Fakultät für Informatik
der Technischen Universität München

ISSUE-BASED MODEL REVIEW

HELMUT N. NAUGHTON

Vollständiger Abdruck der von der promotionsführenden Einrichtung Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Hans Michael Gerndt
Prüfende/-r der Dissertation:	1. Prof. Bernd Brügge, Ph.D.
	2. Prof. Dr. Anne Brüggemann-Klein

Die Dissertation wurde am 12.07.2018 bei der Technischen Universität München eingereicht und durch die promotionsführende Einrichtung Fakultät für Informatik am 31.10.2018 angenommen.

Helmut Naughton: *Issue-Based Model Review,* Doktor der Naturwissenschaften (Dr. rer. nat.), © 2018

ABSTRACT

Reviews – and their variants, such as inspections and walk-throughs – have been an integral part of software engineering research and practice since their introduction by Fagan (Fagan, 1976) and they have been shown many times to be the most efficient way of removing defects in a software project (Parnas and Weiss, 1985; Porter and Votta Jr, 1997; Laitenberger and DeBaud, 2000; Ciolkowski et al., 2003; Hedberg and Lappalainen, 2005; Bacchelli and Bird, 2013; Salger, 2013).

These researchers focused on code and design artifacts, but not on models. With the rise of model-based techniques such as Model Driven Architecture (MDA) and Model Driven Design (MDD), the review of models is also becoming more important. A key part of these techniques is the use of incremental and iterative approaches.

Rationale management (Kunz and Rittel, 1970) has been shown to be an effective way of capturing decisions and proposed alternatives during the increments and iterations. In many projects, however, the process of capturing rationale is only done at the end of the project or for important milestones (Dutoit et al., 2007).

In this dissertation we have developed a method that treats model reviews as a continuous activity which integrates review feedback into rationale management. The method reduces review related communication peaks and enables structured externalization of review knowledge for access by all the stakeholders.

The method has been implemented as a framework for software engineering projects using a CASE tool or an issue tracker.

It has been validated in two exploratory case studies, demonstrating the feasibility of issue-based model reviews in practice, specifically for improving documentation quality, and feedback traceability and coverage.

ZUSAMMENFASSUNG

Reviews - und ihre Varianten, Inspektionen und Walk-Throughs - sind ein integraler Bestandteil der Software-Engineering Forschung und Praxis seit ihrer Einführung durch Fagan (Fagan, 1976) und es wurde mehrfach gezeigt, dass sie der effizienteste Weg zur Entfernung von Defekten in Software Projekten sind (Parnas and Weiss, 1985; Porter and Votta Jr, 1997; Laitenberger and DeBaud, 2000; Ciolkowski et al., 2003; Hedberg and Lappalainen, 2005; Bacchelli and Bird, 2013; Salger, 2013).

Diese Forscher waren primär auf Code und Design Artefakte fokussiert, nicht auf Modelle. Mit dem Aufkommen von modellbasierten Techniken wie Model Driven Architecture (MDA) und Model Driven Design (MDD) nimmt die Bedeutung von Modell-Reviews zu. Ein Schlüsselaspekt dieser Techniken sind inkrementelle und iterative Ansätze.

Rationale management (Kunz and Rittel, 1970) ist ein etablierter Weg, Entscheidungen und vorgeschlagene Alternativen während der Inkremente und Iterationen zu erfassen. In vielen Projekten wird Rationale sonst nur am Ende eines Projektes oder zu wichtigen Meilensteinen erfasst (Dutoit et al., 2007).

In dieser Dissertation haben wir eine Methode entwickelt, die Reviews von Modellen als eine kontinuierliche Aktivität behandelt, welche Feedback aus Reviews mit Rationale Management kombiniert. Diese Methode reduziert Review-bezogene Kommunikationsspitzen und erlaubt eine strukturierte Externalisierung von Review-Erkenntnissen, die allen Stakeholdern zur Verfügung stehen.

Diese Methode wurde als ein Framework für Software Engineering Projekte, welche ein CASE Tool oder einen Issue Tracker verwenden, implementiert.

Sie wurde in zwei explorativen Fallstudien validiert, welche die Umsetzbarkeit von Issue-basierten Modell-Reviews in der Praxis zeigten, spezifisch zur Verbesserung der Dokumentationsqualität, und zur Traceability von Feedback.

ACKNOWLEDGMENTS

In the first place, I would like to thank my advisor Prof. Bernd Brügge, Ph. D. for not only giving me the opportunity to do research at his chair, but also encouraging me to follow through with my ideas. During the time at his chair I learned a lot about research and about teaching, and I will always be grateful for having worked with him. His passion for teaching inspired me, and his dedication to combine experiences from industry with research and teaching prepared me well for my following endeavors.

My colleagues at the chair were always there to discuss interesting topics of research and beyond. Thank you to Harald Stangl, Michaela Gluchow, Michael Nagel, Yang Li, and in particular to Maximilian Kögel and Jonas Helming, who got me started at the chair. A special thank you also to my office mate Florian Schneider, who was a great source of inspiration and a perfect collaborator for numerous research ideas.

The staff at the chair was also always there for me, so I want to thank Monika Markl, Helma Schneider, Uta Weber and Ruth Demmel for all their assistance.

Without the support of the managing directors at Linova Software GmbH, Andreas Löhr, Horst Mauersberg, and Tobias Weishäupl I could not have completed this dissertation. Thank you for allowing me to finish my research after I transitioned from academia to industry.

I also want to thank my parents, Gudrun Naughton and James Henry Naughton for always supporting me and believing in me. Without them, I would not have been able to do this.

Finally, I also want to thank my friends, who provided moral and practical support. Thank you to Stefan Fritsch, Andreas Hofmann, Thomas Kirschmann, and especially to Mario Romsy, who found the right words at the right time.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

INTRO

1 INTRODUCTION

Reviews in software engineering projects (e.g. technical reviews, inspections, or walkthroughs) have been the subject of many handbooks (Freedman and Weinberg, 1990; Hollocker, 1990; Gilb and Graham, 1993; Wheeler et al., 1996; Rösler et al., 2013) and have been shown by multiple studies to be very efficient in reducing the number of errors in artifacts which were reviewed (e.g. Fagan, 1976; Parnas and Weiss, 1985; Porter and Votta Jr, 1997; Laitenberger and DeBaud, 2000; Ciolkowski et al., 2003; Hedberg and Lappalainen, 2005; Bacchelli and Bird, 2013; Salger, 2013).

For some professions, for instance medicine, reviews are even mandatory (Jetley et al., 2006), following defined standards (*CLSI GP31-A*, 2009). In other areas, such as financial services, audits with outside participation are common (Croll, 2003). Reviews, however, are expensive to conduct (Johnson, 1994), context dependent (Ciolkowski et al., 2002; Ott and Raschke, 2012; Bacchelli and Bird, 2013), and need to follow specific principles and processes in order to be effective (Shirey, 1992).

Although models are an important tool of conceptualizing and structuring large projects, and reviews in early stages of the software development process are more cost effective since defects are cheaper to fix when found early, the focus of most reviews at the moment is still on the code level. Since most projects are not yet fully model driven, and formal methods are expensive to implement, a balanced way of reviewing models is needed which helps designers as well as developers but does not put too much strain on the process.

In this chapter we will first introduce some terminology and then present the components of a framework for integrating rationale management with model review. The chapter concludes with an outline of this dissertation.

1.1 TERMINOLOGY

In the area of reviews and knowledge management, there are a number of similar sounding terms with distinct meaning. In the following, we define these terms, clarify whether they are relevant to the contents of this dissertation, and specify potential deviations from the meaning of the terms in this dissertation.

The definition of *issue* used in this dissertation is taken from the original definition of Rittel and Kunz (Kunz and Rittel, 1970) in their paper on Issue-Based Information Systems (IBIS):

> Issues are the organizational 'atoms' of IBIS-type systems. Among their properties are:
>
> - Issues have the form of questions.
> - The origins of issues are controversial statements.
> - Issues are specific to particular situations; positions are developed by utilizing information from the problem environment and from other cases claimed to be similar.
> - Issues are raised, argued, settled, 'dodged', or substituted.

The following definitions are taken from the IEEE Standard for Software Reviews and Audits (*IEEE Std 1028-2008*, 2008):
A *walkthrough* is defined as

> A static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible anomalies, violation of development standards, and other problems.

Walkthroughs are not part of this dissertation, which defines processes for reviews in the sense below, without one person leading the others through the relevant material. The issue-based capture process, however, might also prove beneficial in this case and could be subject to further investigation.
A *review* is defined as

> A process or meeting during which a software product, set of software products, or a software process is presented to project personnel, managers, users, customers, user representatives, auditors or other interested parties for examination, comment or approval.

There are two subcategories of reviews, the technical review and the managerial review.
A *technical review* is defined as

> A systematic evaluation of a software product by a team of qualified personnel that examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards. [...] Technical reviews may also provide recommendations of alternatives and examination of various alternatives.

This type of review is concerned with software engineering models or other software engineering artifacts.
A *managerial review* is defined as

> A systematic evaluation of a software product or process performed by or on behalf of management that monitors progress, determines the status of plans and schedules, confirms requirements and their system allocation, or evaluates the effectiveness of management approaches used to achieve fitness for purpose.

This type of review is concerned with project management questions.

This dissertation focuses on technical reviews, in particular on the review of models, which are part of a software product and technical in nature.

An *inspection* is defined as

> A visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications. [...] Inspections are peer examinations led by impartial facilitators who are trained in inspection techniques. Determination of remedial or investigative action for an anomaly is a mandatory element of a software inspection, although the solution should not be determined in the inspection meeting.

Since inspections require specially trained facilitators, they are out of scope for this dissertation.

Similar to walkthroughs, inspections could also use an issue-based model such as the one presented in this dissertation to better note down findings, but the corresponding processes are not tailored towards inspections and their formal structure. Nonetheless, as described in Chapter 4 ,we can integrate the more formal aspects of inspection methods into our review process, especially the setup for the search for a solution. This fits well together with the use of issues for capturing a problem or a question independent of its proposed or actual solutions.

An *audit* is defined as

> An independent examination of a software product, software process, or set of software processes performed by a third party to assess compliance with specifications, standards, contractual agreements, or other criteria. [...] An audit should result in a clear indication of whether the audit criteria have been met.

As audits require third party assessors, they are out of scope for this dissertation.

An *anomaly* is defined as

> Any condition that deviates from expectations based on requirements specifications, design documents, user documents, standards, etc., or from someone's perceptions or

> experiences. [...] Anomalies may be found during, but not limited to, the review, test, analysis, compilation, or use of software products or applicable documentation.

A *defect* is defined as

> An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.

In this dissertation, we will later model anomalies as issues and defects as action items.

1.2 COMPONENTS OF A MODEL REVIEW FRAMEWORK

The framework presented in this dissertation aims to alleviate these problems for model review by targeting three aspects at once. First, it proposes a formal model for entities related to model reviews (Chapter 3), second it integrates this model with existing rationale and knowledge management techniques, and third it provides corresponding processes and artifacts (Chapter 4) – all as part of a unified software engineering model (Wolf, 2007; Helming, 2011).

The following sections highlight the components behind the design of the Issue-Based Model Review framework and serve as basis for the needs the framework has to satisfy.

1.2.1 *Model-based approach*

Integrating reviews directly into a comprehensive software engineering model such the unified model proposed by Wolf (2007) allows for usage of task, rationale, and knowledge management techniques to support the review process, resulting in three key benefits.

First, integration of reviews into the unified model provides traceability and context. This allows review results to be viewed in context as part of the project, with traceability links to other relevant project artifacts.

Second, representing reviews as first class citizens of a unified software engineering model enables software engineers to formally specify the rules according to which the review should be done and represent the rules themselves as part of the overall model.

Third, integration into a unified model allows for automation of tasks expressed in a machine-readable format (e.g. in the Object Constraint Language OCL (ISO, 2012)). Automation enables reviewers to concentrate on rules which can only be evaluated by humans and uses the computer to check what can be checked algorithmically.

Since the framework is designed to track changes to both the model and the rules according to which a review is conducted, it is possible

to directly see which reviews are out of date, because the model has changed, and therefore need to be redone. On the one hand, changes to the model necessitate checking if the changed model elements (as well as the model as a whole) are still valid according to the review rules. On the other hand, changes in review rules require the entire model to be examined again, since different review rules can cause defects in model elements which were previously considered defect-free according to the old rules.

Framework support for the common tasks of re-checking a changed model and re-checking a model according to a changed rule set prevents unnecessarily checking already reviewed artifacts, as well as missing reviews of artifacts which were changed since their last review. Especially in safety or security critical domains, this is important since regulations in such domains often mandate keeping proof of formal reviews. It can also help in projects with inexperienced developers, such as large student project courses with industrial clients.

1.2.2 *Support for rationale and knowledge management*

In order to integrate the findings of model reviews into the model itself, a way of representing these findings needs to be established. For this purpose, we took a look at the field of knowledge management and in particular at rationale management. As part of this dissertation, we present a framework for the integration of a simple form of rationale management as part of a unified software engineering model with a model for representing reviews and their findings.

With the framework, reviews are an intrinsic part of the software model and therefore part of the knowledge management base of the project. For many projects knowledge preservation in form of rationale management is rather difficult, especially for beginners, because according to Shum (1996), rationale management consists of three cognitive tasks.

The first task is classification, "deciding what kind of an idea one has", i.e. the task of determining the right way to capture a piece of knowledge. The second task is naming, "how to label [the idea] meaningfully", i.e. the task of labeling the piece of knowledge in a way that others (and oneself, after sufficient time has passed) can easily find it again. Ideally, it also gives the reader a rough understanding of its contents without having to completely read its details. Finally, the third task is structuring, "how [the idea] relates to other ideas", i.e. allowing for traceability between a piece of knowledge and other pieces of information or knowledge.

While the task of naming remains complicated[1], the framework helps with classification and structuring of rationale from reviews. It

1 "There are only two hard things in Computer Science: cache invalidation and naming things" (Phil Karlton) – http://martinfowler.com/bliki/TwoHardThings.html

allows rationale construction to be done "at runtime". When reviews are conducted to ensure quality, the framework allows their results to be directly integrated into the rationale base of the project. This is preferable to reconstructing rationale "after the fact" (Dutoit and Paech, 2012).

1.2.3 *Tailorable process*

We designed the framework to be usable even in the context of projects staffed with beginners. In order to substantiate this claim, we conducted two case studies in university projects with industrial clients (c.f. the chapter on CASE STUDIES). The framework had to be able to be used by inexperienced developers, both in terms of software engineering knowledge in general, as well as proficiency in conducting reviews under time pressure, and with as little overhead as possible. Since these projects are conducted with industrial clients, who expect results they can use in practice, a trade-off had to be made between producing high quality models and creating an executable deliverable. The review process had to be simple enough to prevent developers from falling into analysis paralysis (Brown et al., 2000), but also robust enough to improve the model and to prevent developers from just "hacking" away. So the processes had to be tailorable to support this setting.

The framework allows for tailorable processes, from a simple check by one person, through informal, team-internal reviews, up to formal technical reviews with outside participation (c.f. the section on Review processes. These processes are broken down into individual steps to enable the adaptation of the review process to project specific needs. We also introduce a strategy for enabling reviews to be done as a continuous activity throughout the entire software development life-cycle, instead of the currently used event-based discrete strategy (c.f. the section on Continuous Review).

Teaching or improving software engineering skills is an important part of these projects, but this takes time away from the project. Since the same is true for project-based organizations, especially ones willing to employ large numbers of inexperienced (and therefore cheaper) programmers, such organizations can also benefit from the use of this framework. In large project-based organizations, projects benefit from a model-based approach, which provides context for novices and allowes for better dissemination of knowledge in order to cope with different levels of experience of employees.

The framework even allows for the review of reviews themselves by providing a meta-process to help with process improvement tasks, since reviews and the according rules are part of the model (c.f. the section on Meta-reviews and process improvements). Relevant indicators such as number and type of defects, mean-time to fix, number of

re-opened issues can be readily obtained by querying the underlying model.

1.2.4 *Focus on review of models*

Figure 1.1 shows the results of Google Books n-gram searches for the most common types of reviews and inspections.

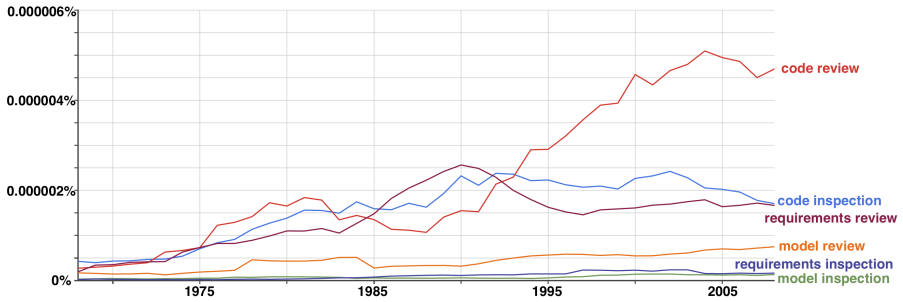

Figure 1.1: Google Books n-gram results for reviews and inspections between 1968 and 2008

One sees a rising of interest in reviews and inspections, starting with Fagan's seminal paper on reviews (Fagan, 1976). The concept was first codified in the "IEEE standard glossary of software engineering terminology" by the IEEE, 1983. Code reviews and inspections are the main focus of publications during that timeframe, other types of reviews and inspections are less commonly written about. While requirements reviews are at least somewhat common (Gorschek and Svahnberg, 2005), model reviews remain a niche subject.

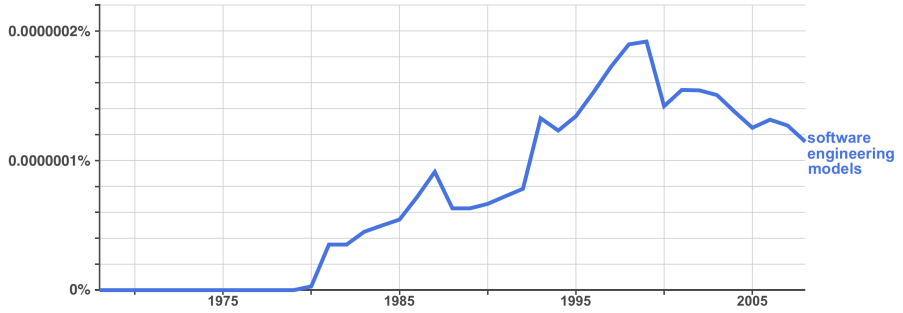

Figure 1.2: Google Books n-gram results for software engineering models between 1968 and 2008

Models, however, have risen in importance for software engineering purposes (c.f. Figure 1.2) and are now the driving force behind an entire software engineering approach (Selic, 2003).

In order to prevent rationale from getting lost, the framework should allow defects found during reviews to be modeled as issues, and combining them with proposals and criteria according to which they

should be evaluated. These issues should then be anchored to meetings or milestones in order to prevent them from getting overlooked as the project continues.

1.3 OUTLINE

The outline of this dissertation is as follows.

We first discuss related work in Chapter 2 and analyze shortcomings with regards to review of software engineering models. In Chapter 3 we define one of the two principal components of Issue-Based Model Review, the *IBMR meta-model*, and cover matters related to traceability and change tracking. The other component, the IBMR Review Process, is described in Chapter 4. We present the core process and its variants, as well as options for tailoring it to a specific project. The evaluation of IBMR is presented in Chapter 5 where we look at two case studies and a survey. The dissertation concludes with a summary and an outlook in Chapter 6.

2 RELATED WORK

Inspection, a formal method of reviewing software engineering artifacts (in particular source code), was developed at IBM by Michael Fagan (Fagan, 1976). It describes a process by which a trained group of people closely study selected software engineering artifacts for the presence of defects. These defects are then recorded and subsequently fixed. Inspection is a formal process consisting of multiple tasks from preparation to rework and follow-up, employing people in many different roles, such as author, reader, or tester, and it is led by a moderator.

In software engineering projects inspections are used to improve the software quality and reduce the number of defects of the software system. Most of these inspections, however, are done on the level of source code, although inspections on other levels of abstraction are also possible. Many modern-day projects, such as projects employing model driven development methodologies, would benefit from inspections on a higher level of abstraction. This would enable project participants to find defects much earlier in the project and on a higher level of abstraction – for example defects in the requirements during an analysis review. This would reduce the cost of correcting them significantly.

Since inspections according to processes like Fagan's are considered heavyweight and elaborate by some, those projects may choose substitute inspections by more informal processes like reviews or walkthroughs. Especially in agile projects, fully formal inspections are rare – Ambler (2004) even argues that they can be considered as a process smell. The results of more informal review processes, however, are often not acted upon as much as hoped for and so decisions are delayed, and information is lost – there should be a person explicitly responsible checking if corrective actions were taken on the basis of the review's result (Doolan, 1992).

At the same time, models are becoming more and more complex by incorporating an increasing amount of technical and organizational aspects. They ideally offer rich traceability information, letting the user determine the relevant context. Integrated models like the MUSE software engineering model (Helming, 2011) offer even deeper integration into project management. MUSE proposes to integrate the previously disparate models for modeling systems and for modeling project management related information. Helming showed that integrating these models helps developers and project managers better understand the context their decisions and tasks exist in. By using

the available traceability information, developers and managers can choose to be informed by the CASE tool itself about changes relevant to them. If a developer for instance worked on a task related to a specific component in the system, they are most likely interested in changes to that component, or components that directly interface with it. A project manager benefits from MUSE being able to make suggestions which developer is best suited to work on a task regarding a specific component, based on the developer's history of tasks involving this or related components.

By extending this integrated model with review centric model elements, we propose to make reviews first class citizens of the model-based software engineering process and to take advantage of traceability information obtained from this integration, allowing for connections not only with the system model and the artifacts reviewed, but also with project management information.

2.1 REVIEW TECHNIQUES

In the following section, we summarize the most common and influential review techniques and point out their shortcomings with regards to the needs listed in the previous chapter.

2.1.1 *Fundamentals*

The basic terminology and concepts for reviews originate from Fagan's initial publication or from the IEEE Standard for Software Reviews and Audits.

Fagan

Most review and inspection techniques in use today are based on the inspection method developed by Michael Fagan at IBM in the early 1970's and published under the title "Design and Code Inspections to Reduce Errors in Program Development" in the IBM Systems Journal in 1976 (Fagan, 1976). Fagan further detailed his inspection method in the follow-up publication "Advances in Software Inspection" in 1986 (Fagan, 1986). He designed inspections to be a formal method of reviewing software engineering artifacts (with particular focus on source code), describing them as a "formal, efficient, and economical method of finding errors in design and code"(Fagan, 1976).

