplicity must be 1."</TD>
429               <TD>«this.owningAssociation.name»</TD></TR>
430          «ENDIF»
431     «ENDLET»
432 «ENDDEFINE»
433
434 «REM»===== Class Operation and parameter rules === == = =====«ENDREM»
435
436 «DEFINE rule18 FOR uml::Operation»
437     «LET hasParameters() AS hasParameters»
438       «IF hasParameters==false && this.qualifiedName != null»
439         <TR>
440       <TD><FONT FACE="Geneva, Arial" SIZE=2>
441       "Rc18: Each operation should have maximum four parameters."</TD>
442       <TD>«getQualifiedName()»</TD></TR>
443         «ENDIF»
444     «ENDLET»
445 «ENDDEFINE»
446 «DEFINE rule19 FOR uml::Class»
447     «LET hasGettersAndSetters() AS hasgetSet»
```

```
448       «IF hasgetSet==false && this.qualifiedName != null»
449           <TR>
450         <TD><FONT FACE=”Geneva, Arial” SIZE=2>
451         ”Rc19: Entity class should have getters and setters.”</TD>
452           <TD>«qualifiedName»</TD></TR>
453         «ENDIF»
454       «ENDLET»
455 «ENDDEFINE»
456 «DEFINE rule20 FOR uml::Operation»
457     «LET isAbsractOperationInAbsractClass() AS hasAbstractOp»
458       «IF hasAbstractOp==false && this.qualifiedName != null»
459           <TR>
460         <TD><FONT FACE=”Geneva, Arial” SIZE=2>
461         ”Rc20: Abstract class should have abstract operations.”</TD>
462           <TD>«qualifiedName»</TD></TR>
463         «ENDIF»
464       «ENDLET»
465 «ENDDEFINE»
466 «DEFINE rule21 FOR uml::Operation»
467     «LET hasReturnType() AS hasType»
468       «IF hasType==false && this.qualifiedName != null»
469         <TR>
470         <TD><FONT FACE=”Geneva, Arial” SIZE=2>
471         ”Rc21: Each operation must have return type.”</TD>
472           <TD>«getQualifiedName()»</TD></TR>
473         «ENDIF»
474       «ENDLET»
475 «ENDDEFINE»
476 «DEFINE rule22 FOR uml::Parameter»
477     «LET hasParameterType() AS hasType»
478         «IF hasType==false && this.qualifiedName != null»
479           <TR>
480         <TD><FONT FACE=”Geneva, Arial” SIZE=2>
481         ”Rc22: Each parameter should have data type.”</TD>
482           <TD>«qualifiedName»</TD></TR>
483         «ENDIF»
484       «ENDLET»
485 «ENDDEFINE»
486 «REM»————————Sequence Diagram Rules ————————«ENDREM»
487 «DEFINE rule23 FOR uml::Interaction»
488         «IF isActorInSequenceDiagram()==false»
489           <TR>
490           <TD><FONT FACE=”Geneva, Arial” SIZE=2>
491           ”Rc23: Each Sequence diagram have at least one actor on lifeline”
492           </TD> <TD>«qualifiedName»</TD></TR>
493           «ENDIF»
494 «ENDDEFINE»
495 «DEFINE rule24 FOR uml::Lifeline»
496       «IF isObjectReferesToClass()==false»
497           <TR>
498               <TD><FONT FACE=”Geneva, Arial” SIZE=2>
499           ”Rc24: Each object in sequence diagram must have
500           corresponding class in class diagram”</TD>
501               <TD>«this.represents.type»</TD> </TR>
502           «ENDIF»
503
504 «ENDDEFINE»
505 «DEFINE rule25 FOR uml::Message»
506       «IF isMessageReferesToOperation()==false»
```

```
507         «IF qualifiedName==null»
508           <TR>
509           <TD><FONT FACE="Geneva, Arial" SIZE=2>
510           "Rc25: Every call message received by the lifeline should have
511           corresponding method in class diagram"</TD>
512             <TD><Font Color=red>No Name</TD></TR>
513           «ELSE»
514             <TR>
515           <TD><FONT FACE="Geneva, Arial" SIZE=2>
516           "Rc25: Every call message received by the lifeline should have
517           corresponding method in class diagram"</TD>
518             <TD>«qualifiedName»</TD></TR>
519           «ENDIF»
520         «ENDIF»
521 «ENDDEFINE»
522
523 «DEFINE rule26 FOR uml::Lifeline»
524         «IF hasMessageCallRelationToClassAssocaition()==false»
525         <TR>
526             <TD><FONT FACE="Geneva, Arial" SIZE=2>
527           "Rc26: If there is a message call between lifeline
528            than there must be an association between corresponding classes"</TD>
529             <TD>«qualifiedName»</TD></TR>
530           «ENDIF»
531 «ENDDEFINE»
532
533
534 «DEFINE rule27 FOR uml::Message»
535
536     «LET isReturnMessage() AS returnMessage»
537       «IF returnMessage==false»
538       <TR>
539       <TD><FONT FACE="Geneva, Arial" SIZE=2>
540       "Rc27: If Message is Empty than it must be a return type message"</TD>