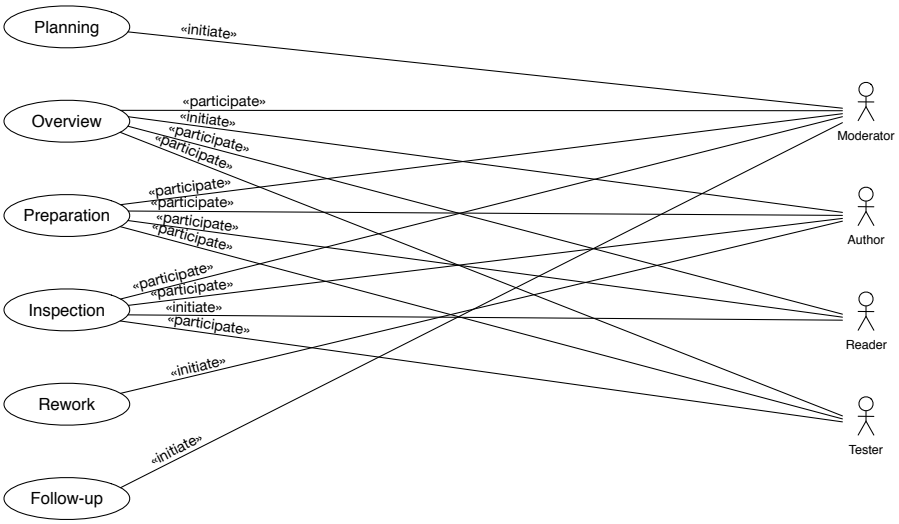

Figure 2.1: Inspection use cases from Fagan

In his first publication on inspections, Fagan (1976) distinguishes between moderator, designer, coder (or implementer), and tester. The moderator is the "key person" for conducting reviews, and along with the tester remains unchanged in his responsibilities in the second publication (Fagan, 1986). The designer and coder/implementer in this set of roles are combined into the author role for the second publication. The reader role which is defined in Fagan (1986) is not present in the first publication.
Fagan (1986) defines the following actors and use cases (as seen in Figure 2.1), which are the basis for the roles of subsequent inspection/review process variants (note that there may be more participants than just those roles, but these then serve no dedicated function for the purposes of the inspection and just provide feedback):

MODERATOR The moderator manages the inspection team, schedules the meeting, reports on its outcome, and follows up on potential re-work. They ideally are from outside the project and have received training in team coaching and moderating inspections but are also a good programmer.

AUTHOR The author is the creator of the artifact (design or code document) and is present to answer questions related to it.

TESTER The tester "views the product from the testing standpoint", i.e. is primarily concerned with aspects like verification, validation, and testability.

READER The reader "paraphrases the design or code as if they will implement it" and brings an outside perspective into the team.

The planning use case is mainly done by the moderator, who checks that the entry criteria for an inspection are met (with regards to the

maturity of the artifacts to be inspected), chooses the participants, arranges their availability, and schedules the meeting.
The overview use case is initiated by the author, who provides more information to the group on the artifacts to be inspected. The other roles, e.g. reader, are also assigned here if this was not done before.
The preparation use case is done by all inspection participants individually. This use case is meant to familiarize them with the artifacts to be inspected and to let them prepare for their roles.
The inspection use case is the heart of the process and is when the group gets together under the leadership of the moderator and checks the artifacts for defects. The inspection session is meant for finding defects and not for trying to find solutions, so the moderator is responsible for keeping the meeting on track.
In the rework use case, the author resolves the defects found during the inspection.
Finally, in the follow-up use case, the moderator verifies that the defects are correctly resolved and that no new defects were introduced.

Inspections are scheduled before implementation begins (to verify the correctness of the requirements), during implementation to check the code, and at the end of the implementation to ensure the correctness of the entire component and to check if it satisfies the requirements.

IEEE 1028:2008

The Institute for Electrical and Electronical Engineering (IEEE) published a "Standard for Software Reviews and Audits". Of all the specifications presented in this standard, we focus on the ones for technical reviews and inspections. Management reviews, walkthroughs, and audits are out of the scope for this dissertation.

General properties of systematic reviews
IEEE defines three characteristics which are needed for systematic reviews. The first is team participation, the second is documentation of results, and the third is documented procedures for conducting a review.

Technical reviews
According to IEEE 1028, a technical review is a "systematic evaluation of a software product by a team of qualified personnel that examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards." A technical review is different from a regular review, which is less systematic and is used to gather feedback from relevant stakeholders ranging from project participants to clients, or even to end users. Management reviews focus on project status and planning and are not covered in

this dissertation, although the methods presented here could be used, if the project management itself was done in a model-based approach.

For technical reviews, IEEE 1028 defines the following roles, which can be viewed as actors in the resulting use case model.

DECISION MAKER The person whose decision making should be improved by the review.

REVIEW LEADER The person responsible for conducting the review.

RECORDER The person responsible for capturing the outcome of the review.

TECHNICAL REVIEWER The people conducting the actual review.

Besides the roles above, IEEE-1028 also defines the following procedures for conducting technical reviews. These procedures (as seen in Figure 2.2) can be seen as use cases which the actors defined above initiate and participate in.

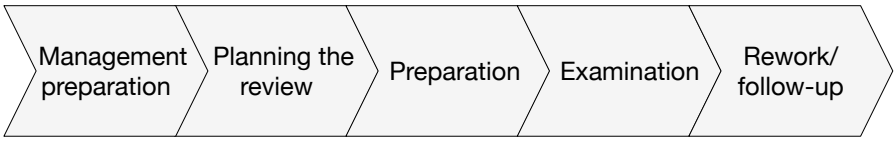

Figure 2.2: Technical reviews according to IEEE 1028

MANAGEMENT PREPARATION In order for a technical review to be successful, management needs to allocate time and resources, provide training, support conducting the review, and remedy any discrepancies found during the review.

PLANNING THE REVIEW The review leader needs to select suitable reviewers and assign specific responsibilities, schedule the review meeting, and set a timetable for the entire review process.

PREPARATION Every technical reviewer should prepare for the review meeting by familiarizing themselves with the artifacts to be reviewed before the actual meeting. If they discover any discrepancies they should note them down to be collected later.

EXAMINATION The actual examination takes place in form of a meeting. In this meeting, participants closely examine the artifacts to be reviewed and check them for completeness, consistency, rules or conformity to regulations, and suitability to their intended purpose. They then decide on the importance and urgency of the findings and the recorder notes them down.

REWORK/FOLLOW-UP The review leader follows up on the implementation of the findings of the review, ensuring all findings are remedied.

Inspections according to IEEE-1028

An inspection is defined as a "visual examination of a software product to detect and identify software anomalies, including errors and

deviations from standards and specifications." It can also target non-functional requirements and check for quality or be used to collect data or inform decision making.

Inspection teams consist of two to six members, the author of the artifact to be inspected always among them. IEEE-1028 defines the following roles and responsibilities for an inspection team:

INSPECTION LEADER Responsible for planning and organizing, selecting the inspection target, preparing and conducting the inspection meeting and issuing the inspection outcome.

RECORDER Responsible for recording the inspection outcome.

READER Responsible for presenting the software product to the inspection team in a "comprehensive and logical fashion".

AUTHOR Responsible for providing details about the software product and for doing rework later on.

INSPECTOR Responsible for "identifying and describing" anomalies in the software product.

The procedures defined for inspections (see Figure 2.3) are quite similar to those for technical reviews, though they differ slightly in their execution.

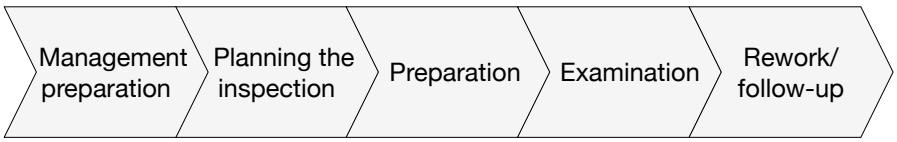

Figure 2.3: Inspections according to IEEE 1028

MANAGEMENT PREPARATION As with technical reviews, management support is needed for time and resources, training, support, and follow-up.

PLANNING THE INSPECTION The author collects the artifacts to be inspected for the inspection leader. The inspection leader assembles an inspection team and schedules a meeting. They also set anticipated inspection rate (pages or lines of code per hour).

PREPARATION The inspection team individually inspects the artifacts before the actual examination meeting.

EXAMINATION During the inspection meeting, the inspection team collects anomalies in a list. At the end the team makes an exit decision, either "Accept with no verification or with rework verification", "Accept with rework verification", or "Reinspect".

REWORK/FOLLOW-UP The inspection leader has to ensure that rework is executed as decided on.

The use case model of inspections following the IEEE-1028 standard is shown in Figure 2.4.

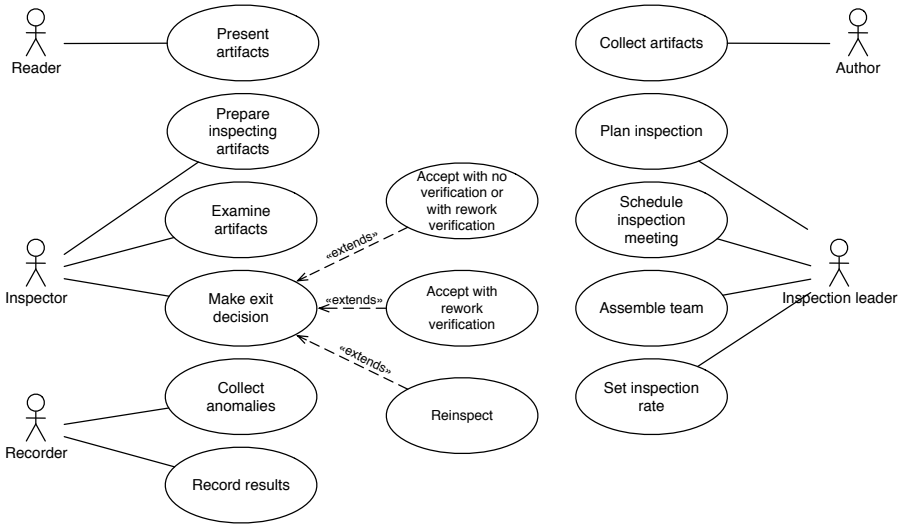

Figure 2.4: Use case diagram of IEEE 1028

2.1.2 *Textbooks*

Multiple textbooks cover the topic of reviews and inspections.

Gilb and Graham

Gilb and Graham (1993) wrote one of the definite textbooks on software inspections. In their book, they provide historical context and describe the benefits of software inspections. In the first half of the book, they break the inspection process up into four parts – Initiation, Checking, Completion, and Process Improvement. They especially go into detail on the responsibilities of the inspection leader.

In the second half of the book, they present six case studies showing the results of applying software inspections in practice.

Rösler et. al.

Rösler et al. (2013) provide a German language textbook on reviews. Since this book was published after the introduction of agile software development methods, they also look at the relationship of reviews and agile software development. They emphasize that reviews are essential no matter what development methodology the project follows.

According to a study presented in this book, 95% of all defects are found during individual preparation (or individual checking in the terminology used by Gilb and Graham).

They also provide a list of reading techniques for reviews.

AD HOC There is no special technique, the reviewers just take a look at the artifacts.

CHECKLIST-BASED The reviewers have a checklist and check the artifacts according to the items on the checklist.

PERSPECTIVE-BASED The reviewers check the artifacts according to the point of view of a specific stakeholder. This is especially useful for requirements documents.

ABSTRACTION-DRIVEN This technique is also called "Reading by stepwise abstraction", and is targeted at design and code documents, in contrast to requirements documents. The reviewers start with the basic structure and successively extract functionality, which is then compared to the specification.

They stress that follow-up to reviews is important to check up on the rework and to make release decisions.

The costs of the reviews in terms of effort spent by the review participants is outweighed by preventing costlier corrections later on by finding major defects early, by providing process feedback and suggestions for improvement, by reducing the error rate of authors (through learning from review results and applying this knowledge to future work), and by providing metrics.

Overall they sketch an estimated ROI (return on investment) as

$$\text{ROI} = \frac{\text{Rework effort saved} - \text{Review effort}}{\text{Review effort}}$$

Freedman and Weinberg

Freedman and Weinberg (1990) specify the following activities for their review process: Selecting Reviewers, Conducting the Review, and Reporting the Results of the Review.

Freedman and Weinberg define three types of reviews: Informal Review, Formal Technical Review, and Project Control Review.

Informal reviews are performed team-internally, are technical in nature, and can be conducted with low effort. Formal technical reviews are the standard type technical review with dedicated reviewers from outside the team itself, and as their name implies have a technical focus. Project control reviews are managerial instead of technical and are aimed at determining the progress of the project and the progress towards milestones or such. Project control reviews may be partially based on the findings of a number of formal technical reviews.

Freedman and Weinberg distinguish formal technical reviews from informal reviews in three aspects: First, a formal review has a written report, i.e. a formally structured summary of the findings of the review team available to all interested stakeholders, while an informal review might only have internal notes or no real written record at all. Second, active and open participation. It is important that all participants know their roles and responsibilities and are doing their share for the success of the review. Third, formal reviews emphasize the full responsibility of all participants for the quality of the review.

They also emphasize the need for consensus, since only recording comments (without discussing the issues and coming to a consensus) allow reviewers to shirk responsibility and not find solutions to the issues they uncover.

Reviews need to be open, i.e. more than one person has to participate in the review. This acts as a check against reviewers who are uninterested, not prepared, or misunderstand the material.

Issues should not be discussed during the review. Since the purpose of the review is to find as many issues present in the artifacts under review as possible, discussing them during the review meeting takes away valuable time. Also, because of the time pressure, the solutions tend to be less well thought out then when solutions to the issues are discussed separately. Although this might be difficult to accomplish at first, since many technically minded people are quick to jump to thinking about solutions, separating collecting issues from solving them leads to more efficient reviews and to better solutions to these issues.

The report for a formal technical review consists of three parts: The first is what was reviewed, i.e. the names or identifiers of artifacts under review and their producers. Second, the report specifies who did the reviewing. This section of the report contains the names, the roles, and signatures of the review participants. Third, the report states the conclusion reached during the review.

Freedman and Weinberg differentiate between major and minor revisions for changes made necessary by reviews. A minor revision lets the issue be fixed without necessitating a new review. A major revision requires another review after the fixes have been made.

Freedman and Weinberg make related issues part of the documentation of a work unit. This holds only for unresolved issues, resolved issues in their approach are only preserved in the Summary Report as part of the historical archive.

For reports, the sheer number of issues is not a good metric. Freedman and Weinberg observe that making the number of issues a metric only leads to project participants changing the way they write issues. If it is encouraged to find as many issues as possible, they will split up defects they find into units as small as they can make them, and report any inconsistencies they find as issues, thus potentially burying more important ones. Conversely, if a project having many issues is considered undesirable, project participants will reduce the number of reported issues by raising the bar of what is considered to be an issue, and by grouping (potentially tangentially related) issues together into one. Freedman and Weinberg instead propose to look at the outcome of the reviews, i.e. how many major or minor revisions were needed for the artifact under review.

An inspection evaluates material “confining attention to a few selected aspects, one at a time”.

Informal reviews allow a producer of an artifact to gain feedback from familiar people before gathering feedback from outside sources. They are less expensive, since they are done ad hoc and in a much smaller scale.

Freedman and Weinberg suppose that the focus on code as opposed to other software related artifacts for reviews is due to the "tendency to blame errors on [the coding] step", since it is the last.

Hollocker

Hollocker (1990) also distinguishes between project reviews (focused on the software development process and aimed at gathering information about the status of a project, as well as possibilities for process improvements), product reviews (focused on the software under development and aimed at assessing the quality of the software product and at uncovering defects), and audits (aimed at meeting regulatory or legal requirements). Again, audits are out of the scope of this dissertation, although model reviews as described in Chapter 4 can be used to prepare for audits.

Product reviews are the type of reviews which are of the most interest to us for the purposes of this dissertation. Hollocker defines three subclasses of product reviews. First, there are technical reviews. This type of review focuses on deviations of the software artifact under review from its specification (verification) or intended purpose (validation) and is conducted by a small group, usually a leader, a scribe, and a regular team member. The second type is the software inspection. Software inspections are closely related to technical reviews but are held with a larger group of participants (including a dedicated moderator and reader in addition to the scribe) and follow more of a checklist-based approach, focusing on potentially necessary rework of the software artifact. Finally, the third type of product review is the walkthrough. In contrast to the first two types of review, the walkthrough is led by the author of the software artifact under review. They present the artifact and explain its purpose and structure, while the rest of the team ask questions and try to find defects in the artifacts presented.

2.1.3 *Other publications*

The idea of using inspections for improving code quality was proposed by Michael Fagan (Fagan, 1976) in 1976. In his paper he details the necessary process steps and roles. He also reports on a study on coding productivity.

In 1986, Fagan (1986) looks back at 14 years of code inspections and presents improvements on the original review ideas. Among other changes, Fagan proposes extending the inspection method to require-

ments, documentation, test cases, and other software engineering artifacts instead of focusing only on source code.

Ambler (2004) in his book on agile model-driven development sees reviews as a kind of "process smell" and claims that agile methods do not need reviews anymore, since test- or behavior driven development can find defects more quickly and cheaply. On his blog "Agile Modeling" (Ambler, 2003), however, Ambler admits that even in an agile context, reviews can sometimes be useful. Sometimes, regulatory requirements may necessitate conducting reviews. For products not created collaboratively, reviews can bring a better understanding and a feeling of shared ownership. Reviews can also serve as proof for stakeholders that the project is progressing well. For larger projects, they can be used as proof that the chosen architecture is suitable. Finally, if outside guidance is desired, reviews are a way of incorporating such guidance in an agile context.

Reviews are not limited to the code alone. Fey and Stürmer (2007) propose to use inspections for quality assurance in model-based development. They suggest a combination of automated checking and manual inspection to improve the quality of models. Their approach, however, is not targeted at software engineering models (such as UML models), but at mathematical models and models used for simulation (e.g. Matlab). The only real software engineering model they discuss is the requirements model, which they represent in the requirements tool DOORS. They also propose to review the requirements model entirely in DOORS, a tool not well suited to such a purpose.

Kemerer and Paulk (2009) conducted a study showing that "both higher design quality and higher code quality are consistently associated with higher software quality as measured by fewer test defects".

Babar and Gorton (2009) surveyed the state of the practice for architecture review and found that while many of the surveyed companies practiced architecture review of some sort there was much room for improvement.

From their survey, they reported that "The majority of architecture reviews are informal, without a systematic approach guiding the review process." Only 40% of respondents of the survey used checklists, and less than 20% used metrics or similar approaches (respondents were allowed to give more than answer). The majority of respondents (80%) answered that the primary technique was personal experience.

They also found "practitioners [to be] more inclined to leverage individual review techniques (for example, scenarios) from the research community rather than use a complete method to support architecture review processes." This indicates that a tailorable approach consisting of activities which can be adjusted to project specific needs might be more easily accepted than monolithic approaches.

Finally, "most architecture reviews occur on an ad hoc basis. Ad hoc reviews might not provide long-term process improvement to increase an organization's return on investment." This shows that providing a guiding structure in which reviews can be held can help improve organizational learning and save money later on in the project.

Especially the tool support was found to be lacking severely, with only limited specialized tool support available out of the box. While some companies were reported to have developed their own tooling, the authors state their hope that specifically designed review support tooling will make it easier for more companies to enjoy the benefits of architecture review.

Bernhart et al. (2010) provide an approach for reviews in agile software development. While code reviews have many benefits, they are usually seen as time-consuming and expensive, which makes them at odds with typical agile development. The authors therefore propose a continuous approach to code reviews to reduce the overhead and to make code reviews more feasible in an agile setting. Commits into the source code management system (SCM) automatically trigger the creation of reviews, which are assisted by showing file diffs obtained from the SCM. They differentiate between post-commit reviews, which happen after the changes are already accepted into the SCM, and pre-commit reviews, which are conducted before the changes are actually integrated into the source code on the SCM.

Gorschek and Svahnberg (2005) found that "After the initial specification requirement reviews were not common practice.".

2.2 RATIONALE MANAGEMENT

The field of rationale management has produced many competing approaches. Since the advantages and disadvantages of different manners of expressing captured rationale are outside of the scope of this dissertation, we will only briefly highlight the history of rationale management by describing the IBIS and QOC approaches, and then move on to rationale management in the unified model, which forms the basis of rationale management used in the remainder of this dissertation.

2.2.1 *Issue-Based Information Systems*

The idea of Issue-Based Information Systems (IBIS) was introduced by Kunz and Rittel (1970) to help with political decision making. Initially IBIS was conceptualized as a human operated system, which was only later replaced with computers. The authors characterize IBIS as a system to "[guide] the identification, structuring, and settling of issues raised by problem-solving groups, and [provide] information pertinent to the discourse".

The following concepts are defined by IBIS:

TOPIC A topic is the basic area, often denoted by a "trigger phrase".
ISSUE An issue is a point of contention related to the topic, which is caused by participants having different positions.
ARGUMENT An argument is a statement made to support a certain position.
QUESTION OF FACT A Question of Fact is a part of the discussion where objective answers exist, and experts can be called on to provide answers.
POSITION A position is a point of view of a participant with regards to a certain issue.
MODEL PROBLEM Model problems correspond to scientific or managerial models and are applicable to many situations.

The last three of these concepts are not relevant for the purposes of this dissertation, since they focus on objective answers, comparable problems, and positions of individuals. We mainly focus on issues and arguments in the following.

Issues are managed by IBIS in the "Issue bank" subsystem. This subsystem contains all recorded issues and can be queried by their status (living, settled or abandoned, and latent). IBIS keeps track of the discussion using an "Issue map", which graphs the relationship between issues, questions of fact, arguments, etc.

The other subsystems of IBIS are the "Evidence bank", which contains answers to questions of fact, the "Handbook", a collection of model problems, a "Documentation system" to support search and analysis, and a "Topic list".

IBIS is mean to help the participants to "[develop] more structured pictures of the problem and its solutions" through a "counterplay of questioning and arguing".

2.2.2 *Questions, Options, and Criteria*

Questions, Options, and Criteria (QOC) by MacLean et al. (1991) is a semi-formal notation used in Design Space Analysis. The authors describe the key concepts as follows: "The main constituents of QOC are Questions identifying key design issues, Options providing possible answers to the Questions, and Criteria for assessing and comparing the Options."

QOC was constructed to be useful both while designing, and while redesigning or reusing at a later time. For this, it supports representing design rationale, with reasons for why a specific design was chosen and what the consequences of changing it later might be. It can also be used to facilitate communication about design.

A question gives rise to multiple options, representing different design possibilities. An option, when chosen, can lead to more questions

being raised. Criteria are dimensions in the design space according to which trade-offs of design decisions, i.e. options, can be assessed. An option can be rated according to multiple criteria. These ratings, called assessments, can be either positive or negative and are usually expressed as binary choices, although multiple levels would also be possible (though the authors recommend against this, because of the added complexity).

The notation was designed to represent unstructured discussions and can be employed using only a pen and paper system. Its main goal is to "lay open for argument the elements of the rationale". The biggest difference to IBIS is the lack of elements denoting statements of fact, and the focus on criteria as a means of assessing the importance of options, independently of individual persons. While IBIS has the notion of arguments, criteria are more specific and measurable in comparison.

2.3 RUSE, MUSE AND MEETING MANAGEMENT

The model presented in this dissertation is based on the results of research on three previous software engineering models and extends them by focusing on an aspect not explicitly supported by these models.

The three underlying models, the Rationale-based Unified Software Engineering model, the Management-based Unified Software Engineering model, and the Meeting Management in the Unified Model are detailed in the following sections.

2.3.1 *Rationale-based Unified Software Engineering model*

Rationale-based Unified Software Engineering (RUSE) was developed by Wolf (2007) as a way of integrating rationale directly into software models by creating a meta model allowing for direct connections between technical model elements such as use cases or classes and rationale or task-oriented model elements such as issues (open questions) or tasks. These connections are made possible by representing all possible model elements in a consistent manner, with a unified representation for all artifacts related to the software engineering process, and by allowing for arbitrary dependency traces between them, as shown in Figure 2.5.

The RUSE model "integrates system models, collaboration models and organizational models", improving inefficiencies in the communication of distributed project participants, and reducing inconsistencies between related development artifacts, such as software models and the corresponding documentation. System models are the models traditionally used in software engineering, with artifacts such as stakeholders, requirements, hazards, and UML models. These models

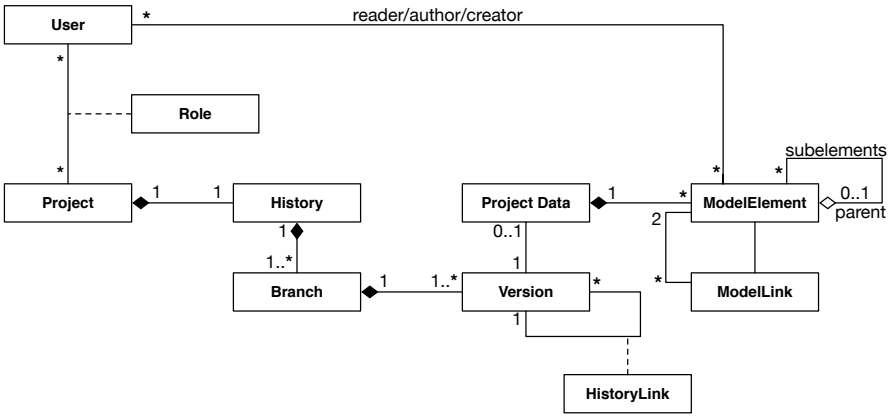

Figure 2.5: Meta model overview of RUSE according to Wolf, 2007

are used to model the system under development itself, i.e. its goals, architecture, and structure.

The collaboration model is not directly related to the system under development, but instead represents communication related to the development, such as documentation, requests for clarification, comments, open questions, tasks, and milestones to name a few. These artifacts have been seen as separate by most other software engineering frameworks and were therefore insufficiently integrated with the rest of the project artifacts.

The organizational model models the organizational units of the project, i.e. the teams and participants. Modeling these elements explicitly allows creating dependency traces between e.g. project participants, their assigned tasks, and the subsystems of the software under development related to these tasks.

In summary, RUSE addresses the problem of inconsistent models, inefficient collaboration, and lost rationale in software projects by providing a singular meta-model encompassing the system model, collaboration model, and organizational model all in one. Rationale is captured as part of the collaboration model in the form of Issues, Proposals, and Criteria, similar to the QOC (Questions, Options, Criteria) approach by MacLean et al. (1991).

2.3.2 *Management-based Unified Software Engineering model*

The Management-based Unified Software Engineering model (MUSE) builds on RUSE. MUSE extends the RUSE meta-model by providing more explicit support for project management modeling artifacts such as sprints, milestones, and iterations. These project management artifacts are as much part of the model as traditional system model elements such as requirements, scenarios, use cases and classes. Both subtypes of model element can be grouped into projects as part of sections, as seen in Figure 2.6. This integrates more context into the

meta-model, allowing for automating tasks like bug assignments and supporting change awareness.

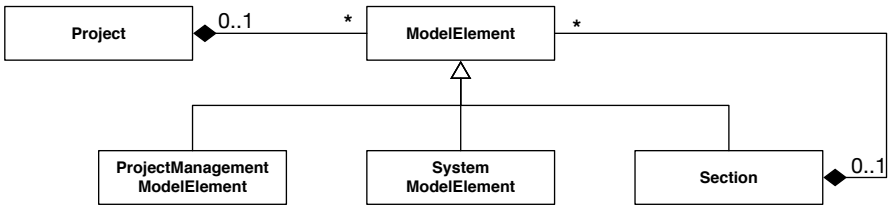

Figure 2.6: Muse model structure from Helming, 2011

The project management model as defined by MUSE is shown in Figure 2.7. Tasks are modeled in the form of action items, input by external stakeholders is captured in the form of requests, the rationale model is again based on QOC (MacLean et al., 1991), and grouping of work items can be done via work packages. These work packages can be used to model iterations and milestones. Project participants are modeled as users and can be grouped together in teams as part of the organizational unit composite pattern.

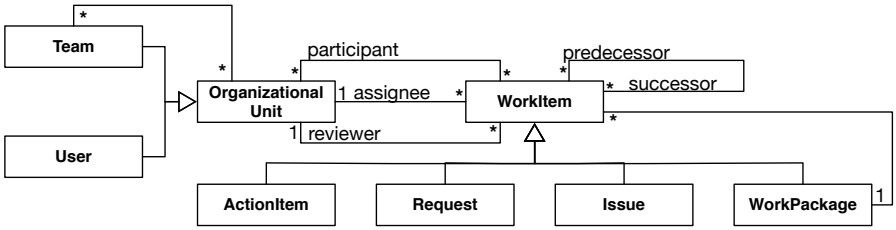

Figure 2.7: The project management model according to Helming, 2011

2.3.3 *Meeting Management in the Unified Model*

Building upon the Management-based Unified Software Engineering model, "Meeting Management in the Unified Model" (MMUM) (Naughton, 2008) merged MUSE with the meeting concepts presented in "Object Oriented Software Engineering" by Bruegge and Dutoit (2009), which in turn are based on concepts developed at XEROX (Doyle and Straus, 1982). Doyle and Straus propose three main parts to a meeting, the first being "status and information sharing", the second "discussion", and the third "wrap up". MMUM introduced an explicit Meeting model element and structured the meeting sections according to Bruegge and Dutoit (2009), as shown in Figure 2.8.

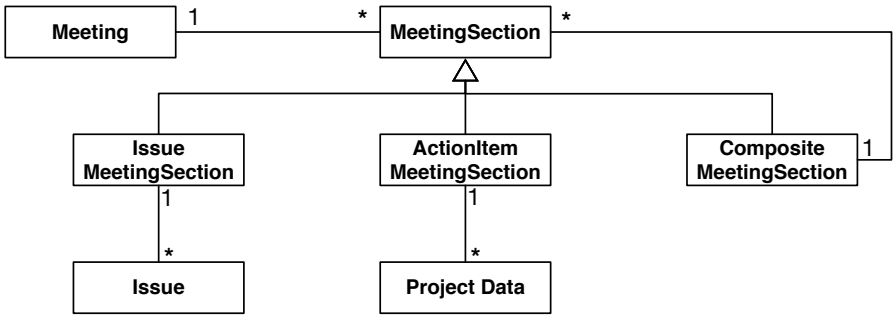

Figure 2.8: Analysis model of the meeting integration into the unified model from Naughton, 2008.

A MMUM meeting contains an ActionItem meeting section for status and information sharing, in which the ActionItem that were worked on since the last meeting are captured and reported on. It also contains an Issue meeting section, which includes all the open/unsolved questions to be discussed during the meeting, or which arose during the meeting. Finally, a second ActionItem meeting section is used to capture all ActionItems which were created during a meeting.

2.3.4 *Rationale capture in textual communication*

Complementary to the Unified Model, new approaches to rationale capture have been proposed. REACT (Alkadhi, 2018) focuses on analyzing textual communication between developers in chat tools or issue trackers in order to capture rationale. REACT shows that with automated assistance, communication artifacts can be used to facilitate capture and reconstruction of rationale. This approach can be used in addition to the approaches described above, to capture rationale from textual communication and therefore can help enrich the rationale model part of the unified model.

2.4 REVIEW DELIVERABLES

Part of every review is the documentation of the outcome. In the following, we describe the different ways the results of a review can be made persistent (shown in Figure 2.9):

		Traceability	Creation	Comprehension	Implementation
Document-based	**separate**	--	++	--	++
	integrated	+	+	o	+
Issue-based	**separate**	-	+	+	+
	integrated	++	o	++	--

Figure 2.9: Comparison of review deliverables.

Document-based separate

The findings of the review can be written down in a stand-alone document. This can either be done in an analog way by taking notes a piece of paper, or digitally with a writing software such as TextEdit, or Word, a spreadsheet software such as Excel, or a note-taking application such as Evernote.

While capture on paper allows for the quickest capture and easiest addition of additional rich content such as sketches or diagrams, paper is also the most problematic in terms of referring back to the results. It is hard to distribute, since only one original copy exists, which must either be Xeroxed or scanned in order to be provided to the other participants. Also, managing and storing paper copies is a cumbersome task.

There are, however, also some disadvantages to separate documents in digital form. The review results are captured separate from the subject matter which was reviewed, which means that it is hard to read the document and immediately understand what is being written about and where the concrete problems lie. On a similar note, it is hard to trace from the defects mentioned in the report to the affected model elements. While some tools provide mechanisms to link to specific model elements, this is not universal and also not all digital note taking applications support the concept of linking.

Advantages	**Disadvantages**
All in one location	Disjunct from the subject matter
Easy to read/write	Hard to find/trace
	Discussion not well-structured

While this approach provides ease of use for both authors and readers of review documentation, the lack of traceability and of structure pose major problems.

Document-based integrated

The main alternative to noting down findings in a separate document, is integrating the findings with the subject matter. With support for comments or attachments, the findings can be done in-line via comments or attached to the document reviewed.

The advantages of this approach are that the findings are easy to note down, and also to understand for a reader. In the integrated approach, the results are more closely linked to the document being reviewed. In the case of attached results, at least the document they refer to is traceable. In the case of in-line attachments it is even better, here the comments can be attached to the relevant parts of the document.

Unfortunately, this solution still does not provide optimal traceability. Since traceability links are not explicit, they can be lost when the document is restructured or changed. Comments to deleted model elements are not preserved, leading to loss of information. Also, the discussion about model elements is only captured ad hoc and not in a unified way.

Advantages	**Disadvantages**
Easy to write	Discussion not well-structured
Attached to the document	Traceability problems

This approach lacks the comfort and power of dedicated modeling and rationale management tools (such as semantically meaningful traceability links, customized editing features, or support for automation).

Issue-based separate

The issue-based separate approach captures defects found during a model review in the form of action items and issues, in a tool separate from the one used for modeling. This approach has the advantage of capturing discussions in a unified way and according to best practices. It also gives the users more freedom in choosing the tools they want to use for a particular purpose, which helps with acceptance. Using separate tools brings with it the usual problems of traceability, discussed by Gotel and Finkelstein (1994) and others. A good discussion and useful hints on dealing with traceability issues this can be found there.

Advantages	**Disadvantages**
Discussion well-structured	Traceability problems
Detached from document	

Due to the lack of support for traceability, context awareness or review automation may suffer with this approach.

Issue-based integrated

Findings in a review can also be captured in the issue-based integrated approach. This presupposes a tool suitable for modeling not only the system itself, but also aspects of the project itself (such as rationale management), as proposed by Helming (2011). One such tool is Unicase, which supports modeling the system model as well as the project model. Findings are expressed as action items and issues attached to the model elements in question with explicit traceability links.

Advantages	Disadvantages
Discussion well-structured Findings traceable to source Attached to the document	Depends on unified model

This approach combines traceability support with support for rationale management.

2.5 TOWARDS BETTER SUPPORT FOR MODEL REVIEW

The quality of models in software engineering projects suffers from many different problems. First, developers are often not experienced at modeling, and may not have enough understanding of current software engineering concepts and terminology (Black, 1994). Second, developers may only work on projects part-time and as such they balance work on the project with other demands on their time. They need to switch contexts often, which makes a cognitively hard job like modeling even more difficult, since the modeler needs to recreate the detailed understanding they had when last working on the model. Third, they often have trouble partitioning the problems in a suitable way and thus produce overlapping work, which needs to be brought into sync. Finally, developers often see modeling as more of an academic exercise for which they do not see the immediate benefit, as opposed to programming, which they are more familiar with. Because of this, it is necessary to motivate and assist them in the creation and enhancement of models and to show developers what they can gain from a larger upfront investment of time and effort. One of the ways of improving model quality is by giving them better tools and processes with which to review their models. This dissertation provides a model such tools can use and accompanying processes. The model and processes were evaluated in a pair of case studies.

2.5.1 *Shortcomings of current model review approaches*

Currently, model reviews are mostly captured in form of stand-alone documents, describing what was reviewed and what the findings were. For example, they may be stored as comments on a wiki-page for the model. They are usually not linked to the changed model over time, meaning that updated versions of the model do not allow access to previous review results, and issues and their resolutions are hard to reconstruct. A suitable model review environment needs to allow for traceability between old versions of the model, the issues found in these models, their resolutions, and the models resulting from them. These models would then also be traceable throughout the entire software engineering process down to the source code level.

If there is no unified way of capturing review results, this also leads to the danger of duplication of issues. If issues are recorded in separate documents, it is more likely that information already captured is duplicated, wasting valuable review time. With a unified way of capture, review results are easier to see, thus avoiding duplication. Also, the tool itself can help spotting duplicates by doing textual analysis on the recorded issues.

The rules and guidelines according to which a model review is conducted, should also be persisted. Currently, there is often no easy way of seeing the rules regarding which a model was already checked, making reviews hard to reproduce. Also, such persisted guidelines can serve as a basis for reviews later in the project, or in subsequent projects.

When model reviews are done on the basis of exported diagrams or diagrams uploaded to wikis or other collaboration sites, it is not guaranteed that the model that was reviewed is the latest version of the model. A next generation model review approach should ensure that only the latest model is available for review, with older versions only being accessible for reference. Reviews should also be attached to specific versions of the model, such that review comments are always consistent with the model shown. In current approaches, a model might have its own wiki page with associated comments, which may be inconsistent to the latest version of the model shown on the page.

If there is no tool support for model review, it is hard to guarantee that all model elements have indeed been checked. A tool supported version of the process can keep track which model elements have been checked and which have not been, reducing errors of omission in the review process.

If model review is conducted with tool support, this allows to calculate quality metrics on the models, listing what percentage of model elements have been checked according to which guidelines. Ensuring model quality otherwise becomes a difficult task, since self- or peer-checking is hard for inexperienced developers.

Automatic tool support can also help improving the frequency of reviews, by keeping track of the review schedule, alerting when reviews are delayed, or if no reviews have been scheduled for a certain amount of time. Currently, reviews are often scheduled ad hoc or only slightly in advance, often as a result of informal or formal feedback on the model. The predictability of the review schedule would be much improved if it were more integrated into the development process.

2.5.2 *Requirements for a next generation model review framework*

In this section, we present a list of functional requirements a framework targeted at model review should address in order to cope with the issues mentioned above.

Specifically targeted at model review

Current available review techniques and tools do not offer specific support for models. Also, unified modeling environments as developed by Wolf (2007) and Helming (2011) offer capabilities for special treatment of models in reviews which are not available in other modeling tools. They can leverage traceability and context to better understand a model element and can store the review history together with the revision history.

Integration reviews with project model and system model

With the integration between project model and system model developed by Helming (2011), reviews can be related to elements of the system model which were reviewed, as well as to the project model, which allows to specify e.g. results tasks. Also, the full information on creation and history of a model element to be reviewed is readily available to reviewers. With current processes and tooling, the creation and change history of a model element is not necessarily immediately obvious. Associated iterations or sprints are not accessible.

Enable delayed decisions

Decisions about defects are sometimes not immediately possible. Input by project participants not attending the meeting, or even by outside experts may be necessary. A next generation model review framework needs to support capturing open questions and tracking them in order to ensure they are resolved in a timely manner.

Capture rationale at runtime

According to Dutoit and Paech, 2012, rationale should be captured during the project execution, not after the fact. By treating reviews and rationale as separate concepts, the effort necessary to keep both up to date and relevant is not shared. By integrating these concepts and embedding them into a unified software engineering model, reviews can form the basis of rationale capture.

Separate discussions from the review

Johnson, 1994 writes "The expense of review meetings and the complexity of software dictates that review sessions not evolve into pro-

blem-solving sessions". A review framework needs to address this and offer a solution to separate review from discussion, but at the same time capture and store any suggestions which are brought up during review.

Support change tracking on a small scale

With current approaches, there is no explicit support for review and change tracking on a small scale (i.e. model element) level. Often, reviews are done at a higher level of granularity than the individual model element, e.g. at diagram level, and small incongruences are lost. The unit of analysis for model reviews should be the individual model element itself.

Follow-up on the resolution of defects

Without tight integration into the rest of the development process, review results can be forgotten, and the underlying defects are not resolved, or resolved too late, when subsequent steps have already been started based on the faulty, incorrect models. A system for model reviews needs to ensure that review results are treated as a regular part of the development process, meaning that the follow-up is directly integrated into regular meetings and traceable to the model itself.

Part II

MAIN

3 THE ISSUE-BASED MODEL REVIEW MODEL

The IBMR meta-model builds upon the MUSE meta-model which combines system modeling and project modeling as proposed by Helming (2011). The MUSE meta-model itself is based on the RUSE meta-model by Wolf (2007).

As described in Chapter 2 MUSE combines two previously separate aspects. First, the system under development, which is modeled with the help of standard techniques, such as UML diagrams, textual use cases, and functional and non-functional requirements. Second, the project itself, which is modeled with the help of action items and bug reports, work packages and milestones, users and groups, meetings, and rationale in the form of an issue model.

The chapter is structured as follows: First, we give a high-level overview of the meta-model in Section 3.1. In Section 3.2 we go into specifics of both the model elements newly introduced by the IBMR meta-model as well as their connections to the model elements already present in the MUSE meta-model[1] which are relevant for conducting reviews. For each class, its attributes and its associations are described in detail, accompanied by UML diagrams for a visual overview. Whenever a class is referenced in the text, it is formatted in the following way: `SomeModelElement`.

Wolf (2007) defines `ModelElement` as "the most abstract and generic modeling class that represents any concept of the software engineering domain. Classes of the IBMR meta-model are subclasses of this generic superclass `ModelElement` and are therefore collectively referred to as *model elements*. A model element can have many child model elements and can be linked by Model Link classes to many other model elements." In the remainder of the diagrams in this chapter, model links are depicted as UML associations, and all depicted classes are model elements unless noted otherwise.

3.1 OVERVIEW

Figure 3.1 gives an overview of all relevant model elements and associations. The packages in the diagram represent the three main components of the IBMR meta-model, the basic RUSE elements and issue-model, the MUSE project management extensions, and finally

1 Since many of the model elements referred in this chapter have their origin in the RUSE meta-model, we refer to them as being part of the RUSE/MUSE meta-model (the MUSE meta-model is a true superset of the RUSE meta-model and is generally regarded as its successor).

the IBMR specific extensions. We start with the model element central to the review process, the `ReviewResult`.

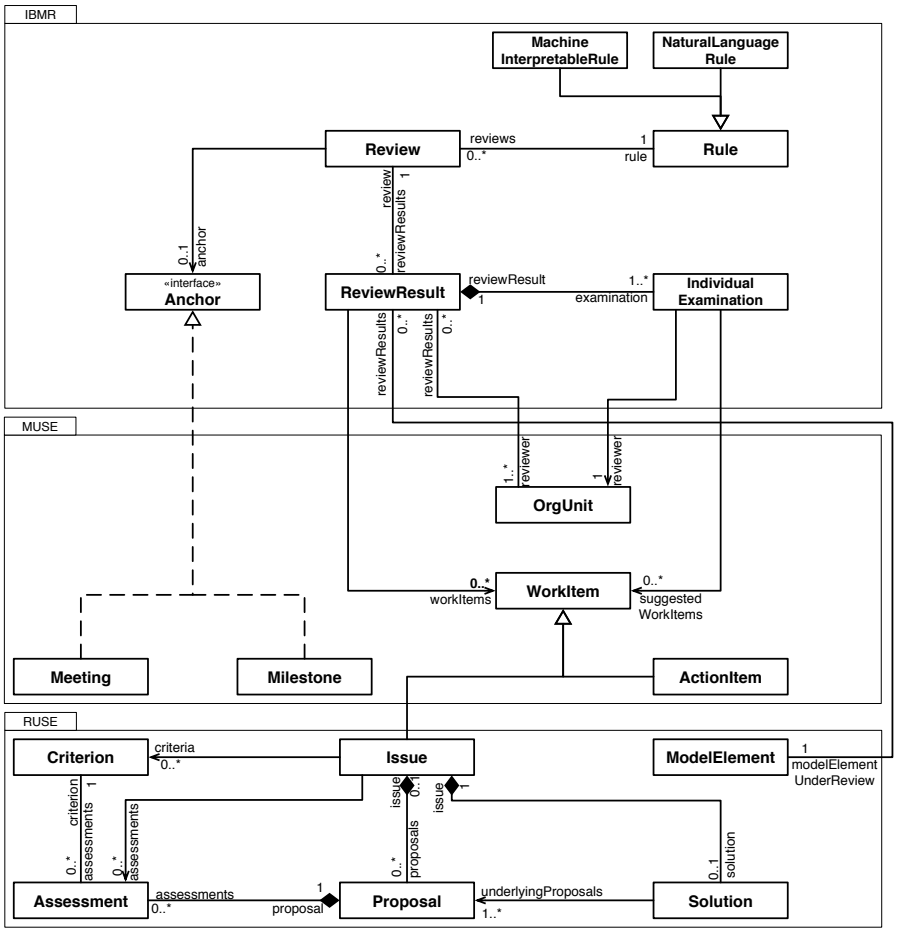

Figure 3.1: The IBMR Meta-Model (overview)

The model element **`ReviewResult`** expresses the result of a full examination of the model element under review at a point in time. It uniquely refers to one `ModelElement`, whereas a `ModelElement` can have multiple `ReviewResults` (according to different criteria and at different points in time). The time of the `ReviewResult` is recorded, as is the version of the `ModelElement` that was reviewed for models which are under version control. It has associations to the person or group responsible (the reviewing `OrgUnit`), their `IndividualExamination` if applicable, the `Rule` the `Review` conformed to, the `Review` which can group connected `ReviewResults`, the `WorkItems` detailing the work to be performed or the questions to be discussed, and the `Anchor` specifying the point in time the `ReviewResult` should be checked upon. A `ReviewResult` without any attached `WorkItems` signifies that a `ModelElement` was reviewed and deemed in compliance with the `Rule` of the review. A `ModelElement` with linked `ReviewResults` which themselves are linked to `WorkItems` is either defect, if all

corresponding WorkItems are not yet resolved, or was defect and has been fixed, if all corresponding WorkItems are resolved.