541         <TD>«qualifiedName»</TD></TR>
542         «ENDIF»
543       «ENDLET»
544 «ENDDEFINE»
545
546
547 «REM»Activity Diagram Rules «ENDREM»
548
549 «DEFINE rule28 FOR uml::Activity»
550     «LET hasReferenceClass() AS msgOpRelation»
551         «IF msgOpRelation==false»
552           <TR>
553         <TD><FONT FACE="Geneva, Arial" SIZE=2>
554         "Rc28: One Activity diagram should reference to one class operation."</TD>
555         <TD>«qualifiedName»</TD></TR>
556
557       «ENDIF»
558   «ENDLET»
559 «ENDDEFINE»
560
561 «DEFINE rule29 FOR uml::Activity»
562     «LET hasTooManyDecisionPoints() AS msgOpRelation»
563         «IF msgOpRelation==false»
564         <TR>
565       <TD><FONT FACE="Geneva, Arial" SIZE=2>
```

```
566         "Rc29: The maximum decision point should be 12 in activity diagram. "</TD>
567         <TD>«qualifiedName»</TD></TR>
568           «ENDIF»
569     «ENDLET»
570 «ENDDEFINE»
571
572 «DEFINE rule30 FOR uml::Activity»
573       «LET hasTooManySwimlanes() AS msgOpRelation»
574           «IF msgOpRelation==false»
575           <TR>
576         <TD><FONT FACE="Geneva, Arial" SIZE=2>
577         "Rc30: Each activity diagram should contain 0 to 3 swim lane."</TD>
578         <TD>«qualifiedName»</TD></TR>
579           «ENDIF»
580       «ENDLET»
581 «ENDDEFINE»
582
583 «DEFINE rule31 FOR uml::Activity»
584       «LET hasTooManyIntialAndExitNodes() AS msgOpRelation»
585           «IF msgOpRelation==false»
586         <TR>
587         <TD><FONT FACE="Geneva, Arial" SIZE=2>
588         "Rc31: Each activity diagram should contain one initial
589         node and one exit node."</TD>
590         <TD>«qualifiedName»</TD></TR>
591           «ENDIF»
592     «ENDLET»
593 «ENDDEFINE»
594
595 «DEFINE rule32 FOR uml::Action»
596       «LET hasActivityReferenceToClass() AS activityRefClass»
597           «IF activityRefClass==false»
598             «IF qualifiedName==null»
599             <TR>
600           <TD><FONT FACE="Geneva, Arial" SIZE=2>
601           "Rc32: Activity in activity diagram could reference to class operations."</TD>
602           <TD><Font Color=red>No Name</TD></TR>
603             «ELSE»
604             <TR>
605           <TD><FONT FACE="Geneva, Arial" SIZE=2>
606           "Rc32: Activity in activity diagram could reference to class operations."</TD>
607           <TD>«qualifiedName»</TD></TR>
608           «ENDIF»
609         «ENDIF»
610       «ENDLET»
611 «ENDDEFINE»
612
613 «DEFINE rule33 FOR uml::Action»
614         «LET isDeadActivity() AS deadActivity»
615             «IF deadActivity==false»
616               «IF qualifiedName==null»
617               <TR>
618             <TD><FONT FACE="Geneva, Arial" SIZE=2>
619             "Rc33: Dead activity must not present in activity diagram."</TD>
620             <TD><Font Color=red>No Name</TD></TR>
621             «ELSE»
622           <TR>
623           <TD><FONT FACE="Geneva, Arial" SIZE=2>
624           "Rc33: Dead activity must not present in activity diagram."</TD>
```

```
625           <TD>«qualifiedName»</TD></TR>
626         «ENDIF»
627       «ENDIF»
628         «ENDLET»
629 «ENDDEFINE»
630
631 «DEFINE rule34 FOR uml::CentralBufferNode»
632       «LET hasObjectReferenceToClass() AS objectRefClass»
633           «IF objectRefClass==false»
634         <TR>
635         <TD><FONT FACE="Geneva, Arial" SIZE=2>
636         "Rc34: Objects of activity diagram should corresponds
637         to the class in class diagram."</TD>
638           <TD>«this.activity.qualifiedName»</TD></TR>
639           «ENDIF»
640         «ENDLET»
641 «ENDDEFINE»
642
643
644 «DEFINE rule35 FOR uml::State»
645       «LET isDeadState() AS deadstate»
646             «IF deadstate==false»
647
648               <TR>
649             <TD><FONT FACE="Geneva, Arial" SIZE=2>
650             "Rc35: Dead state must not be present in a state machine."</TD>
651               <TD>«qualifiedName»</TD></TR>
652
653           «ENDIF»
654       «ENDLET»
655 «ENDDEFINE»
656 «DEFINE rule36 FOR uml::State»
657       «LET isStateUnique() AS unique»
658           «IF unique==false»
659
660             <TR>
661           <TD><FONT FACE="Geneva, Arial" SIZE=2>
662           "Rc36: State names must be unique."</TD>
663             <TD>«qualifiedName»</TD></TR>
664           «ENDIF»
665