A number of thematically connected ReviewResults are grouped together in a Review. A thematic grouping might be according to project phase, or according to project specific regulations. ReviewResults can also be grouped temporally, for instance all ReviewResults resulting from a particular team meeting. The **Review** model element can be viewed as corresponding to a formal document summarizing the findings of a round of examinations and contains links to all individual ReviewResults.

A **Rule** describes a set of criteria according to which the ReviewResult is determined. A ReviewResult can only be associated with one Rule, but each Rule can be used in multiple ReviewResults. A Rule can either be an instruction in natural language, modeled as a **NaturalLanguageRule**, or a machine interpretable rule, modeled as a **MachineInterpretableRule**. Rules can refer to Properties of a ModelElement to clearly indicate which part of the ModelElement should be focused on during the review. NaturalLanguageRules specify the check to be performed on the model element under review in the form of human readable text detailing the necessary criteria for the model element to pass review. Logical expressions are used in MachineInterpretableRules to exactly specify the checks to be done in a computer interpretable format (e.g. an OCL constraint).

The Properties (attributes and relationships) of the ModelElement which are to be examined are related to each Rule. This allows tools to detect if the Properties of a ModelElement which already had been reviewed have changed, and to flag the ModelElement flagged as unreviewed and as to be checked again.

The person or group responsible for producing a ReviewResult is modeled as an **OrgUnit** from the MUSE meta-model. OrgUnits can be individuals (**Users**) or **Groups**. The results of an individual examiner reviewing a ModelElement is captured as an IndividualExamination. The synthesis of all IndividualExaminations forms as a ReviewResult.

An **IndividualExamination** uniquely belongs to a ReviewResult and represents the examination of a ModelElement according to the rules of a Review by a single OrgUnit, that is a group or an individual. An IndividualExamination can also have a relationship with a number of WorkItems constituting the work to be done in order to fix a defect.

If a ModelElement is defect according to the Rules of the Review, then, depending on the complexity of the fix, **WorkItems** are used to capture the tasks to be done in order to fix the defect or the discussions to be held in order to fix it. A WorkItem can either be an ActionItem, representing a task to be done, or an Issue, representing a question to be solved.

The **ActionItem** model element represents a task with an assignee, a due date, and a status which can be tracked. It can also contain additional information such as the corresponding software life-cycle activity, preceding or succeeding ActionItems, estimations or effort, and priorities.

The **Issue** model element from the MUSE meta model is used to capture defects detected in the examination. A ModelElement is regarded as valid according to a Rule if the last ReviewResult regarding this Rule had a Review without any Issues or ActionItems attached. Issues are classified by their priority (how important they are) and their urgency (how fast a decision has to be made).

In order to facilitate solving issues, **Proposals** can be made and attached to the Issue. Proposals constitute suggestions on how to find **Solutions** to solve an Issue. These Proposals are evaluated with **Assessments** according to **Criteria** also captured during the review.

Proposals can contain the **Changesets** necessary to perform the operations on the model to bring it into the proposed state. For details on executable proposals see Koegel (2011).

Because some Issues cannot be decided immediately, ReviewResults are anchored to an **Anchor** which is either a **Meeting** or a **Milestone**. This allows managers and developers to track the progress towards solving Issues and ensures that no Issues get lost. Every ReviewResult with open Issues attached to it needs to have an Anchor with a date laying in the future, thus ensuring a decision on the Issue is scheduled.

3.2 IBMR META-MODEL

The IBMR meta-model consists of three packages. Section 3.2.1 details model elements from RUSE/MUSE, that the IMBR meta-model builds most closely upon. Section 3.2.2 goes into detail specifically on the implementation of rationale management used. Finally, Section 3.2.3 provides specifics on the extension to the RUSE/MUSE meta-model introduced to model Review related classes.

3.2.1 *Basics of the RUSE/MUSE meta-model*

The model presented in this dissertation builds on the RUSE/MUSE model described in Section 2.3.1 and Section 2.3.2. It incorporates predominantly a variant of the Issue model of RUSE, which will be detailed in Section 3.2.2, and the basic RUSE/MUSE model elements described in this section, mainly model elements for modeling tasks and organizational units.

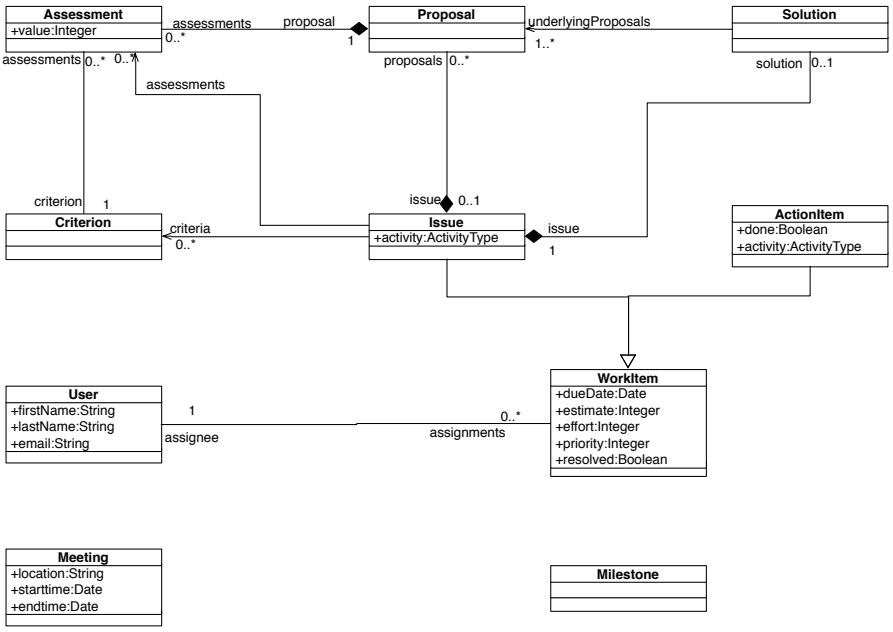

Figure 3.2: Basic elements of the RUSE/MUSE model

A UML representation of `ModelElement` and `ModelLink` is shown in Figure 3.3. Model elements can form two kinds of graphs. First, tree-like hierarchies with their *subelements* association, in which one model element (the parent) can have multiple sub elements, but each model element has at most one parent. Second, arbitrary graph edges via model link classes, which act as binary associations with exactly one source and one target. These model links allow traceability between any classes in the MUSE meta-model. Since model links are model elements themselves, they also can have a name and description and carry semantic information about the kind of link they represent.

A `ModelElement` has a `uniqueID`, uniquely identifying the model element, a `name`, identifying the model element to a human reader and briefly indicating its contents (often with a short sentence or fragment) and a `description`, which provides a detailed characterization of the model element. This characterization can be as detailed as necessary, since it is only shown when a user wants to obtain detailed information about the model element instance. It also provides a mechanism for being versioned with a `version` number (the versioning of software engineering models is described in Koegel, 2011). Model elements also keep track of their `creator` and `creationDate`.

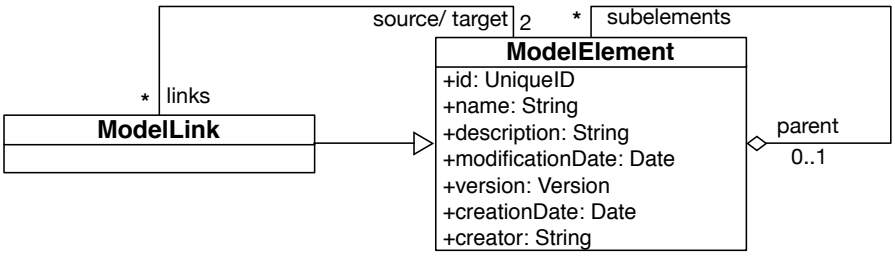

Figure 3.3: The basis of the RUSE/MUSE meta-model: `ModelElement` and `ModelLink`

`WorkItem` is the superclass of all model elements representing units of work. For a unit of work to be considered a `WorkItem`, it has to satisfy three conditions. First, it has to have a specific person responsible for its outcome. Second, the have a due date, which distinguishes them from e.g. todos, which do not specify such a time frame. Third, they can be resolved, meaning they have clearly defined metrics for deciding whether they have been successfully completed or not.

Subclasses of `WorkItem` are model elements such as `ActionItem`, `Issue`, `BugReport`, or `Milestone` which will be described in the following. Properties they have in common are having an `OrgUnit` as an `assignee`, and a `dueDate`. `WorkItems` also capture the associated estimated effort as `estimate`, expended effort as `effort`, and their priority as `priority`, all in the form of integers. A `WorkItem` also has a boolean attribute `resolved`, which reflects whether this work on this item has been completed.

`ActionItem` represents the simplest type of WorkItem, a task to be completed. In addition to the attributes inherited from `WorkItem`, `ActionItem` can keep track of the project `activity` (e.g. analysis, design, implementation, testing). In the IBMR framework, `ActionItems` are used to track the simplest types of defects, which can be repaired by a clearly definable task.

`OrgUnit` is the superclass of the composite organizational structure built out of users and groups. It provides both classes with common attributes such as an `ID`, a list of `groupMemberships`, and a list of all `assignments` in form of `WorkItems`. The three model elements `OrgUnit`, `User`, and `Group` form a composite pattern which enables modeling of hierarchical org-charts. Cross-connections in the reporting structure can be modeled by adding additional `ModelLinks` representing them.

`User` represents a participant of the project, who can take on units of work. The model element allows to specify the `firstName` and `lastName` for each user, as well as the `email` address as a way of getting in contact with the user.

`Group` model element allows for grouping organizational units as part of a composite pattern. Groups can be used to map to teams,

departments, business units, or companies. They can contain other groups or individual users.

`Meeting` models the attributes of a gathering of project participants to accomplish a stated goal. `Meetings` reflect the scheduled `starttime` and `endtime`, the `location`, the `facilitator`, who is responsible for guiding it to archive its stated goal, the `minutetaker` who notes down the questions, decisions, tasks, and information items arising during the meeting, and the `timekeeper`, who keeps an eye on the clock in order to help the meeting progress smoothly and not get bogged down in details. It also specifies all other `participants` of the meeting, so that the setup of the meeting can be reconstructed. Meetings are segmented into multiple sections (e.g. information exchange, discussion, recap), which can reference relevant `ActionItems` and `Issues`.

Meetings are useful in the context of reviews as a means of scheduling. In order to incorporate `Review` results quickly and following up on the results regularly, the checkup can be linked to a meeting of the project team. Since these occur in regular intervals in short sequence, it allows to schedule them at any point in the development process.

Milestones are specific dates marking the deadline for a project deliverable. A milestone can be the completion of a requirements document, or the delivery of a software feature. In case the `Review` results are closely tied to a project phase, the checkup can be linked to a specific `Milestone`. These checkups occur less frequently than those connected to the meeting anchor, but this approach is often better suitable for projects which follow a more waterfall-like process model.

3.2.2 *Rationale*

Rationale capture can be done in many forms with varying degrees of formality and complexity. In principle any system for capturing rationale can be used for implementing IBMR. For our framework, we chose the version of the rationale management used in RUSE/MUSE (Wolf, 2007; Helming, 2011) shown in Figure 3.4.

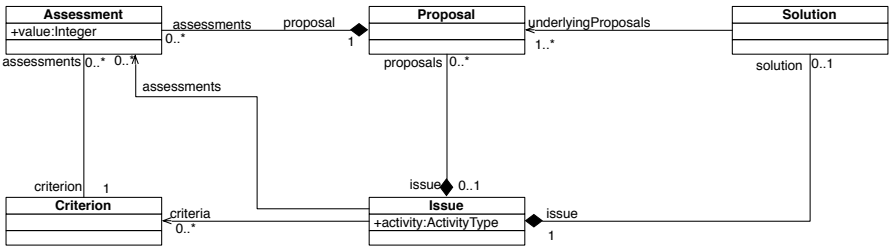

Figure 3.4: Model elements used to capture rationale

`Issues` represent a problem to be solved and are recorded with the `name` of the `Issue` phrased in form of a question. The `Issue`'s `description` can expand upon the question and convey additional

details, while model elements related to the `Issue` can be linked to from the `relatedModelElements` attribute. The `resolved` status of the `Issue` is false (i.e. the `Issue` is open), as long as there are no proposals, or none of the existing proposals have been selected as the resolution to the `Issue`. An `Issue` has an `assignee` in order to assign the accountability for solving the `Issue` to a single and identifiable person. An `Issue` can also have a `dueDate` specified in order to signify the date at which the `Issue` must have been solved. An `Issue` can be closed (`resolved` is true) by selecting a `solution` based on one or a combination of multiple associated `proposals`. A closed `Issue` can be reopened if it has to be reevaluated in the light of new information, new developments, or a change in the project itself. To solve more complex issues, `criteria` for evaluating the different proposals can be specified and rated in form of `assessments`. As with the `ActionItem`, an `Issue` may also track the corresponding `activity` to which it belongs.

`Proposal` symbolizes a suggestion made for solving an issue. Proposals represent depersonalized ideas, following the principles of Fisher and Ury (1981). They have a short summary of the idea as their `name`, with more intricate detail as their `description`, and relevant model elements linked from their `relatedModelElements` attribute. They are related to the `issue` they are meant to solve and are linked to it. They also link to the `assessments` relevant to them.

`Solution` models the result of the decision making process and links the solved `issue` to the solving proposal(s) in the form of `underlyingProposals` for the issue.

`Criterion` expresses a constraint restricting the solution space for a specific issue. They convey individual preferences which may conflict with each other. They are meant to guide the decision-making process by allowing participants to express preferences and emphasis for certain options, i.e. proposals. Decision making is done taking into account a decision matrix, listing the proposals on one axis, the criteria on the other, and the suitability of a given proposal for the relevant criterion in the respective entry of the matrix. As mentioned before, if necessary a number of proposals may be selected to solve a particular issue. The actual ratings of proposals according to criteria, which are expressed in the entries of the matrix are modeled as `assessments` and linked to from each criterion.

`Assessment` rates a `proposal` according to a `criterion`. This is expressed in the form of an integer `value`. Higher valued assessments are better than lower valued ones.

3.2.3 *Review model elements*

This section lists the model elements introduced to incorporate reviews into the RUSE/MUSE software engineering model as shown in Figure 3.5.

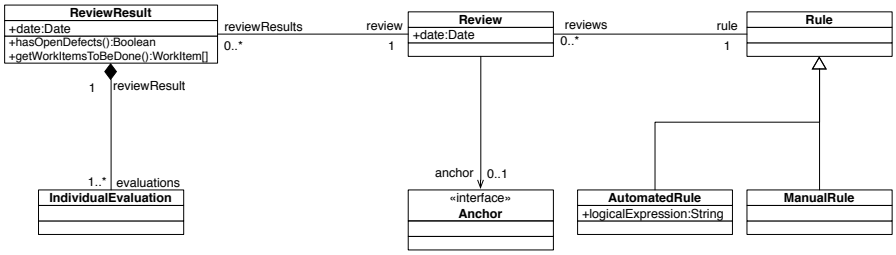

Figure 3.5: Model elements proposed to capture reviews

`Review` bundles a number of `reviewResults` all belonging to the same `rule`. It also reflects the `date` of the `Review` and can specify an `anchor` determining the occasion on which to check if the detected defects were already corrected.

`ReviewResult` is the combined assessment of a set of `reviewers`, if a `modelElement` is valid according to the review's `Rule` at a `date`. It lists the individual `assessments` of the reviewers, and the resulting `workItems`. A `ReviewResult` can convey whether the corresponding model element has unresolved defects (`hasOpenDefects()`), and the work still to be done (`getWorkItemsToBeDone()`).

`IndividualExamination` reflects the assessment of an individual `reviewer` as part of a `reviewResult`. It lists all `suggestedWorkItems` suggested by the reviewer to fix all defects of the model element according to the `Rule` of the review.

Review results are anchored to events in the time-line of a project, such as meetings or milestones. `Anchor` defines the time horizon in which decisions have to be made and work has to be done.

`Rule` describes a criterion according to which the `Review` takes place. This can be either a `NaturalLanguageRule`, which has to be checked by a human, or an `AutomaticRule`, which can be checked by a computer.

`NaturalLanguageRule` can be of several forms. These rules can be specific and detailed checklists to be consulted for each model element (e.g. "which of the following security threats can arise from each component of a given system" followed by a list of possible threats), thus ensuring optimal agreement between reviewers. On the other end of the spectrum, they can also be more like guidelines or even just a common goal, for instance "Let us review this requirements document in order to bring it to a state in which we can show it to the client". These instructions are specified in the `description` of the `NaturalLanguageRule`.

All manual rules have in common that the amount of information specified according to which the `Review` should be done is not sufficient for the creation of machine interpretable rules. A human being is required to check these, and they are therefore by default subjective and might be inconsistent.

`AutomaticRule` models rules which were specified to a level on which a machine can do the checking. There are two disadvantages to this approach. First, the limitations of machine interpretable rules restrict the set of expressible rules. Second, automatic rules presume the existence of suitable tool support, whereas manual rules only depend on the project participants themselves.

3.3 REVIEW TRACEABILITY

One of the main advantages of the IBMR framework is the treatment of reviews as first class citizens in a software engineering model. By treating them in this manner, the capabilities of the unified software engineering model and its suitability for traceability can be used (Bruegge et al., 2006; Wolf, 2007; David et al., 2009a; Helming et al., 2009a; Helming et al., 2009b; David et al., 2009b) .

This integration allows users of the model to get additional information about the context of a model element under review and to configure reviews to precisely target relevant parts of the model according to project specific needs.

3.3.1 *Traceability and review coverage*

Developing software for highly regulated industries (e.g. medical, financial, aviation, or nuclear power plants) requires developers to examine models according to criteria, such as potential hazards, or security concerns in requirements specifications (e.g. Jetley et al., 2006; Croll, 2003). These examinations need to be documented and traceable, guaranteeing that all relevant model elements were inspected according to the regulations. Software controlling a medical device for instance needs to document in which ways a patient could come to harm, when this device is used on them.

Implementing the IBMR framework reduces the cognitive load on the reviewer by determining the subset of the entire model which needs to be reviewed and only showing model elements relevant to the current review. By attaching the result of the review directly to the model element, the maturity (i.e. the number of times it has been successfully reviewed) and correctness of the model can be determined by collecting all work items (action items and issues) connected to a subset or the entirety of the model and determining the amount of open work items out of the set of all connected items. These model

elements can then be flagged by the framework to indicate that they need to be changed in order to be compliant.

3.3.2 *Traceability and change monitoring*

Review traceability also ensures that the model stays compliant even in the face of change. Model elements which have already been reviewed but whose last modification date is later than the date of the last review are no longer considered to be reviewed and are shown with other unreviewed model elements in a summary. This way, the percentage of model elements with up-to-date examinations for compliance with regulations can be calculated. By tracking compliance with IBMR, compliance with multiple regulations can be tracked independently for each regulation, i.e. a model element can be marked as being compliant with one regulation, but not another.

By automatically keeping track of the part of the model not yet reviewed, a subset of the unreviewed model elements can be calculated and flagged as input for review processes, reducing cognitive load even further.

3.3.3 *Traceability and project management*

Furthermore, review traceability allows for better accountability. The creator and the authors of a model element can be determined by looking at the change history of the model element, while all reviewers of a model element can be established by collecting all the assessors of review results.

Connecting review results with open work items, which are attached to upcoming meetings or milestones, helps preparing following up on them. It also keeps the discussion limited, since only open, i.e. still undecided, work items are discussed. At the same time, with this connection, no open work items are forgotten. For example, open issues are included in the discussion section of a meeting, and the status of open action items is covered in the information sharing section of a meeting. Alternatively, open work items are connected to project milestones to indicate that they need to be resolved before this milestone is met. They can be also be included into any project management workflow which uses the unified software engineering model.

3.3.4 *Examples*

In the following two subsections, we describe how IBMR helps with seeing relevant context, and how an IBMR-enabled modeling tool allows for the configuration of reviews.

Context of model elements

Review traceability inherent in the IBMR model allows to discover context information such as the author of a model element, its change history and related collaborators, and even developers with general knowledge about artifacts related to the model element. This makes it easier to determine whom to consult for decisions on the model element, or whom to ask about design tradeoffs. For instance, the author of an interface specification for a subsystem would be the first choice to ask, but if they are not available, an author of the subsystems internal model could maybe provide the necessary input.

Context information can also be useful for improving technical decisions. Since system model elements are not only connected to project model elements such as authors or collaborators, but also to other system model elements, these traces are used to understand a model element and to be able to make a more informed decision about its validity. For instance, a reviewer unsure about the specifics of a requirement under review can look up its associated use case in order to better understand the problem. The reviewer can then conclude that the requirement is correct, and the problem of understanding it was on their own part – or they can decide an improvement to the wording is necessary, if the requirement is ambiguously defined. In the latter case, they can be more specific what needs to be improved, since they are better aware of the context.

Configurability of the review process

Modeling tools based on IBMR allow users to configure which model elements to review, the interval in which a review rule needs to be checked, and whether model elements not compliant to a rule are considered blockers for the project. An IBMR-based tool can also add context information about related model elements as necessary. This context can either be configured in advance by the modeling lead of the project, or machine learning can be used to observe the model elements most navigated to, and to construct the relevant context on the fly.

3.4 TRACKING OF REVIEW-RELATED CHANGES

One of the advantages IBMR offers is the integration of reviews and review results into the project model, allowing for integrated tracking of these results. This provides support for follow-up by integrating periodic checkup on the execution of necessary changes as a project management function. It also helps with change tracking, since changes due to review results can be traced to the review they originated from and to the corresponding discussion if one exists, or

else the task which affected the change. It also enables validation of the model according to specific guidelines or regulations.

3.4.1 *Follow-up on review-related changes*

Having explicit anchors for checking up on the status of defects discovered during reviews makes integration of review results into the project management workflow a lot easier. With IBMR, meetings and milestones are fully integrated into the project model and are therefore the ideal candidates for attaching follow-ups. Milestones are used for follow-ups, if the review results are tied to a specific release or document and have to be completed in time for the milestone. Meetings are used for follow-ups, if review results need to be integrated based on sprints, or if they should be acted upon as quickly as possible.

3.4.2 *Validation*

Expressing review results as explicit model elements allows IBMR-based tools to integrate and reason about them. This can be used to automate compliance checks, give warnings about likely candidates for defects, or quickly check when a model element was last reviewed.

3.4.3 *Integration with model change tracking*

Since the IBMR meta-model supports change tracking on model elements, changes are linked to tasks or discussion, providing more context for a particular change than a plain commit message for a versioning system would. It maintains the chain of traceability by linking to the model element describing the actual work or discussion, which in turn refer to all other model elements relevant in this context. The advantages of such a system where shown in Koegel, 2011.

Model change tracking can also provide a basis for model reviews on commit to the model repository, similar to the ability of certain version control systems to mandate pre-commit reviews for source code.

4

ISSUE-BASED MODEL REVIEW PROCESS

This chapter details IBMR's issue-based review process. It defines categories of review processes and lists their activities and describes how they can be implemented. Furthermore, it outlines how reviews can be implemented as a continuous activity throughout the whole project instead of as discrete events.

ISSUE BASED CAPTURE OF DEFECTS

The purpose of a review is to capture defects in order to reduce the number of defects present in a product (Fagan, 1986).

Managing these defects in the form of lists as suggested by Fagan has a set of drawbacks. First, the list is not directly connected to the artifacts in question, causing traces to break in case artifacts get renamed or reorganized. And while traces from the defects to the artifacts are feasible, the other direction, from artifacts to all related defects, is hard or impossible to trace. Second, while the defect itself is captured, the associated rationale is not, even though it is implicitly available at the time of detection. Because of this, project participants at a later stage in the project have trouble retracing how decisions were made for fixes to defective model elements. Third, when defects are detected during a review, the fix may not be immediately clear, or require discussion and coordination with parties not present during the review.

Issue-based capture of review defects solves these problems in the following way. By linking an issue describing the detected defect to the model element in review, the defect is explicitly traceable to the model element and vice versa. This trace is stable regarding renaming as well as reorganizing and is only broken by the deletion of either the issue, the model element, or the link itself. Also, since the purpose of an issue is to capture argumentation and decisions, rationale capture can be done "at runtime" and is enforced by the structure of the system. Finally, since issues do not always have resolutions or proposals associated with them throughout their life cycle, they are well suited to capturing defects whose solution still needs further investigation and discussion.

For quick in-situ fixes for defects, instead of capturing an issue with only one proposal as its solution, it is preferable to immediately capture the solution to the defect in form of an action item instead. This allows for tracking of progress made on fixing the defects, assigns clear responsibilities for every individual defect found, and al-

lows for integration of these fixes with the rest of the development life-cycle, such as further modeling or implementation tasks. Action items can be included in sprints or iterations, attached to milestones, or just given due dates in order to specify their target.

For example, the review team may determine that a parameter needs to be added to the signature of a key interface of the system. Since all agree and there is no further discussion needed, the easiest way of capturing the defect and the agreed upon solution is to create an action item and to assign it to a suitable candidate. This action item is then added to the current sprint to make the time frame during which it needs to be done explicit and traceable.

4.1 ISSUE-BASED MODEL REVIEW

The following subsections describe the IBMR actors, use cases, and processes. First we describe the actors and use cases that make up an IBMR-based review. Then we describe the possible classifications for different types of defects according to the IBMR framework. The next subsection details possible process variants made up of subsets of the set of review use cases described in the first subsection.

4.1.1 *Review actors*

The IBMR framework defines the following roles for conducting reviews (actors in the review use cases): Review manager, reviewer participant, and knowledge manager.

Review manager

The role of review manager is assumed by anyone intending to have a model or model-subset reviewed. While they can conduct the review themselves, taking on the role as meeting facilitator, they can also yield that duty to a dedicated meeting facilitator and work with them in conceptualizing and planning the review, but take over when actually conducting the review.

Review Participant

The role of review participant or reviewer is assumed by project participants invited to a review, but not responsible for its moderation or for recording the results. Their function is to inspect the model elements presented by the meeting facilitator, check them for defects according to the agreed upon criteria, and relay the observed defects to the knowledge manager. Time permitting, they can also suggest proposals to solve the defects categorized as issues, but that is optional.

Knowledge Manager

The knowledge manager is the review participant responsible for capturing the identified defects and classifying them in form of action items and issues or fix them in-line if a quick and trivial fix is possible. The classification is done according to the heuristics presented in Section 4.1.3.

4.1.2 *Review use cases*

The use case model (Jacobson et al., 1992) of the IBMR framework is shown in Figure 4.1. These use cases can be combined in form of several different processes.

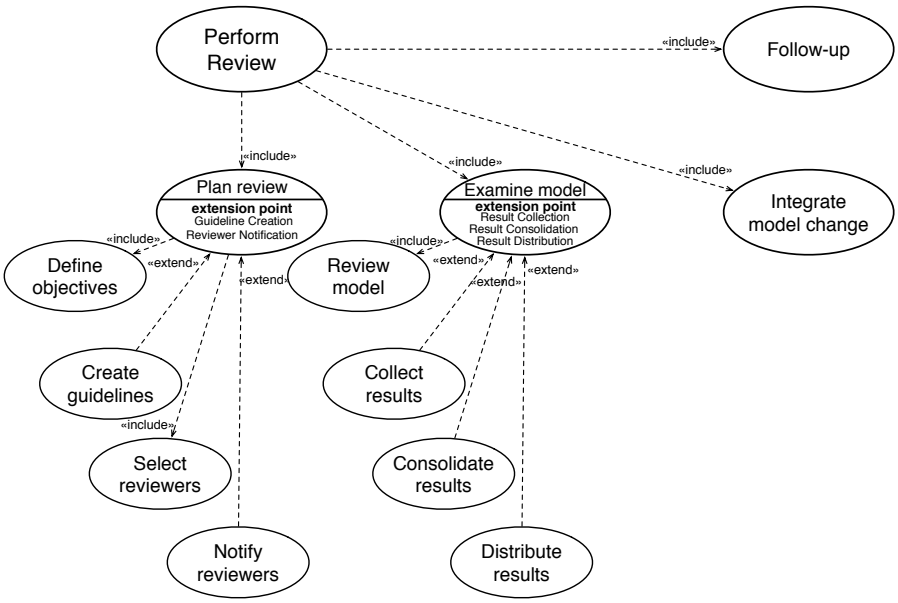

Figure 4.1: Use cases in the Model Review Process

This list of use cases is based on the steps of the inspection process suggested by Wheeler et al. (1996). How the actors in the review process interact with the use cases is shown in Figure 4.2.

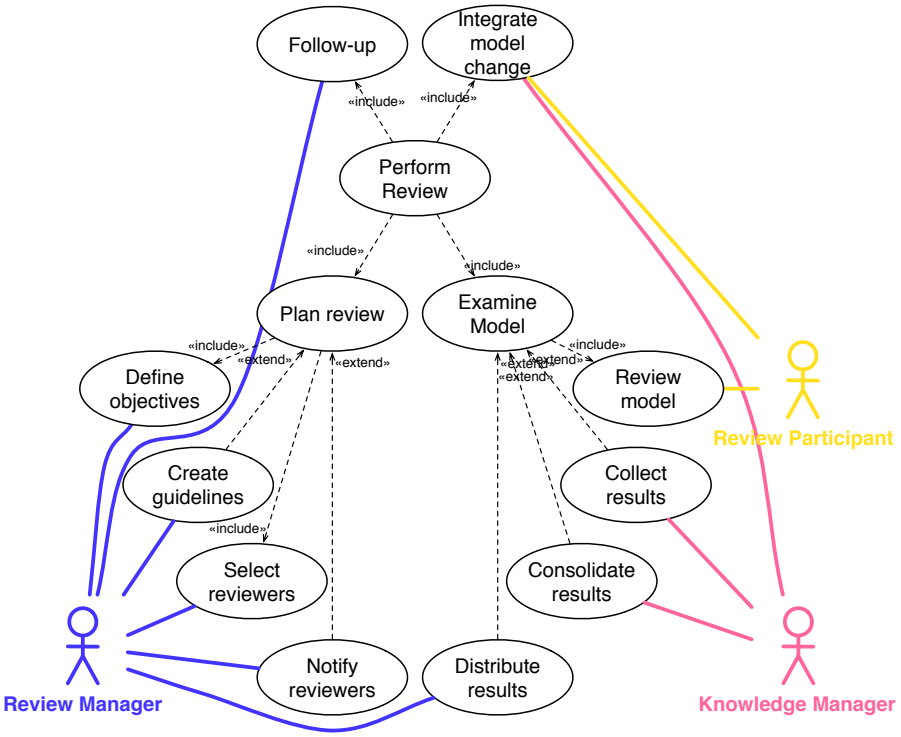

Figure 4.2: Use cases with actors in the Model Review Process

Example scenario

In the following, we will use a hypothetical case of a medical software project developing software to apply specific dosages of radiation to patients as a running example. The activities with swim lanes for the corresponding actors can be seen in Figure 4.3.

The review manager is the person responsible for setting up the review and deciding on the contents, structure, and guidelines of the review. For this scenario, the review manager is the safety engineer responsible for conducting the safety analysis of the control software.

The review manager determines the objective to be reached by the review. Does the review serve as an opportunity to generate general internal feedback on a newly created model or should it serve to finalize a specific subset of an existing model with a client?

In our scenario, the review manager decides that the software model of the project is mature enough to do a real hazard analysis. The objective is to do a risk analysis on the system model as mandated by the FDA (Kamm, 2005).

According to the objective the review manager selects the review participants based on their expertise.

The review manager decides to include the team leader of the software team, the team's software architect, the person responsible for doing the team's software documentation, and an outside consultant specialized in hazard analysis. They schedule the meeting for Friday

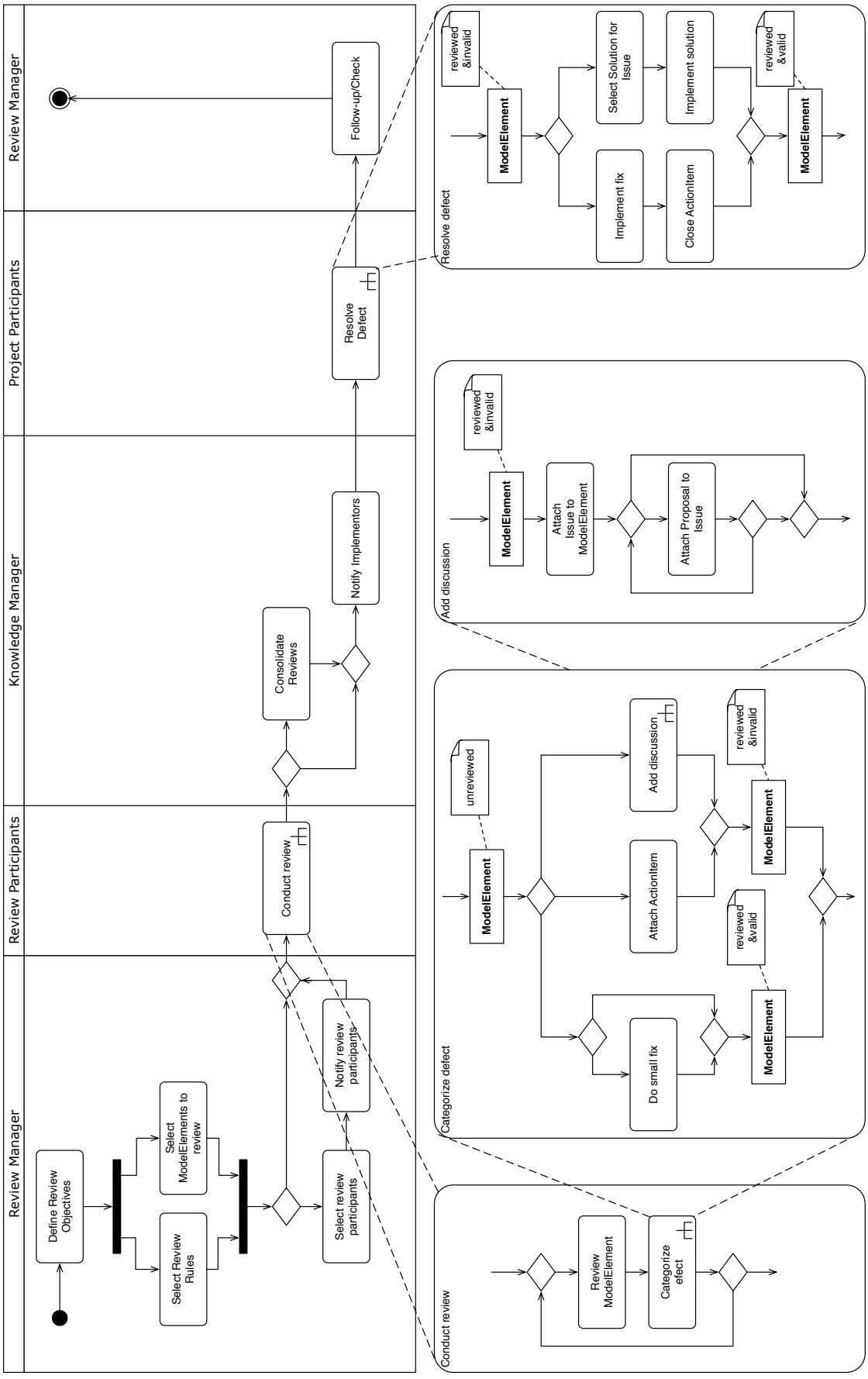

Figure 4.3: Activity Diagram for Model Review with IBMR

the following week, send out invitations and book a conference room, since they decided to hold the meeting face-to-face.

At the specified date, the reviewers examine the artifacts specified by the review manager and give their feedback. This feedback is collected by the knowledge manager; it is then consolidated and sent out as a joint review report by all reviewers.

The following Friday, all review participants gather in the conference room. The review manager takes the role of meeting facilitator, guiding the review team through the meeting. The documentation expert acts as the team's knowledge manager, noting down all hazards the team finds.

Finally, the defects listed in the review report are then fixed and integrated into the existing model.

The relevant use cases for our sample scenario are the following:

Objective definition and guideline creation

Every review is based on the need for an artifact or a set of artifacts to be examined for defects. The actor expressing the need for such an examination is called the review manager. This may be the same person later conducting the review proper (sometimes called the meeting facilitator) but is in fact a separate role. The review manager specifies the objectives of the review, which include the artifact(s) to be reviewed as well as the type of defects to look for primarily. As the review participants should work with a common set of rules or guidelines according to which the review is conducted, the review manager provides such rules, or delegates the creation.

While in practice guideline creation is often done implicitly and ad hoc (Porter and Votta Jr, 1997; Ciolkowski et al., 2003), providing explicit, agreed-upon rules and checklists improve reviews because they lead to less friction and misunderstandings (Laitenberger and DeBaud, 2000). If the rules require no human assessment but can be deterministically checked (e.g. determining whether every hazard does indeed have a mitigation), these natural language rules can provide input for an automated version of the review process described in Section 4.2.2.

Finally, having explicitly stated rules helps reconstruct the details of a review later on, since the criteria they were checked for can be understood. The guideline creation should precede the *reviewer selection* use case since the composition of the review panel depends on what exactly needs to be checked, but it does not necessarily have to happen beforehand.

The result of this use case is a specified objective for conducting the review and a set of corresponding rules (or at least guidelines in case of informal reviews) for the review.

Reviewer selection

This use case determines the composition of the review panel and is comprised of multiple use cases. It is initiated by the actor "Review Manager". The first included use case is deciding which other participants are needed for the review, or if the review manager has the expertise and capability to do the review on their own. If so, they can do the review according to the review process described as Individual Review in the Review processes section.

If more input is needed, the review manager needs to determine the extra skills needed and select reviewers accordingly. The review manager needs to determine which technical expertise is needed, if independent input (from outside the team) is needed, and whether participants of the customer side should be included (and if so, which ones - end users, technical experts, or management). If no external input is needed, the review manager can choose to hold a Team Review.

If external input is needed, the review manager can schedule a Project Review to solicit outside input. Both team and project reviews are described in the Review processes section.

As described in Figure 4.1.2, a knowledge manager is responsible for taking down the results of the actual review. Since the review manager ideally should not also be the knowledge manager (Doyle and Straus, 1982), the reviewer selection should also include the assignment of a review participant to the role of the actor "Knowledge Manager" for that review.

It is also possible to allow the review team to be entirely self-selected, if more than one reviewer is needed but no specific composition of the reviewer panel is required. In this case, only the number of necessary reviewers is specified.

This use case results in a participant list for the review.

Reviewer notification

If the *reviewer selection* use case resulted in a participants list including more participants than only the review manager themselves, and the reviewer selection process is not entirely done as self-selection, the next use case is *reviewer notification*.

For the *reviewer notification* use case, the review manager contacts all reviewers and provides them with an agenda for the review, containing date, time, location, review guidelines, and the list of participants and their roles. This is necessary to ensure all participants can take part in the review and should at least a day, ideally a week in advance (Doyle and Straus, 1982).

This use case results in a set of notifications to the review participants.

Examine model

The *examine model* use case and its variants are described in more detail in the section on Review processes.

Results collection, consolidation, & distribution

In case multiple participants took notes during the review and there is no tool support for automatic integration, the notes need to be collected and consolidated.

The extending *collection* use case involves bringing all notes together and sorting through them. The knowledge manager checks with each review participant for their notes of the review and gathers them into a common repository.

For the extending *consolidation* use case the knowledge manager checks which defects are duplicates and create an exhaustive list of unique action items and tasks to be followed up on.

The results of the review need to be stored for later reference, and review participants must be given access to them for verifying them and following up on tasks and decisions. With IBRM tool support, the knowledge manager can send the consolidated results to the review manager who then distributes them to all participants.

This use case results in a consolidated list of action items and issues, describing defects found during the review.

Model change integration

The integration of model changes and the removal of defects is the principal reason for doing reviews in the first place. Therefore, it is important to track the process of implementing these changes. Action items can be closed if a task is done, and issues can be set to resolved if there is agreement on a particular resolution. This way, project managers can use the traceability links to follow up on the results of the review.

This use case results in fixes for defects found during the review being applied to the model.

Follow-up

To track the progress of model change integration, the review manager incorporates the state of work items resulting from a review into upcoming project or team meetings. This can be done to only share information (e.g. reporting on closed action items), report blockers (e.g. problems with resolving action items), or to discuss open issues.

This use case results in the discussion of work items regarding fixes of defects in upcoming team meetings.

IBMR framework support

Figure 4.4 shows how the IBMR framework supports both the review manager and the knowledge manager by assisting them with some use cases, while taking care of some use cases entirely.

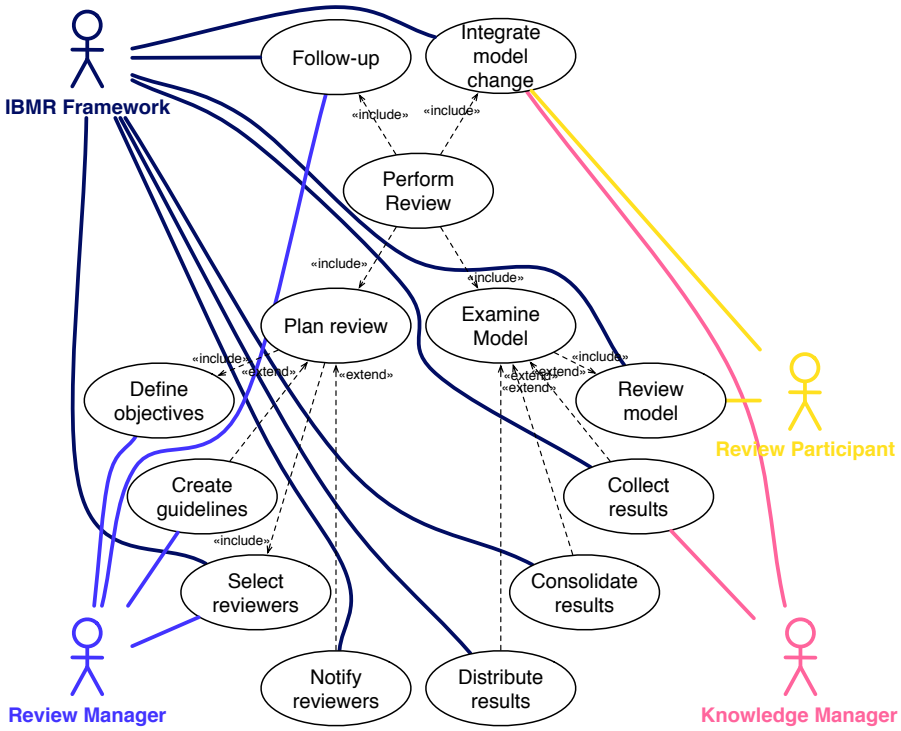

Figure 4.4: Use cases with actors in the model review process with IBMR framework support

This figure shows how IBMR supports the actors during the review process. The *consolidate results* use case is now handled by the framework instead of the knowledge manager. Similarly, the *notify reviewers* and *distribute results* use cases are handled by the framework instead of the review manager.

Additionally, the framework now assists the review manager with the *select reviewers* and *follow-up* use cases, and the knowledge manager with the *collect results* and *integrate model change* use cases. The assistance for the *integrate model change* use case also benefits the review participant, who is also assisted with the *review model* use case.