666   «ENDLET»
667 «ENDDEFINE»
668
669 «DEFINE rule37 FOR uml::StateMachine»
670         «LET hasTooManyTransitions() AS transitions»
671             «IF transitions==false»
672           <TR>
673           <TD><FONT FACE="Geneva, Arial" SIZE=2>
674           "Rc37: Initial state should have at least one Initial Node."</TD>
675             <TD>«qualifiedName»</TD></TR>
676           «ENDIF»
677         «ENDLET»
678 «ENDDEFINE»
```

Listing H.1: Xpand Html Report for complete Model

Appendix I

Modeling Workflow Engine (MWE) for Report Generation

```
1  <?xml version="1.0" encoding="windows-1252"?>
2  <!-- This is a main workflow file to generate html report-->
3  <!-- Author: Akhtar Ali Jalbani-- Date: 18-09-2010>
4  <workflow>
5    <property name="model" value=
6  "UMLCourseModel/BlueGroup/Blue1.1/UseCaseBäckerei_BlaueGruppe_10_04.uml" />
7    <property name="modeldir" value="BlueGroup"/>
8    <property name="src-gen" value="src-gen/${modeldir}" />
9    <bean class="org.eclipse.xtend.typesystem.uml2.Setup" standardUML2Setup="true" />
10   <bean id="mm_emf" class="org.eclipse.xtend.typesystem.emf.EmfRegistryMetaModel"/>
11   <component class="org.eclipse.xtend.typesystem.emf.XmiReader">
12       <modelFile value="${model}" />
13       <outputSlot value="modelSlot" />
14   </component>
15   <component id="dirCleaner"
16     class="org.eclipse.emf.mwe.utils.DirectoryCleaner"
17     directory="${src-gen}"/>
18   <!-- create html metrics report -->
19   <component class="org.eclipse.xpand2.Generator">
20     <metaModel idRef="mm_emf"/>
21     <fileEncoding value="ISO-8859-1"/>
22     <expand
23       value="templates::root::main FOR modelSlot" />
24     <outlet path="${src-gen}" overwrite="false">
25       <postprocessor class="org.eclipse.xtend.typesystem.xsd.XMLBeautifier">
26       </postprocessor>
27     </outlet>
28   </component>
29 </workflow>
```

Listing I.1: Example for WorkFlow Generator

Appendix J

Model-to-Model (M2M) Transformation Templates and Workflow Generator

```
1  import uml;
2  uml::Model transform(uml::Model model):
3    let elementList = model.eAllContents.typeSelect(UML::Element).collect(e|e):
4    elementList.forAll(e|renameElement(findElement(elementList, "Name of the Element")))->
5      model;
6  // find uml::Element in the model
7  List[uml::Element] findElement(List[uml::Element] elem, String name):
8    elem.select(e|e.name == name);
9  // rename UML::Element in the model
10 Boolean renameElement(List[uml::Element] elem):
11   elem.setName("New Name for the Element")->true;
```

Listing J.1: Rename Refactoring

```
1  <?xml version="1.0" encoding="windows-1252"?>
2  <workflow>
3    <!-- input uml model -->
4    <property name="model" value=
5   "UMLCourseModel/BlueGroup/Blue1.1/UseCaseBäckerei_BlaueGruppe_10_04.uml"/>
6    <!-- set the output directory-->
7    <property name="src-gen" value="src-gen" />
8    <!-- Setup UML2 support -->
9      <bean class="org.eclipse.xtend.typesystem.uml2.Setup" standardUML2Setup="true" />
10   <!-- load uml model -->
11   <component class="org.eclipse.xtend.typesystem.emf.XmiReader">
12         <modelFile value="${model}" />
13         <outputSlot value="modelSlot" />
14     </component>
15   <!-- Clean src directory -->
16   <component id="dirCleaner"
17     class="org.eclipse.emf.mwe.utils.DirectoryCleaner"
```

```
18      directory="${src-gen}"/>
19    <!-- model to model transformation -->
20    <component class="org.eclipse.xtend.XtendComponent">
21      <metaModel class="org.eclipse.xtend.typesystem.uml2.UML2MetaModel" />
22  // you need to change renameTransformation::renameActor for rename Actor refactorings
23      <invoke value="renameTransformations::renameElement::transform(modelSlot)"/>
24      <outputSlot value="outputSlot"/>
25    </component>
26    <!-- write output UML model -->
27    <component id="writer" class="org.eclipse.emf.mwe.utils.Writer">
28      <modelSlot value="modelSlot"/>
29      <uri value="./src-gen/Transformations/tranformedModel.uml"/>
30    </component>
31  </workflow>
```

Listing J.2: Refactoring Workflow

Appendix K

Quality Assessment Reports for the Bakery System

This appendix contains quality assurance reports for the Blue and Red group for Incomplete and Complete type of models.

K.1 Report of Group BLUE for Incomplete Model

ViolatedRule	*Location*
"R_{i1}: Each Use Case must be inside the subsystem"	No Name
"R_{i2}: Each Use Case must be associated with an actor"	No Name
"R_{i2}: Each Use Case must be associated with an actor"	Data::Bäckerei::Lagerverwaltung:: Zutatendatenbank verwalten
"R_{i2}: Each Use Case must be associated with an actor"	Data::Bäckerei::Lagerverwaltung:: Lagerkapazität verwalten
"R_{i2}: Each Use Case must be associated with an actor"	Data::Bäckerei::Lagerverwaltung:: Zutatenbestellung durchführen
"R_{i2}: Each Use Case must be associated with an actor"	Data::Bäckerei:: Verkaufssystem::Kassen starten
"R_{i2}: Each Use Case must be associated with an actor"	Data::Bäckerei:: Personalverwaltung::An Personalverwaltung anmelden
"R_{i2}: Each Use Case must be associated with an actor"	Data::Bäckerei:: Lagerverwaltung::Backplan eingeben
"R_{i5}: Each Usecase should not be associated to more than three actors"	No Name

"R_{i5}: Each Usecase should not be associated to more than three actors"	Data::Bäckerei::Lagerverwaltung:: Zutatendatenbank verwalten
"R_{i5}: Each Usecase should not be associated to more than three actors"	Data::Bäckerei::Lagerverwaltung:: Lagerkapazität verwalten
"R_{i5}: Each Usecase should not be associated to more than three actors"	Data::Bäckerei::Lagerverwaltung:: Zutatenbestellung durchführen
"R_{i5}: Each Usecase should not be associated to more than three actors"	Data::Bäckerei:: Verkaufssystem::Kassen starten
"R_{i5}: Each Usecase should not be associated to more than three actors"	Data::Bäckerei::Personalverwaltung:: An Personalverwaltung anmelden
"R_{i5}: Each Usecase should not be associated to more than three actors"	Data::Bäckerei:: Lagerverwaltung::Backplan eingeben

Table K.1: First Partial Report for Incomplete Model of BLUE Group

K.2 Report of Group RED for Incomplete Model

ViolatedRule	*Location*
"R_{i7}: Each subsystem contain minimum 3 and maximum 5 use cases i.e.,., UC= 3-5"	Data::Bäckerei::Lagerverwaltung
"R_{i7}: Each subsystem contain minimum 3 and maximum 5 use cases i.e., UC= 3-5"	Data::Rollen
"R_{i7}: Each subsystem contain minimum 3 and maximum 5 use cases i.e., UC= 3-5"	Data::Bäckerei
"R_{i7}: Each subsystem contain minimum 3 and maximum 5 use cases i.e., UC= 3-5"	Data::Bäckerei::Verkauf
"R_{i7}: Each subsystem contain minimum 3 and maximum 5 use cases i.e., UC= 3-5"	Data::Actors
"R_{i8}: Each subsystem Name should start with capital letter and contains one to two words"	Data::System-Level Use Cases

"R_{i8}: Each subsystem Name should start with capital letter and contains one to two words"	Data::High-Level Use Cases
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung::Mitarbeiter verwalten AD:: Mitarbeiter Daten aktualisieren
"R_{i17}: Each Activity in activity diagram should refers to usecases"	Data::Bäckerei::Verkauf::activity_use_cases:: Bestellung ablehnen
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung::Zeit erfassen AD::Zeit Aktualisieren
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use _cases::Backwaren eingeben
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung::Zeit erfassen AD::Eingangszeit Markieren
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Kundendaten eingeben
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Bestellung in Backplan aufnehmen
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Gesamtpreis berechnen
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Beleg drucken
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung::Lohn auszahlen AD::Arbeitszeit berechnen
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung::Zeit erfassen AD::Ausgagnszeit markieren
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung:: Schichtplan erstellen AD::Aushilfe einstellen
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Lagerverwaltung:: Aktivitätsdiagramm::Lager befüllen
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Kapazitäten berechnen
"R_{i17}: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Lagerverwaltung:: Aktivitätsdiagramm::Waren annehmen / kontrollieren
"Rule-17: Each Activity in activity diagram should refers to usecase"	No Name
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Kundendaten eingeben
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Verkäufer anmelden