4.1.3 *Classification of defects in the IBMR framework*

In this section, we will give examples on how the IBRM meta-model can be used to note down review results. Defects identified during a review can be classified into the following categories:

Simple fix The fix to correct the defect is simple enough to be applied immediately. This could be correcting a spelling error or creating a simple association between two classes.

Complex fix The model element under review needs correction, but this correction is too complex to be done on the fly. However, the fix is clearly defined and does not require further discussion.

Discussion required The model element under review has a defect, but the correction is both complex and non-obvious. Further discussion is required and the problem (and possible proposals for solving it) needs to be documented.

The review status of a model element is determined by the status of the action items and reviews attached to them (c.f. Figure 4.5).

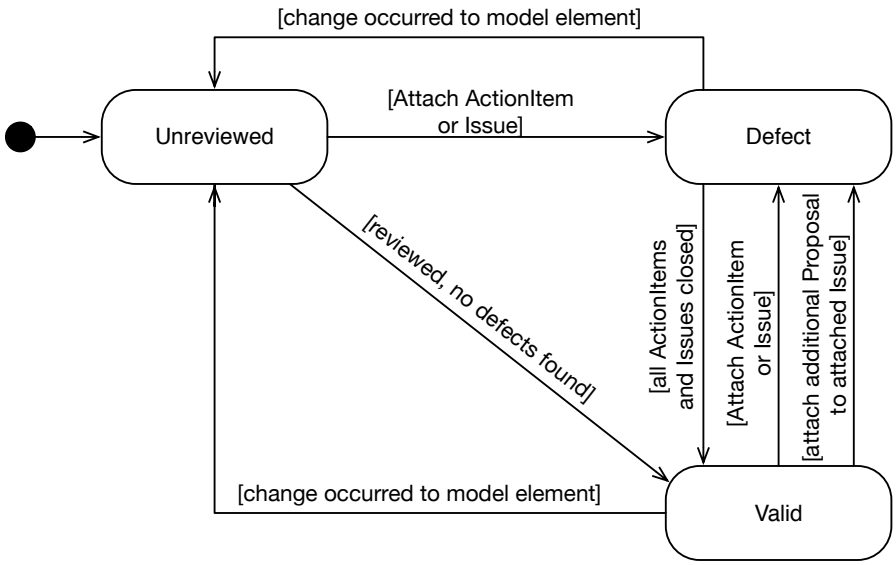

Figure 4.5: IBMR model element review states

The following subsections will describe the steps that document the review results for each type of fix.

Simple fix (or no fix)

In order to review a model element requiring either a fix which can be immediately applied (or no fix at all) and is therefore not deemed necessary to being documented (e.g. fixing a typo), the review participants examine the model element (Step 1 - Figure 4.6) and decide there is no need to document any necessary changes to this model element. If necessary, a change is done in-place by the knowledge manager.

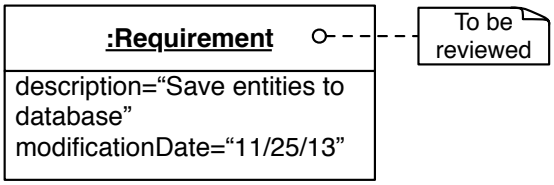

Figure 4.6: No fix or simple fix - Step 1

In order to mark the model element as reviewed, the knowledge manager attaches a review result with the current date to the model element (Step 2 - Figure 4.7).

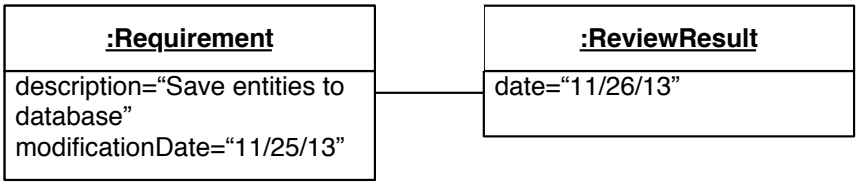

Figure 4.7: No fix or simple fix - Step 2

Since no further modification is necessary, there is no need for follow-up work. The model element is now considered reviewed, as long as it is not modified again after that, i.e. the date of the review result is more current than the modification date of the model element.

Complex fix

In order to review a model element requiring a complex fix, the review participants examine the model element, in our example an instance of Danger (Step 1 - Figure 4.8) and decide that a fix is needed to correct the model element and that the instructions for this fix need to be documented.

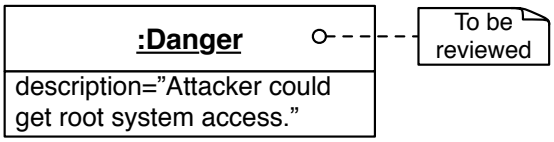

Figure 4.8: Complex fix step 1

To document the instructions for the fix, the knowledge manager attaches a review result with the current date to the model element,

and attaches an action item containing a description of the fix to be done to that (Step 2 - Figure 4.9). At the time of creation, this action item is unresolved, indicating that the fix is not yet applied. In our example, the knowledge manager notes that the Danger model element does not have any associated mitigations.

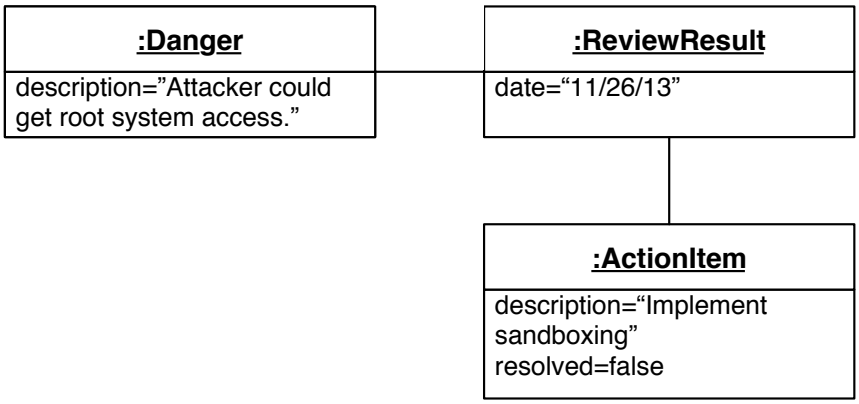

Figure 4.9: Complex fix step 2

To fix the defect of the model element, a review participant later processes the action item. In our example, a Mitigation model element is added to mitigate the Danger. The review participant then resolves the action item, indicating the work has been performed (Step 3 - Figure 4.10).

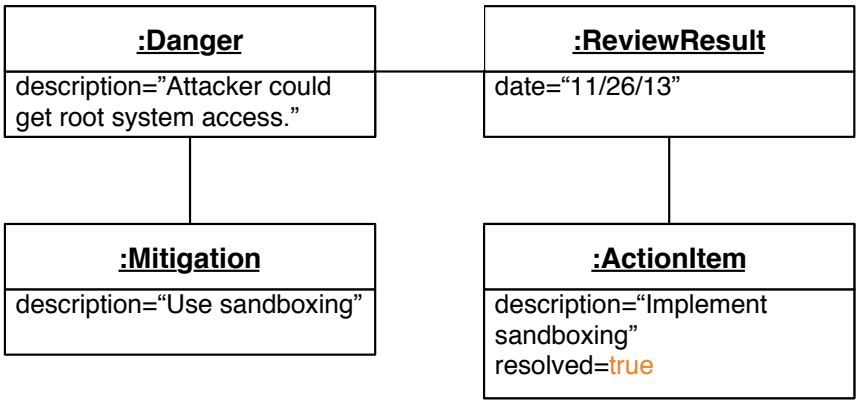

Figure 4.10: Complex fix step 3

The defect documented by the review result is now fixed. However, since this entailed changes to the model element, it is necessary for it to be reviewed again in order to ensure that no new defects were introduced and mitigate against regressions. We call this a *regression review*.

Discussion required

In the case of a model element requiring a fix which involves a discussion, the review participants examine the model element, for this example an instance of Requirement (Step 1 - Figure 4.11) and decide that a discussion is needed to correct the model element, and that the discussion needs to be documented.

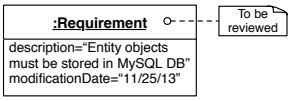

Figure 4.11: Discussion required step 1

As in the previous cases, the knowledge manager attaches a ReviewResult model element to document the outcome of the review and attaches an Issue, describing the question that needs to be solve to the ReviewResult (Step 2 - Figure 4.12). Initially the Issue is unresolved, indicating that the discussion has not been conducted successfully yet.

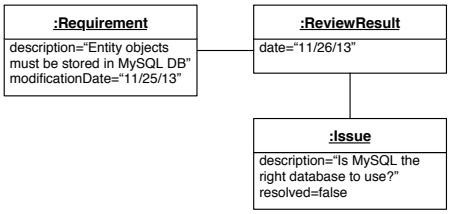

Figure 4.12: Discussion required step 2

Next (Step 3 - Figure 4.13), review participants can add Proposals to the unresolved Issue. In our example, two competing Proposals p1 and p2 are added. This documents the discussion to solve the Issue and fixing a defect on the original Requirement.

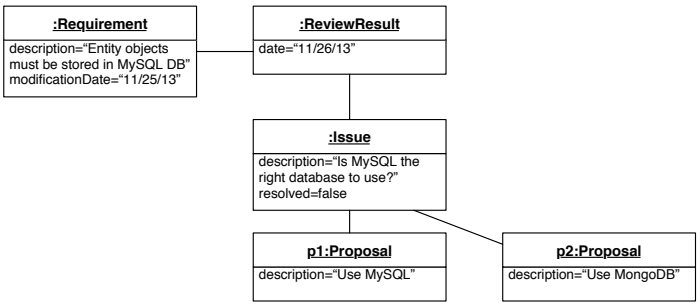

Figure 4.13: Discussion required step 3

In the final step (Figure 4.14), a Solution is added to resolve the issue. The solution is linked to the Issue it solves, as well as to the underlying Proposals. In our case, the Solution is based on Proposal p1. The original Requirement is modified according to the decision captured in the Solution.

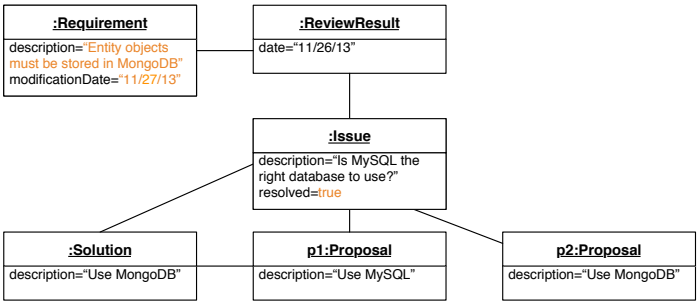

Figure 4.14: Discussion required step 4

The discussion initiated by the Review Result is now concluded, a Solution has been found, and the according modifications have been made. As before, this may require a regression review, since the modification entailed changes to the model element, and it needs to be ensured that no new defects were introduced.

4.1.4 *Review processes*

Based on the underlying meta-model presented in Chapter 3, IBMR allows for a variety of different review processes. The simplest process are the Individual Review, and the Continuous Review processes. These processes are especially suited for agile projects.

The following sections detail three review processes suitable for use in iterative and incremental software development processes such as the Rational Unified Process (Kruchten, 2004), or agile processes. They are targeted at reviews that occur during specific milestones of a project (e.g. at the end of an iteration), or as preparation for these. For a way of conducting reviews as a continuous workflow during the entire project, see Section 4.3.1

The first of the three review processes is the Individual Review (IR), the smallest unit of review. It allows for a single project participant to use IBMR when reviewing model elements on their own.

The second is the Team Review (TR), which is used to guide the review of models by teams in the setting of a team meeting. The TR can be used to ensure the whole team reviews project artifacts and ensures a higher degree of quality.

The third is the Project Review, which (PR) is centered on the discussion and review of project artifacts by the client or end-user. This could be the discussion of a model in early stages of a project with a technically skilled client or the demonstration of a prototype to the intended end user. The PR helps to capture client feedback and make it actionable.

Besides these manual review processes, there is also the possibility to automatically check for the compliance to specified rules. This is called Automated Review (AR) and is used for checking rules which can be specified as logical expressions. These automated rules are executed on every model change, and the modeler is alerted whenever such a rule is violated. This variant is discussed in Section 4.2.2.

Individual Review

The Individual Review is targeted at project participants who review parts of the model or the whole model. While this examination of model elements can help detect defects, it is not as effective as group-based review techniques, such as inspection. Also, regulations or quality assurance mechanisms often require review by more than one person.

The Individual Review starts with the decision of a project participant to review a model. The first step is to determine a subset of the model, which can be grouped by type (all model elements of type Requirement), document (all model elements belonging to the System Design Document), time range (all model elements created or modified in the last month), or another suitable configuration.

The second step is to specify or select the rules according to which the review is performed. Phrasing them succinctly and unambiguously is important, since they will be used to provide a checklist which is visible to the reviewer throughout the review. Besides this brief rule summary, each rule should also have a longer description, in which the intent of the rule is stated for reference. In case of confusion, this description should provide the necessary information to proceed.

The next step is the actual review itself. Here, each model element of the subset selected for review is presented to the reviewer. The reviewer is also presented with a checklist of the rules according to which the review is to be done and with an interactive summary of the model element's context.

The reviewer checks the model element for compliance to the rules in the checklist. If the model element conforms to the rule, the reviewer marks the model element as compliant to this rule. In case the model element violates the rule in some degree, the reviewer has to decide whether an immediate fix is possible. If so, they apply the fix and mark the model element as compliant. If the reviewer cannot make the fix immediately, the model element needs to be marked as defect and the further course of action needs to be specified.

In case the reviewer can come up with specific instructions on how to solve the defect without the need for discussion, they can attach an action item specifying the respective instructions to the model element. This action item needs to be marked as done in order for the model element to be marked as compliant.

If the reviewer cannot come up with specific instructions for fixing the defect, the problems needs to be discussed with other project participants. For this purpose, the reviewer attaches an issue to the model element under review, characterizing the issue and possible proposals to solve it. Again, the model element cannot be marked as compliant, unless the issue has been solved. The last step is done for each model element to be reviewed. After all of them have been checked, the review is completed.

The system keeps track of the model elements already reviewed as well as changes to reviewed model elements which were changed in a way that potentially invalidates the review.

The system also keeps a list of all reviews done for a model element. This allows others to see when and by whom a model element was last reviewed, and according to which criteria.

This process is meant to either accompany other types of reviews for the continuous review process described in Section 4.3.1, or to be used in smaller projects where all reviewers have some expertise in every part of the project. It can be combined with agile methods such as Scrum, where a product owner can review the models produced by the Scrum team, such as the requirements model, or in case they are technically competent, the analysis or design model.

Team Review

The Team Review process differs from the IR in that a group of people is needed to conduct it. There are two different variants, one is collocated and role-based, the other is distributed and peer-to-peer.

In the collocated, role-based version, people in the review team take on different roles. The meeting facilitator manages the review, specifying which part of the model is to be reviewed, determining and inviting the persons relevant for the review, conducting the actual review itself, and following up on the outcome of the review.

The knowledge manager of the review is responsible for capturing the outcome of the review. This is best done by a dedicated person,

so that the rest of the team can concentrate on the task at hand (Doyle and Straus, 1982). The knowledge manager marks model elements as reviewed and if necessary attaches action items or issues to them. For issues especially, the knowledge manager takes care of noting down all arguments made, persisting all proposals that were made during the discussion, the relevant criteria, as well as a possible solution. The rest of the group is made up of reviewers, participants whose task it is to inspect the model elements under review and to contribute to the discussion.

The author of the model element under review can give the group insights into the model element's rationale. When discussing model elements with the author present, special care has to be taken to decouple people from positions, as described in Fisher and Ury (1981).

The process of a Team Review is as follows. The team examines the model elements and discusses the implications, possibly leading to multiple, conflicting proposals for solving an issue. But although a group decision is behind it, at the end there can be only one outcome, which is captured by the knowledge manager.

The distributed, peer-to-peer process has a different focus. Here, the goal is to get multiple opinions on the model element under review without the need for a special meeting which is attended by all participants. The review is initiated by a project participant who wants to gather feedback on a subset of the model. They specify the subset of the model which should be reviewed and the project participants they are interested in gathering feedback from. Instead of performing a complete review, the team members each give individual assessments, which need to be consolidated. The knowledge manager collects all assessments and integrates them. This results in one review for each model element, from the consolidation of the team members' individual assessments.

Project Review

The Project Review (PR) differs from the other reviews in that the audience of the review is comprised of developers as well as clients and customers. It also has a different focus, concentrating on client visible artifacts, such as high-level models (e.g. deployment diagrams or software architecture models), prototypes, demonstrations, and presentations. As in the TR, one participant plays the role of facilitator, while another plays the role of knowledge manager. Because of the size of the review meeting, comments and issues are often uttered in quick succession, making them difficult to capture. Here, tool support is important to make capturing comments as quick as possible.

During the review, artifacts for which feedback is desired are presented to the clients and customers and their feedback is captured by the knowledge manager. Sometimes discussions about solutions arise, however the purpose of the review is not to already identify

solutions, but rather for gathering feedback and aligning points of view.

The facilitator defines the subset of the model to be reviewed. Ideally a review would encompass the entire model, but in practice some restrictions may be necessary. Parts of the model may not be finished enough for review, or key stakeholders may not have much time to spare and therefore the part of the model presented to them has to be carefully chosen. The selection of the subset should be done in close collaboration with the rest of the team, in order to ensure they receive the feedback they need most.

After establishing the model subset to be reviewed the facilitator should inform the team about the choice of date and review content. This enables the team to do a preliminary team-internal or personal reviews (see previous sections) in order to improve the quality of the review's content, which improves the results of the project review. The team also needs to make sure that the models reflect the latest state of design the team agreed upon and are consistent. Especially in large scale projects with lots of inexperienced developers, the models diverge from the actual state of planned design, since beginners sometimes have trouble understanding the value of models.

In the case of student projects, which are limited to one semester, students never need to revisit the models anymore since the course is then over. The usefulness of good documentation therefore is not readily apparent to students. Internal pre-review checks allow students to compensate for this and fix defects that they themselves notice.

With the model subset having been chosen, it is now necessary to fix the rules for the review and to decide upon the focus of the review. Is syntactical correctness the most important aspect, or completeness of the model, or compliance with regulations of the application domain? These expectations need to be communicated and should be part of the review agenda in order to ensure that all participants are working towards the same goal. The facilitator then sends an invitation to all review participants, containing time and date, location, purpose of the review, and chosen model subset, to allow participants to prepare accordingly. This concludes the first phase of the project review process.

To prepare for the review, the facilitator has to ensure that the review location is suitable and everybody has access to the review content. Review content can be distributed in multiple ways, e.g. as paper printouts, or via digital projection. A shared screen ideally should be used for IBMR because in this case tool support can be used to guarantee that the model reviewed reflects the latest state available. Also, presenting the review content directly in the tool allows for quick in-situ fixes of small mistakes. Jointly viewing the review content directly on a computer screen, while possible for small groups, should

be avoided, since it is not ideal for discussion purposes. The facilitator also needs to determine a knowledge manager, who is tasked with classifying and capturing the review results as described in Section 4.1.3. The rest of the team is encouraged to take their notes, since, especially in discussions, it is helpful to capture as many view points as possible in order to have a complete picture of the discussion. These notes need to be integrated with the main review protocol by the knowledge manager.

The review itself is conducted in the following way: the facilitator presents the review content to be examined and moderates the discussion. They ensure that all people present can participate and are allowed to comment. They also make sure the review stays focused and discussions are kept short and to the point. It is enough that a point for discussion is identified and captured, so that it can be discussed at a later point in time. The knowledge manager captures the discussion results in the following format:

If a model element is determined to be correct and complete by all participants, it should be marked as reviewed in order to ensure nothing gets reviewed twice. Simple changes that everybody agrees on should ideally be done immediately. The knowledge manager quickly makes the change and the facilitator refreshes the content to reflect the change. If direct change is not possible, the facilitator keeps a list of small changes to be included in a single tracking ActionItem. This way, a large number of small changes do not pollute the list of resulting ActionItems, while still being traceable. All these changes should be treated atomic and be done all at once or not at all in order to ensure consistency.

Changes that all participants agree upon, but which cannot be implemented within a minute need to be captured as ActionItems. The knowledge manager notes down the change to be make in appropriate detail and links it to the model element(s) to be changed. They also link the ActionItem to the Review itself in order to keep track of all decisions undertaken in the review.

If a discussion takes place, the knowledge manager captures this according to the rationale model described in Section 3.2.2. The knowledge manager notes down the point of discussion in form of a question as an Issue (e.g. “Which encryption algorithm are we going to use to encrypt this data?”). The solutions which were discussed are denoted as proposals, sketching the idea behind them. Finally, if there is a discussion, criteria for decision making which were proposed can also be captured as part of the issue-based discussion capture. This way, all alternatives presented in a concluded discussion are preserved and can be revisited at a later point. For discussions which could not be completed, this way of capturing allows taking up the discussion at the point it was suspended and preserve all proposed ideas.

The knowledge manager captures everything visibly to all review participants in order to allow them to see whether misunderstandings occurred or something was overlooked. If this is not possible, the knowledge manager is encouraged to restate the decisions captured to facilitate consensus about the results. Time permitting the knowledge manager restates the major findings at the end of the review. This concludes the second phase of the project review process.

Finally, the knowledge manager consolidates the captured feedback if necessary and incorporates notes gathered by other participants where appropriate. They in turn distribute the review results to all project participants. The facilitator ensures that the ActionItems and Issues are followed up upon. They place them on the agenda for the following team meetings and make sure that the respective assignees do their tasks in the specified time frame.

4.2 REVIEW PROCESS AUTOMATION

By having reviews and related concepts as first-class citizens of the project model, several checks, such as the presence of mitigations for each hazard modeled in the system, can be automated. By automation we mean that formal rules can be specified which a tool can use to monitor the model and to alert the users if these rules are violated. Setting up such an automated checking system consists of three parts: first, a way of specifying the rules to be checked, second, a method for automatically applying checks to the model under development, and third, a means of scheduling these automated reviews to happen either at predefined times or intervals or depending on changes to the model.

4.2.1 *Specifying rules*

Providing review rules as explicit elements of the model offers various benefits. First, the usage of explicit rules for reviews is encouraged, instead of just doing ad hoc reviewing of the model. Second, not only are rules used, they are also consistently stored, can be integrated into a review rules database spanning multiple projects, and can be reused. Finally, if rules can be specified in a machine interpretable format (e.g. OCL), tool support can automate these checks and take some of the effort required away from human reviewers. Also, automatic checks can be done continuously, ensuring all model elements are in compliance with automatic rules at all times.