"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung:: Mitarbeiter verwalten AD:: Neuen Mitarbeiter anlegen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Backwaren aufnehmen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Sonderkunden aufnehmen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung:: Schichtplan erstellen AD:: Arbeitstage berechnen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung:: Mitarbeiter verwalten AD:: Mitarbeiter Karteikarte aufrufen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung:: Schichtplan erstellen AD:: Schichtpläne erstellen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung:: Lohn auszahlen AD::Lohn berechnen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Personalverwaltung:: Schichtplan erstellen AD:: Schichtplan an Personalchef übergeben
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Bestellformular öffnen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Quittung erstellen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Backwaren austragen
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use _cases:: Backwaren eingeben
"Rule-17: Each Activity in activity diagram should refers to usecase"	Data::Bäckerei::Verkauf::activity_use_cases:: Bestellung bestätigen
"R_{i18}: Each subsystem of use case diagram should be represented as a class subsystem in class diagram"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung KD

"R_{i18}: Each subsystem of use case diagram should be represented as a class subsystem in class diagram"	Data::Bäckerei::Verkauf::Verkauf Klassendiagramm
"R_{i18}: Each subsystem of use case diagram should be represented as a class subsystem in class diagram"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung Klassendiagramm
"R_{i22} Each Class name should start with Capital letter and must be one word."	Data::Bäckerei::Personalverwaltung:: Personalverwaltung KD::Schichtplan Entity
"R_{i22} Each Class name should start with Capital letter and must be one word."	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung Klassendiagramm::Barcode Scanner

Table K.2: First Partial Report for Incomplete Model of RED Group

K.3 Report of Group BLUE for Complete Model

ViolatedRule	*Location*
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Backverwaltung:: Backverwaltung::Datenbank
"R_{c1}: Each Class should have attributes	Data::Bäckerei::Verkaufssystem:: Verkaufsystem::Datenbank
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::Einkaufsliste
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::ScannerTemperatursensor
"R_{c1}:Each Class should have attributes"	Data::Bäckerei::Verkaufssystem:: Verkaufsystem::LinkedList
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::AutoLieferant
"R_{c1}:Each Class should have attributes"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::Datenbank
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Verkaufssystem:: Verkaufsystem::Currency
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Backverwaltung:: Backverwaltung::Backplan
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Datenbank
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Kalender
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::PDF-Dokument

"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Verkaufssystem:: Verkaufsystem::Kassenbondrucker
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Adresse
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::Einkaufsliste
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Telefonnummer
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Verkaufssystem:: Verkaufsystem::LinkedList
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Kontodaten
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Verkaufssystem:: Verkaufsystem::Currency
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Backverwaltung:: Backverwaltung::Backwarenmenge
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Datenbank
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Backverwaltung:: Backverwaltung::Bestellmenge
"R_{c6}: Each Class Should have maximum 10 operations"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::LagerverwaltungGUI
"R_{c6}: Each Class Should have maximum 10 operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Personalverwaltung
"R_{c6}: Each Class Should have maximum 10 operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Schichtplan
"R_{c6}:Each Class Should have maximum 10 operations"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::Lagerzutat
"R_{c6}:Each Class Should have maximum 10 operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Schichtplanverwaltung
"R_{c6}:Each Class Should have maximum 10 operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::PersonalverwaltungGUI
"R_{c6}: Each Class Should have maximum 10 operations"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::Lagerverwaltung
"R_{c6}: Each Class Should have maximum 10 operations"	Data::Bäckerei::Verkaufssystem:: Verkaufsystem::Kasse
"R_{c6}: Each Class Should have maximum 10 operations"	Data::Bäckerei::Verkaufssystem:: Verkaufsystem::Kassenverwaltung
"R_{c6}: Each Class Should have maximum 10 operations"	Data::Bäckerei::Backverwaltung:: Backverwaltung::Backverwaltung

Table K.3: First Partial Report for Complete Models of BLUE Group

K.4 Report of Group RED for Complete Model

ViolatedRule	*Location*
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::DatenbankLagerverwaltung
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Schichplan
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Verkauf:: Verkauf::Reportobjekt
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Verkauf:: Verkauf::Date
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Verkauf:: Verkauf::Int
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::enum
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::Wareneingangssensor
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::Objekt
"R_{c1}: Each Class should have attributes"	Data::Bäckerei::Verkauf:: Verkauf::time
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Schichplan
"R_{c2}:Each Class should have Operations"	Data::Bäckerei::Verkauf:: Verkauf::Reportobjekt
"R_{c2}:Each Class should have Operations"	Data::Bäckerei::Verkauf:: Verkauf::Date
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Verkauf:: Verkauf::Int
"R_{c2}:Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::enum
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Verkauf:: Verkauf::Kunde
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::Objekt
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Verkauf:: Verkauf::time
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Stechkartensystem
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Karteikarte
"R_{c2}: Each Class should have Operations"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::Object
"R_{c2}: Each Class should have Operations"	No Name

"R_{c5}: Each Class name should start with Capital letter"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::enum
"R_{c5}:Each Class name should start with Capital letter"	Data::Bäckerei::Verkauf:: Verkauf::time
"R_{c5}:Each Class name should start with Capital letter"	Data::Bäckerei::Verkauf:: Verkauf::viud
"R_{c5}: Each Class name should start with Capital letter"	Data::Bäckerei::Verkauf::Verkauf::double
"R_{c5}:Each Class name should start with Capital letter"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::time
"R_{c5}: Each Class name should start with Capital letter"	Data::Bäckerei::Personalverwaltung:: Personalverwaltung::boolean
"R_{c5}: Each Class name should start with Capital letter"	Data::Bäckerei::Lagerverwaltung:: Lagerverwaltung::string
"R_{c5}: Each Class name should start with Capital letter"	Data::Bäckerei::Verkauf:: Verkauf::void

Table K.4: First Partial Report for Complete Model of RED Group

Curriculum Vitae

Akhtar Ali Jalbani

Personal Information

Date & Place of Birth:	27th April 1976, Village Wazir Khan Jalbani, Larkana Sindh, Pakistan
Nationality:	Pakistani

Academic Information

1981-1985	Primary Education: Govt. PC School Larkana Sindh, Pakistan
1986-1990	Secondary Education: Govt. Pilot Sec. School Larkana Sindh, Pakistan
1991-1993	Higher Secondary Education: Govt. Degree College Larkana Sindh, Pakistan
1995-1999	Bachelor in Electrical Engineering, NED University of Engineering and Technology, Karachi, Pakistan
2000-2002	Masters in Computer Software Engineering, National University of Science and Technology, Rawalpindi, Pakistan
Since 2007	PhD Student, Software Engineering and Distributed Systems Group, Institute for Computer Science, Georg-August-Universität Göttingen, Germany Funded by Higher Education Commission Pakistan and DAAD Germany