4.2.2 *Automated reviews*

For rules specified in a machine interpretable format, Automated Reviews (AR) can be conducted with appropriate tool support. The first

area where tool support can help is with keeping track of the review status of model elements. The tool can keep track of the review status via the trace from the model element to the review model element. By comparing the modification date of the model element with the time of the latest review, the tool can flag unreviewed model elements and filter out reviewed model elements from other views. This is important in projects with safety or security compliance requirements, that mandate reviews of the system model (*CLSI GP31-A*, 2009). The tooling ensures that unreviewed model elements (i.e. model elements which were changed after their last review, or model elements which have never been reviewed at all) can be found and re-checked.

Tool support can also assist developers by taking care of some simple but repetitive checks. For instance, reviewing the project model for ActionItems without assignees can be automated by specifying an AutomatedRule. This rule is a boolean condition on the attributes and relations of a model element which can be checked by the tool. The tool flags model elements not in compliance with specific rules and presents them in a defects view. This frees the developers' time for other tasks such checking for semantic correctness, completeness, which are less suitable for a machine.

4.2.3 *Scheduling*

With information about project planning available in the model, it is possible to support automatic creation of reviews before important milestones. This way the project manager is supported in their task of review preparation, since the model contains all other meetings as well and can provide input on when best to schedule a milestone review.

Partitioning of the model into chunks of manageable size according to review rules in order to not overexert reviewers can also be automated. This can be done by the class of model element, by the package they are contained in, by creation/modification date, or by author.

Since the model is fully traceable, issues related to defects are automatically correlated with the appropriate teams, and decision making are anchored to suitable meetings and milestones as described by Helming, 2011.

4.3 REVIEW PROCESS VARIATIONS AND IMPROVEMENTS

The following subsections present variations and improvements on the basic review process described in Section 4.1.4.

First, we introduce the notion of Continuous Review, a review process implementing reviews as a continuous activity similar to a workflow in the Unified Process (Kruchten, 2004). Second, we describe a

Concurrent Review process, allowing for automated integration of review results by many different reviewers reviewing in parallel. Finally, we list a number of smaller process variations.

4.3.1 *Continuous Review*

Large formal meetings are not necessarily the most efficient way of conducting reviews, as shown by Votta Jr (1993). Often the list of participants of a particular review is not well thought out, and valuable time is wasted, since larger meetings are often less efficient (Slater, 1958). Votta for example suggests replacing meetings with *Depositions*, three-person meetings with only the author, the reviewer and the moderator present.

Also, meetings with a large number of participants lead to delays, since a suitable time for a common appointment has to be found. Sometimes even the completion of milestones or project phases has to be accomplished before the meeting can take place. This means that work either builds upon unreviewed models or has to be delayed until a review could be performed. Either case is far from ideal, so reviews should be driven by the content to be reviewed, and not just by time and milestones.

With the IBMR framework and its support for change tracking on the model, continuous review is possible. Continuous review can be used to make reviewing a continuous activity throughout the software development life-cycle, a workflow in the terminology of the Unified Process. A tool implementing the meta-model presented in Chapter 3 keeps track of the review status of all model-elements, allowing reviews to be done at any time and at any level of granularity. A reviewer can review an entire document, or only a particular diagram, one particular class of model elements, or even a singular model element on its own. The framework then tracks the reviewer, the defects found, and the criteria according to which the review was done.

While reviews in Continuous Review can take place at any point in time during software development and are done without convening special meetings, this does not mean that there is no place for review meetings anymore. Similar to Votta's depositions, we still suggest reviews featuring either the client or outside consulting experts as reviewers, the project team as authors, and the team leader (or project manager) as moderator. Ideally such a large review would be broken up into many reviews with a single reviewer and a single author at a time.

The person in charge of model quality can define a set of review rules at the beginning of a project. Whenever new model elements are created, the framework immediately flags them as pending review. After a developer reviews them and fixes them as necessary, they are

flagged as reviewed and do not show up with the unreviewed model elements. Any changes to the model element (or the underlying review rule according to which the review was conducted) lead to the model element being flagged as pending review again.

With Continuous Review, reviews can be conducted when it best fits the reviewer, but the overall state of the quality of the model and the degree to which it has been reviewed can be quickly obtained by filtering for all model elements which the IBMR framework has flagged as pending review.

4.3.2 *Concurrent Review*

The concurrent review consists of four main activities.

In the *preparation* activity, the reviewer determines the basic parameters of the review. Parameters include the model elements to be reviewed, what defects to focus on and criteria for correctness and completeness, the participants of the review (and whether it is co-located or distributed), and the date and time it is done. In this activity the people responsible for the quality of the model elements should also take the time to check for syntactical correctness and for consistency in order to ensure that the review does not get bogged down in finding trivialities. The model to be reviewed needs to be of the highest quality possible beforehand for difficult to detect defects to be found. The preparation activity is mainly the responsibility of the facilitator of the review, as well as of the persons responsible for the documents to be reviewed.

In the *review* activity, the participants of the review check the documents selected in the preparation activity according to the criteria determined in that activity. The facilitator guides the review, checks whether the participants digress, and moderates between different opinions. The knowledge manager serves as the group memory, capturing the outcome of the review according to the model described in Chapter 3. The rest of the review participants (though usually not clients if they are participating) are actively encouraged to take their own notes as sources of additional information. The model elements to review are broken down into manageable chunks which are read in the group and then discussed.

In order to streamline the process, a variant of planning poker can be used. Participants are handed a notepad and an index card showing a **o** on one side and a **?** on the other. Participants then examine the model elements, making notes of issues they want to discuss, and finally hold up their card after they are finished reviewing. If they have no remarks on the current chunk of the model, they hold up **o**, if they want to mention or discuss something, they hold up **?**. The group then only discusses model elements for which at least one participant has signaled questions (**?**). In our evaluation (cf. Chapter 5),

this worked to efficiently review a large number of model elements in one session.

In the *post review* activity, the knowledge manager consolidates their own results with those of the other participants and compiles a unified review. The review consists of a list of all action items and all issues found during the review. A review protocol then is distributed to all stakeholders, who are asked for their feedback regarding correctness and completeness. Once the review protocol is agreed upon, the work on the action items and issues can start.

In the *follow-up* activity, the facilitator checks whether the captured defects are actually acted on and closed. For clearly defined tasks, this means following up on the status of the according action items. For more complex problems, it entails scheduling time for discussing the relevant issues, e.g. in a team or work group meeting. In general, most action items should be closed until the next meeting, in order to not impede progress. Issues should be discussed as early as possible and only if they cannot be solved at that point in time, be postponed to a later date.

4.3.3 *Other variations*

This section sketches small variants which can be combined with the major processes described above.

For larger projects with bigger models, it is beneficial to assign people to be responsible for parts of the model. With each person only dealing with a small subset of the model, the complexity of the overall model can be reduced.

For projects with traceability requirements or compliance criteria, the four-eyes principle can be employed. By ensuring every model element was reviewed by two independent reviewers, the number of overlooked defects can be reduced.

For projects in the medical, financial, or government sector, a formal change management process may be required. IBMR can support this by not allowing changes to model elements which were already reviewed and baselined but requiring approval by a change advisory board.

Finally, we sketch how the IBMR framework can be used to conduct reviews asynchronously and in a distributed manner.

Model coordinators

In some projects, there are specific persons responsible for particular documents. This allows for assigning clear responsibilities and makes it explicit whom to contact in case of questions. This person is also responsible for scheduling reviews and tracking the state of completion of the document(s) assigned to them.

This variation can be used in big projects, where technical expertise about the models is not evenly distributed, instead there are specialists for different parts of the application. In this scenario, model coordinators can be appointed to act as custodians for their part of the model, distributing the responsibility across multiple shoulders. This variation can be combined with any of the other variations, as it is orthogonal to them.

Four eyes

For security or safety critical projects, a four eyes principle for reviews can be established. In this case, two reviews of two independent reviewers are needed in order for a model element to be considered reviewed and correct.

This variation can be used to increase the effectiveness of reviews in for projects where later change or defects in the final products would be even more problematic than usual. Examples for these sorts of projects are medical software, where undetected defects may cause harm to patients, or financial software, where undetected defects may cause significant financial losses.

Formal change management

For projects requiring a stricter process, the system can support formal change management (ITIL, 2016). With formal change management, fixes to model elements cannot be immediately applied, but the change itself is subject to a review by the change management board. Only after they have signed off on it, it can be applied to the model.

This variation can be used if the project setting requires formal change management. This is usually stipulated in the contract and is standard for many government contracts.

Asynchronous reviews

Reviews can also be done asynchronously. Review participants can prepare their comments individually, and later integrate their findings and discuss issues - either face to face or distributed as well. This distributes responsibilities more evenly, since every review participant needs to review the model independently of the others and come up with their own findings. By doing reviews in this manner, the individual contributions can be seen, motivating everyone to contribute more. The disadvantage of this is that it is missing group dynamics (Shaw, 1971). Reviewing and discussing together leads to a shared understanding of the models and can identify additional defects.

Distributed reviews

Distributed, but synchronous meetings are also possible. Video conferencing tools such as Skype or Google Hangouts allow for simulating face to face meeting. Face to face meetings generally provide for a better experience (Bos et al., 2002), although measures may be introduced to mitigate this to an extent (Nguyen and Canny, 2009).

4.3.4 *Meta-reviews and process improvements*

Outside of the scope of this thesis, but definitely a suitable application of IBMR, is review of reviews. Since reviews are model elements themselves, IBMR can be used to review them and find defects or opportunities for improvement. This can be used for continuous improvement of the tailored review process itself, much in the way of Kaizen (改善), the Japanese term for continuously improving existing processes over time, as described by Imai, 1986.

5 CASE STUDIES

To determine the feasibility of the IBMR framework, we conducted two case studies, the second of which was followed by a survey.

The goal was to study the effects of the IBMR framework on reviews. By employing the framework in two case studies in two large scale university student projects, each with an industrial client, we concentrated on the feasibility of the approach. The first case study was conducted in the DOLLI 6 project held in winter semester of 2012/13 (*DOLLI 6 Project Page* 2012), the second in the iOS project in the summer semester of 2013 (*iOS Praktikum 2013 Project Page* 2013).

The first case study covered a project using the SLPC++ software development model (Bruegge et al., 2011), with reviews scheduled after each major phase (analysis, system design, object design, and after each sprint in the development phase at the end).

The second case study covered a project using the Tornado software development model (Bruegge et al., 2012) with more document-centered reviews, focusing on the review of the requirements analysis document and the system design document.

5.1 BACKGROUND

As both evaluations used the "case study" research method, we first define the concept. We then provide an overview of the tooling used in both case study projects.

5.1.1 *Research method*

The research method of case studies belongs to the field of qualitative research and is a descriptive method. We employed this method in an exploratory way.

Yin (2014) defines a case study as an "empirical inquiry that investigates a contemporary phenomenon (the 'case') in depth and within its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident."

Case studies describe cases aimed at exploring the results of the hypothesis and falsifying it in case the hypothesis is incorrect (Popper, 1934). In the following two case studies, we describe the observations made by using IBMR in two projects, and report on the students' reaction to and impressions of usage of the IBMR framework.

5.1.2 *Tooling*

Because of external constraints, we did not use a unified modeling tool in case study projects but had to work with a simplified version of the IBMR model. We used the commercial bug-tracking and Wiki solutions Jira[1] and Confluence[2] for project communication and documentation. The quality of the multiuser support, the ease of use, and the smooth transition between project phases, all principal factors in selecting a suitable CASE tool according to Finnigan et al. (2000), were taken into account. Although the underlying model of Jira is not expressive enough for the IBMR model to be mapped onto in a one-to-one manner, we were able create a basic representation of the main IBMR components in Jira.

5.2 DOLLI6

The goal of the first case study was to test out the feasibility of the IBMR framework, the acceptance by the students, the customer satisfaction level, and possible improvement of the quality of the models produced in the project. In order to gain an initial understanding of the general impact and potential benefits of the IBMR process, we decided to use the large-scale student project DOLLI6 as a field experiment for this case study.

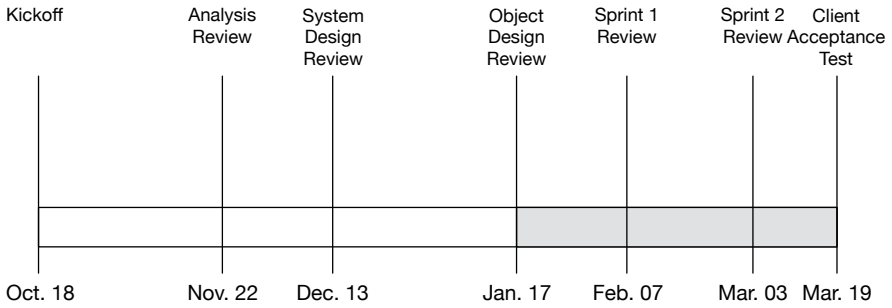

Figure 5.1: Time line of the DOLLI6 project

The DOLLI 6 project lasted from October of 2012 until March of 2013 and was organized as one of a series of client oriented large-scale project courses offered by the Chair for applied Software Engineering at the Technische Universität München for bachelor and master students. The goal of these project courses was to provide students with realistic project environments in which they learn software engineering, starting from requirements analysis with the customer, to conceptualizing and modeling the solution, implementing it, and finally delivering it to the client. The chair cooperated with a large

1 https://www.atlassian.com/software/jira
2 https://www.atlassian.com/software/confluence

industrial partner, the Flughafen München GmbH (FMG), the operating company of the Munich airport. The Flughafen München GmbH is an enterprise with about 7000 employees and an IT department of about 180, with an IT specific revenue of about 15 million EUR.

We ran a case study to evaluate the feasibility of the IBMR framework as part of the DOLLI 6 project.

The airport operates a suite of systems called the Airport Process Management (APM) suite. The APM deals with airplane data, such as schedules, parking positions, and location data, passenger data received from the airlines, ground handling operation data, and much more. All systems follow a common architecture and communicate via a CORBA[3]-based bus. The DOLLI projects had to be integrated into this software landscape. While the first DOLLI was more of a greenfield engineering project, just receiving data from the APM systems, but not integrating tightly with them or conforming to their architecture, the later DOLLI projects in general, and DOLLI 6 in particular, were meant to produce results which could be productized and then integrated into the APM, requiring the developed systems to conform to the general architecture and to meet the security regulations and development guidelines set forth by the airport IT.

For each project in the DOLLI series, the FMG provided a number of specific problems the students needed to solve in the course of one semester. For the DOLLI 6 project, they stated two distinct problems: the first was to develop a system for fetching, analyzing, consolidating, and forwarding IATA messages, a standard originally developed to exchange data by teletypewriters in the 1920's and later standardized by the International Air Transport Association to allow airports and airlines to exchange data about weather, flights, passengers, baggage, and delays. The task was to build components for fetching the messages from a server, parsing and persisting them, building a rules engine capable of deciding which message to forward to which target, and a user interface allowing for manual checking and correction of incorrectly transmitted messages. The name for this system was IATAM, short for IATA Messages.

The second problem was to develop a mobile application and corresponding back-end server for the ground handling services division AeroGround to enable pilots and crew members to request buses and other services to their aircraft before landing. This system was designated MUCS, short for MUC Services.[4] The idea was to make an iOS app which could be used by the airline personnel to send requests to the ground service team, enabling faster turnaround times and more efficient use of resources. For the purposes of the project, the scope of services was limited to crew buses only, but the system was to be designed to be extensible to other services as well. The iOS app should

3 Common Object Request Broker Architecture
4 MUC is the IATA code for the Munich airport

support creation of requests and modifications of existing requests as well as status updates via push for every modification relevant for the end user. The server component should handle communication with the mobile devices as well as the Apple iOS push services on the one side, and the AeroGround dispatching systems on the other.

In order to accomplish this, 7 teams were formed out of the 59 students who signed up for the project. The IATAM sub-project consisted of 4 teams, corresponding to the 4 main components of the system, analyzing, data management, flight consolidation, and user interface. The MUCS sub-project consisted of 3 team, app, account manager, and server.

For collaboration and communication purposes, the students used an infrastructure built around Confluence and Jira. Jira was extended to support a simplified version of the IBMR model. Each of the teams was guided by a coach, an experienced student well versed in such kinds of projects at the chair. Supervising these coaches were two instructors, both doctoral candidates at the chair and both with previous knowledge and experience in managing such projects – each responsible for one sub-project.

The students followed the process described in Bruegge et al. (2011) and covered all relevant software engineering activities, starting with analysis, followed by system design, object design, and implementation in the form of three sprints (c.f. Figure 5.1). The software lifecycle model used was based on a combination of the Rational Unified Process and Scrum, with the first part of the project (November until February) following a RUP like process with multiple concurrent work flows, based on the phases described in Bruegge and Dutoit (2009)

The students worked part time (about 10h/week) at the university during the first phase of the project. For the second phase (March), the project switched to Scrum. Then the students worked co-located with the client at the client's site full-time for 2.5 weeks. During the 22 weeks, the students worked on their problems and presented their results in the form of multiple reviews.

For preparing, conducting, and following up on the reviews, the IBMR process was used. The first review was performed on November 22nd 2012, the students conducted an Analysis Review on November 29th 2012, a System Design Review on December 13th 2012, and an Object Design Review on January 17th 2013, all with client participation. The final results were presented to the technical contacts at FMG on March 11th 2013. The students gathered open issues stored in the JIRA database to be addressed in the reviews. During and after the reviews, they recorded the action items and issues in Jira. Finally, they used the recorded action items and issues to follow up on the results of the reviews.

5.2.1 *Case Study*

The results on how the students applied the IBMR model and process in their project work is described in the following subsections.

Analysis Status Review (internal)

For checking the results of the requirements elicitation and early analysis, the teams conducted an internal review on November 22nd 2012. The participants of this internal review consisted of the instructors, the teams, and their coaches. The goal of this review was to synchronize the project participants with the results of the other teams, familiarize them with the review process, and provide a first check for the consistency and completeness of the requirements, in order to improve their quality for the external review with the customer the following week. From the feedback obtained this review, the developers in the 4 IATAM teams captured 34 ActionItems and 6 Issues, reflecting many clearly defined improvements to be done and a few questions to be resolved. All ActionItems and Issues were later followed up on and resolved.

The developers in the 3 MUCS teams captured 24 ActionItems and 6 Issues. Of those 24 ActionItems, however, 6 were duplicates in the sense that 6 developers had to do the same task, which was recorded individually for each participant. Otherwise, there were no anomalies in the captured items. All work items of the MUCS teams were followed up on as well.

One observation was that the most feedback was given on the most detailed parts of the presentation, which means that a presentation format as opposed to the format described in Section 4.1 is more dependent on the presenter and does not cover all parts of the model equally.

Using issues to capture the discussion worked well, participants were able to attach discussions and proposals to the respective parts of the model.

Analysis Status Review (external)

On November 29th 2012, the status of the requirements elicitation and analysis was reviewed in a larger project setting with the client present. Each team showed their results in a ten-minute presentation, which was followed with 5 min Q&A, with optional follow-up afterwards, so the feedback was more compressed and had to be captured more quickly. The feedback for the IATAM teams resulted in 22 ActionItems and 16 Issues. One team still had trouble following the process and again only attached a PDF and had to be reminded to enter ActionItems and Issues directly. The high number of Issues reflects the results of feedback from the client, which is typical for the anal-

ysis phase of this type of software project. They lead to the need for further clarification of aspects of the system, e.g. requirements or end user involvement. The MUCS teams captured no ActionItems during this review but captured 6 new requirements instead. Additionally, they also captured 7 Issues. 2 of these Issues remained open, the rest was later resolved.

System Design Review

The purpose of the System Design Review was to evaluate the proposed system design, and how they the design fit into the overall architecture of the APM. Since the results of the two sub-projects were intended to be used in practice after completion of the project course, the quality and suitability of the architecture were the focus here. The setting for this review was similar to the External Analysis Status Review, with additional technical experts from the client side present in addition to domain experts. The results of this review were as follows: the IATAM teams recorded a total of 4 ActionItems and 13 Issues, with no notable complications. 1 ActionItem and 2 Issues remained unclosed - although the state of the final software shows that they were implicitly solved, a more explicit solution is missing. For the System Design Review, the MUCS teams captured 2 ActionItems and 4 Issues, and additionally 4 new requirements. Of these ActionItems and Issues, all but one, which was a repeat of the unsolved Issues of the previous review, were solved. Overall, the client was satisfied with the state of the model and the proposed design.

Object Design Review

The Object Design Review was again an internal review, consisting of the same participants as in the first Analysis Status Review. In addition to a revised architecture, the focus of this review was on the quality of object-orientation in the models and the use of patterns. For the IATAM teams, 11 ActionItems and 4 Issues were recorded, reflecting more need for directed changes and less for new discussion. 1 new requirement for error handling was also identified during this session. All ActionItems and Issues were resolved in time. The MUCS teams recorded 11 ActionItems and 8 Issues, although it has to be noted that 7 of the ActionItems reflected the same task for multiple participants, resulting in a total of 5 unique ActionItems. It is also worth to note, that one of the MUCS teams, which recorded 5 of the Issues, did not resolve them explicitly. The rest of these work items were all resolved correctly.

Sprint Review

The Sprint Review (for legacy reasons sometimes also called the System Integration Review) was the last technical review, followed by

the Client Acceptance Test. The CAT in the case of our project courses consists mainly of management being shown the results of the project and a large, hands-on demo portion of developed systems. The Sprint Review was done at the clients site and consisted mainly of technical personnel from the client, the instructors, and the project teams. In the Sprint Review, the IATAM team recorded 15 ActionItems and 2 Issues, and 2 new Requirements. All but 1 ActionItem and 1 Issue were resolved. The MUCS team recorded 13 ActionItems and 1 Issue, and additionally 16 new requirements. The newly captured requirements were mostly refinements of existing ones, providing clearer descriptions of all required functionality. Two of the three teams managed to resolve all their ActionItems and Issues, whereas one team did not resolve any and left 7 ActionItems unresolved because of time constraints.

Client Acceptance Test

Due to the limitations of the student project, the client acceptance test was not designed to elicit specific feedback, but more as a presentation of the work that was accomplished. Nonetheless, the client expressed full satisfaction at the results which were archived, both in terms of the running system, as well as the underlying design and integration into the APM.

5.2.2 *Findings*

This case study showed the general feasibility of the approach. The teams quickly learned to capture action items and issues and to follow up on them.

All teams used the issue-based model review approach to track tasks and open questions. This allowed the doctoral student managing the project more insight into the teams' progress than was possible in previous years without the approach.

Overall, there were 1897 action items and 479 issued captured overall during the duration of the project. Out of these only 101 action items and 50 issues remained unresolved at the end of the project.

This resulted in about 8 issues per developer, which is an improvement to previous student projects of a similar scope, also deploying rationale management. For example, Wolf (2007) reports only 2 issues per developer in a project employing RUSE, but not IBMR.

5.2.3 *Limitations*

Because of the nature of our large-scale student projects, tool choices were constrained by ease of use and real-world applicability. This naturally precluded research tools, which cannot be as optimized for

usability as commercial tools can be, and whose knowledge does not provide a competitive advantage later on in the workplace. Therefore, the IBMR framework has to be implemented in existing commercially applicable tools, in our case Confluence and Jira. While this has limitations regarding the expressiveness of the underlying model (cf. Section 3.2, it does provide the benefit of showing how the framework can be used in a commercial setting, without the need for the introduction of new and unpolished tooling.

Extensive reviews of entire documents consume a lot of time. If this is not possible for some reason, time can be saved by letting the participants examine the document on their own, make their own notes, and in the meeting just consolidate and discuss the items. This, however, results in an entirely different group dynamic.

The reviews conducted as part of this case study were done in the setting of team meetings and of review meetings with customer participation.

5.3 NTT DATA

The goal of the second case study was to focus more on one individual team instead of a large project and to observe and compare it to other teams doing similar work. For this case study, we chose the iOS13 practical course, which was conducted at the Chair for applied Software Engineering at TUM. This course offers students the opportunity to do application development for Apple's iOS platform. It features industrial clients posing real problems to the students. The course ran from April 2013 to July 2013 with 89 students and 10 participating companies. The industrial clients ranged from large, international companies such as Audi, B/S/H, and Siemens, to small startups like kisi[5] and Jamie Jacobs[6]. Each industrial client provided the students with a problem description, containing the basic requirements, necessary interfaces to other systems if applicable, and in some cases a draft of a system architecture. The students were tasked to elicit a complete set of requirements, analyze them, and model the system accordingly. After this, they implemented a mobile application, and for some projects a corresponding server component as well. Each of the teams were supported by a doctoral student working as an advisor to the team, and a more experienced student (selected from courses of previous years) to act as a coach.

5 http://www.getkisi.com
6 https://jaimiejacobs.com

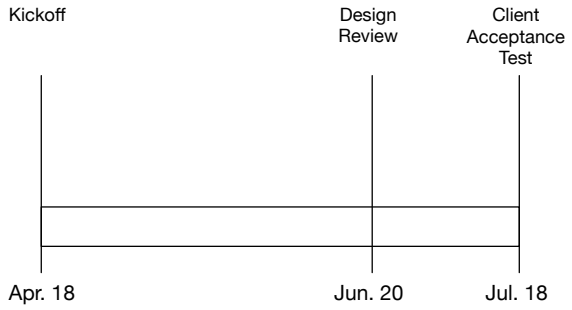

Figure 5.2: Time line of the iOS13 project

In order to closely observe the impact of the IBMR framework, we decided to select one team and observe it throughout the entire course. We chose the NTT Data team, because the doctoral student serving as its advisor was already familiar with the IBMR framework. NTT Data is an IT services and consulting company with more than 60.000 employees worldwide, and a revenue of 15 billion USD. It is a subsidiary of NTT (Nippon Telegraph and Telephone), a Japanese telecommunications company. The German branch of NTT Data operates an office in Munich, which has a strong focus on the automotive industry and works closely with BMW.

For the iOS13 course, NTT Data posed the following problem. Consultants from NTT Data frequently have to travel from their offices to their clients' offices, which are about a 20 minute walk away. In order to provide a quicker mode of transportation, NTT Data opted for purchasing a fleet of pedelecs (electronic bikes), since public transport between these places is bad, the parking situation is not much better, and normal bikes would be problematic, especially in summer, because of sweating. NTT Data therefore wanted a mobile application for finding the closest pedelecs, showing their range (the state of charge of the bike), make reservations for pedelecs, and also transferring these reservations to other users. Additionally, the usability of the app was a strong focus of NTT Data's, requiring the app to be simple and quick to use, and light and modern looking.

A team of eight, one coach and seven developers, was chosen to solve this problem. Out of the eight students, three were undergraduate students, while five were graduate students pursuing their master's degree. All except one of the bachelor students had a computer science major, only one had games engineering as major. Since the students came from different countries, the team's language was English. The gender distribution was uneven, with six male and two female students. The students were new to the iOS platform and most were also not experienced in real world projects.

5.3.1 *Case study*

The NTT Data team followed an agile approach (cf. Tornado model in Bruegge et al., 2012), with the project duration partitioned into 5 sprints, an initial sprint capturing user stories and experimenting with mockups, 3 sprints focused on the development of the app and the creation of the corresponding documentation, and a project finalization sprint with the goal of polishing the app and finalizing the documentation as well.

Sprint 0 ran from April 25th until May 16th. During this time, the team first had to perform team building exercises to get to know each other and to learn how to work as a team. This was followed by an initial customer meeting, during which the client presented the problem in more detail and the developers were able to ask questions and work out first drafts of the corresponding user stories. They also developed sets of UI mockups, which they created in teams of two. These four sets of mockups the team then presented to the client, eliciting feedback and incorporating that feedback into the final mockup which served as the first potentially shippable product increment. The team was also introduced to the tools and processes used in this project during this period. They were presented with a process for rationale management, the IBMR framework for model review, and Confluence and Jira, the tools used for executing these processes.

Sprint 1 ran from May 17th until June 6th. In this time period, the team began development, starting with the most important user stories, and working their way down to the less important ones. They also started to work on the Requirements Analysis Document (RAD), defining the purpose and scope of the system, the objectives and success criteria, the functional and non-functional requirements, the user stories, and the final UI mockups developed in Sprint 0.

In Sprint 2 (June 7th - June 27th), the team continued to work on the app, implementing additional user stories. In addition, the team reviewed the RAD according to the team review process described in Section 4.1.4. After finishing the team-internal review, they presented the RAD to the client for feedback. At the same time, the team started working on the System Design Document (SDD), detailing design goals, subsystem decomposition, hardware/software mapping, persistent data management, access control and security, global software control, and boundary conditions. The most recent status of both documents was also presented in the Design Review on June 20th. For the presentations as for all previous feedback sessions and reviews, the IBMR framework was used to capture and follow up on all defects. After a short period of acclimatization, the developers were able to employ IBMR without introducing any delay to the actual review process, and were able to capture all detected defects in the observed

reviews. These defects were captured as work items and were dealt with as part of the sprint.

Sprint 3 lasted from June 28th until July 11th. The objective of this sprint was to implement as many of the remaining user stories as possible, and to improve on the SDD. Again, the team first did a team-internal review of the SDD, before presenting it to the customer for additional feedback.

In the project finalization sprint (July 12th - July 18th), the team spent a week polishing the app and testing it thoroughly. They also compiled final versions of the RAD and SDD and prepared them for delivery. Finally, they prepared a final presentation to the client for the Client Acceptance test on July 18th.

The client expressed their complete satisfaction with the results, the app and server component, as well as the underlying models and designs.

5.3.2 *Findings*

During the course of the project, the team captured a total of 385 action items and 86 issues.

Out of the 86 issues, 84 were closed - only 2 remained open. Of the 85 closed issues, 80 were resolved with a solution, and only 5 rejected. On average, 1.62 proposals were added to each issue[7] (for the distribution, see Table 5.1). This indicated that the issue-based approach was able to be successfully employed to facilitate tracking and discussions of questions arising during the project.

Proposals per issue	Number of issues
1	43
2	14
3	12
4	2
5	1

Table 5.1: Proposals per issue

On average, 9.9 days were needed to resolve an issue, with a median of 7.4 days. The issue resolution time ranged widely, from hours up to 42 days, with a standard deviation of 9.4 days.

The vast majority of all solutions to the issues used only one proposal. In 13% of cases however, more than one proposal was used, but never more than 3 (c.f. Table 5.2). Out of these 9 solutions, 5 used all proposals as solution, while 4 only selected a subset.

7 Due to a technical problem, for 8 of the resolved issues no solution was recorded. Since the system should have not allowed for this state to occur, they are omitted from the calculations.

Proposals per solution	Number of issues
1	63
2	6
3	3
4	0
5	0

Table 5.2: Proposals per solution

Only 12 issues had due dates set, indicating that the team perceived progress on these issues as sufficient without.

The number of issues per developer was 10.8, a further improvement on the results of the previous case study.

This deeper involvement with discussing and resolving open questions on the model may have been the cause of the significantly higher satisfaction of the team with the results of the team's documentation in comparison to the other teams (see the survey in the following section).

Tracking and resolving action items was not as successful as tracking and resolving issues. Out of 385 action items 154 remained unclosed. This indicates that our representation of the model in Jira - while manageable enough for complex tasks like discussing issues - was too elaborate for tracking action items. Especially two team members did not use the tool at all to track their action items (only 2 out of 80 items were touched). These two team members accounts for over 50% of unclosed action items (78 out of 154).

5.3.3 *Limitations*

The iOS 13 case study was limited to one team of 7 developers and was conducted in a university setting. While in this second case study more data on the usage of issues and the time taken to solve them was gathered, this still only offers anecdotal evidence on the suitability of IBMR for these kind of projects.

To further support this evidence, we followed this up by a survey of all project participants, to determine if there were any differences in the perception of the review process between the team which followed IMBR and those which did not.

5.4 SURVEY

At the end of the iOS13 course, we surveyed all developers. We asked for their experiences and evaluations of their respective review processes. From 89 developers, 34 filled in the entire survey (a response

rate of 38%). From the 8 members of the NTT Data team, 5 answered all questions (a response rate of 62%).

5.4.1 *Survey results*

In the following, we present the results of the survey. The survey indicates that the complexity of the review process did not seem increased for the team using IBMR. However, the team using IBMR was more certain of having incorporated all feedback into the model, and of having substantially improved documentation. This observation was corroborated by the instructors of the teams.

Complexity

In order to assess how the complexity of the IBMR process employed by the NTT Data team was perceived in comparison to the ad hoc review process used by the other teams, we asked the participants to rate on a 5-point Likert scale, how complex they perceived their review process to be. Figure 5.3 shows the distribution of the levels for the NTT Data team and the remaining teams.

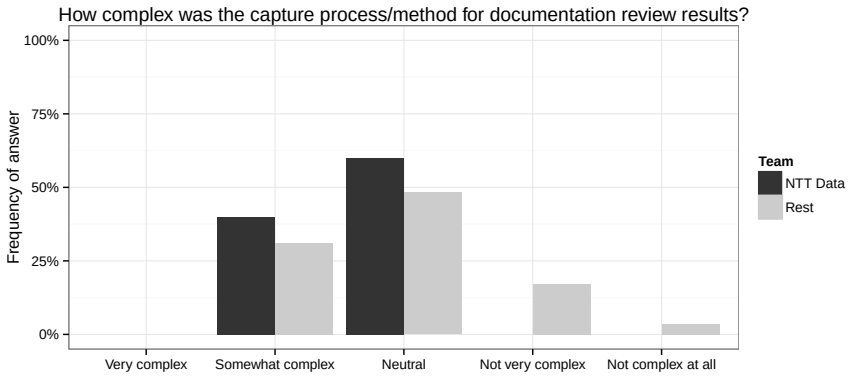

Figure 5.3: Perceived complexity of the review process

This shows that while the perceived complexity of the IBMR process was slightly higher than the ad hoc process, none of the participants rated the IBMR process as very complex, and the majority found the complexity to be neutral. This indicates that while the process seems more complex on paper, the actual effort to implement IBMR is not that much higher than implementing ad hoc reviews .

Improvement

The developers were also asked to evaluate the improvement of the reviewed documentation and models on a 5-point Likert scale (cf. Figure 5.4).

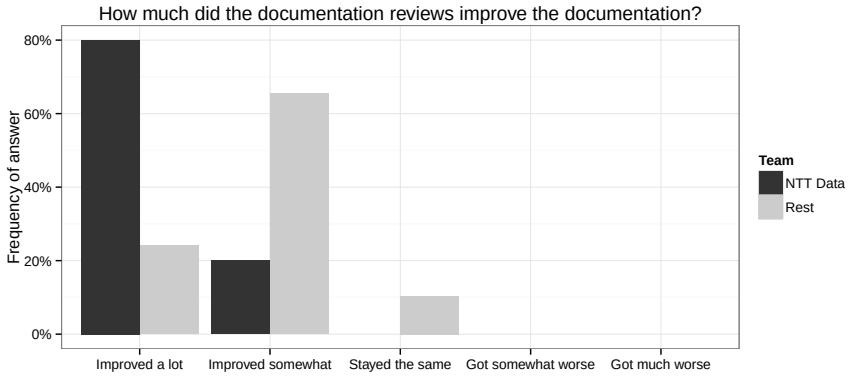

Figure 5.4: Improvement to documentation

The resulting distribution indicates the NTT Data team rated their documentation "improved a lot", compared to the remaining teams, who only perceived some improvement. Since all teams were told to thoroughly review their documentation and that this was part of their grade, they had a similar motivation to do well in this regard. We conclude that IBMR lead to better developer satisfaction with documentation quality.

Incorporation

Another question asked in the survey was about the developers' confidence of having addressed all relevant feedback and incorporated it into the documentation. Figure 5.5 shows the relative frequency of answers on a 5-point Likert scale for both IBMR and ad hoc reviews.

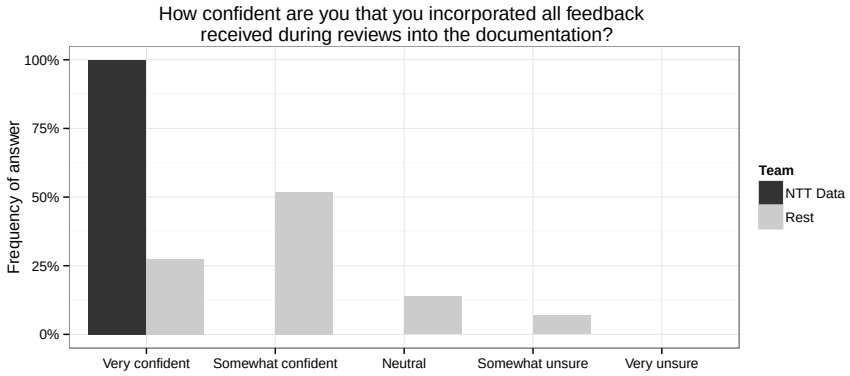

Figure 5.5: Incorporation of feedback

While developers following the ad hoc review process were at least somewhat confident that they incorporated all feedback, developers following the IBMR framework were all very confident that they did in fact incorporate all given feedback. This seems to indicate that the improved capture and follow-up process of IBMR indeed improves execution of changes due to feedback.

Document quality

The developers were also asked to assess the quality of the documentation, in particular the Requirements Analysis Document (RAD) and the System Design Document (SDD) on a 5-point Likert scale. The results for the RAD is shown in Figure 5.6, the results for the SDD in Figure 5.7.

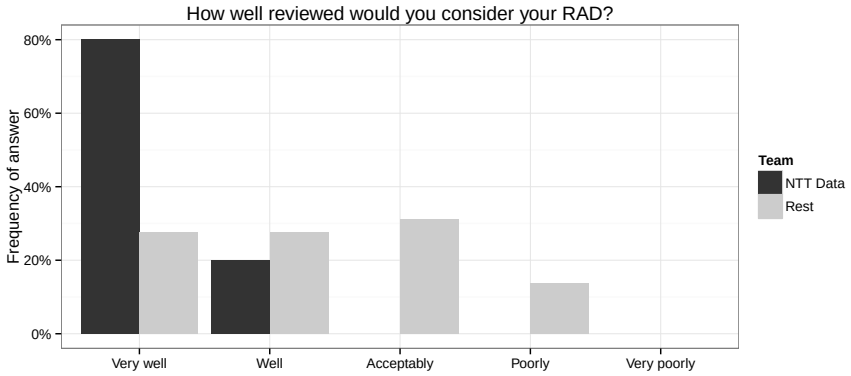

Figure 5.6: Quality of RAD

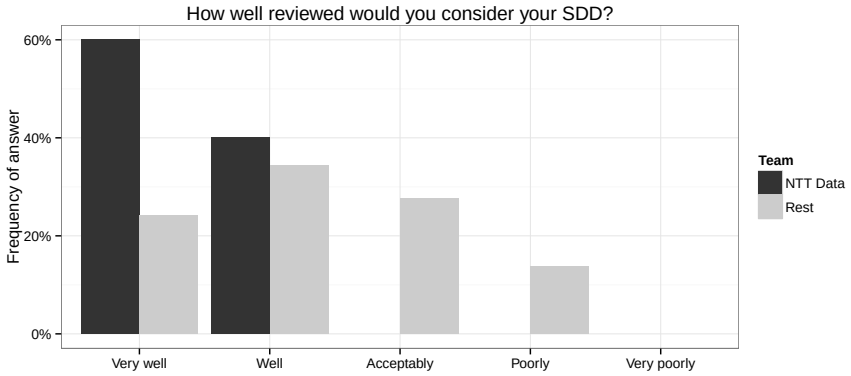

Figure 5.7: Quality of SDD

These figures reflect the improved sense of documentation quality for the developers following the IBMR framework in comparison to the ad hoc approach.

Personal benefit

Since student projects often suffer from the problem that students do not directly benefit from the documentation and thus do not mind having low documentation quality (Ahtee and Poranen, 2009), we wanted to see if the IBMR framework raises the perceived personal benefit from documentation. The results are shown in Figure 5.8.

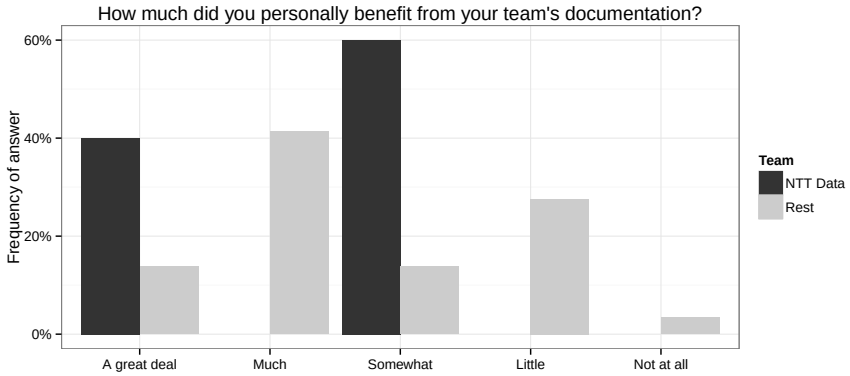

Figure 5.8: Personal benefit of documentation

The result is inconclusive. On the one hand, while the number of participants claiming they benefited "A great deal" from the documentation is higher for the IBMR framework, the majority for IBMR felt they only benefited "somewhat" from the documentation, in contrast to "much", for the comparison group. On the other hand, no participants using IBMR claimed little or no benefit at all.

Value of reviews

Finally, we wanted the students to rate the value of reviews to them on a 5-point Likert scale. The results are shown in Figure 5.9.

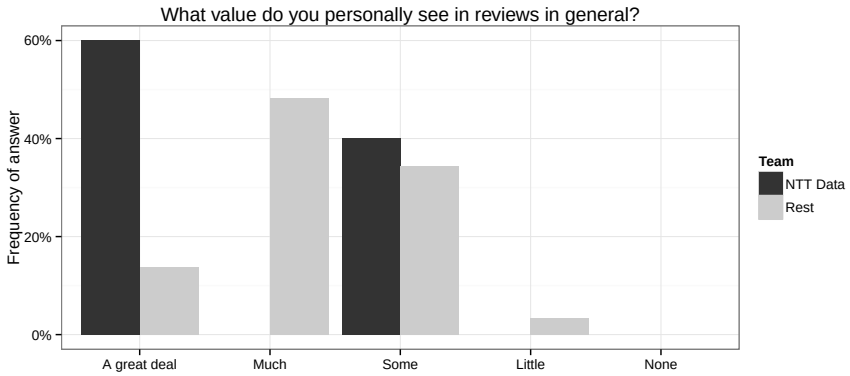

Figure 5.9: Value of reviews

Whereas the ad hoc group's results are distributed around the value of "much", the answers of the IBMR group show an interesting bimodal distribution. Further investigation is needed, but it seems that a percentage of participants only sees moderate usefulness in reviews and this does not change with introduction of an improved process, but that the appraisal of the rest of the participants, who would otherwise see at least "much" value in such reviews, can be raised to value them "a great deal".

5.4.2 *Threats to validity*

In the following, we list the threats to validity. We will start with internal validity, the question whether the study was done correctly. The next subsection deals with external validity, the question whether the results can be generalized. Finally, we discuss construct validity, the question whether the method used is a suitable measure for what is to be tested.

Internal validity

Since filling out the survey was optional, there is a chance for self-selection bias. Developers more interested in the processes used during the course could have been more likely to fill out the entire survey, compared to others less interested. However, the relative response ratio was relatively close for both the experimental team and the control group.

Also, the projects for the experimental team and the teams for the control group were not exactly comparable, since they were real projects posed by external clients. But since all projects were part of the same framework of projects, they were all scoped to be similar in size and complexity (they had to be in order to be able to be comparably graded as part of the university course). They had the same time frame, developers were distributed across teams to provide a balanced distribution of skills among all teams, and they were all working with the same tools and frameworks. Between teams with a client-server infrastructure, there were some small differences due to different choices of server platforms, but on the client side only Objective-C and the Cocoa frameworks were used.

External validity

Since only the experimental team got special instructions on how to do their reviews, there is a risk of the Hawthorne effect (Landsberger, 1958). The Hawthorne effect is a psychological effect, in which participants change their behavior in result to being aware that they are being observed. But while the rest of the teams were not taught the IBMR framework in detail, they also received a detailed lesson in how to review models and documentation, and their instructors encouraged them to review their documentation properly, hinting that documentation quality would be reflected in the final grades.

Construct validity

The survey used personal judgment to determine quality of documents and percentage of feedback incorporated. This does not necessarily have to reflect the real quality of documents, however it was corroborated by the observations of the instructors.

5.5 SUMMARY

The results of the case study showed that IBMR resulted in an improved usage of rationale. The results in comparison to a similar type of student project (VSO) (Wolf, 2007) as depicted in Figure 5.10 show an improved usage of issues in the case study projects.

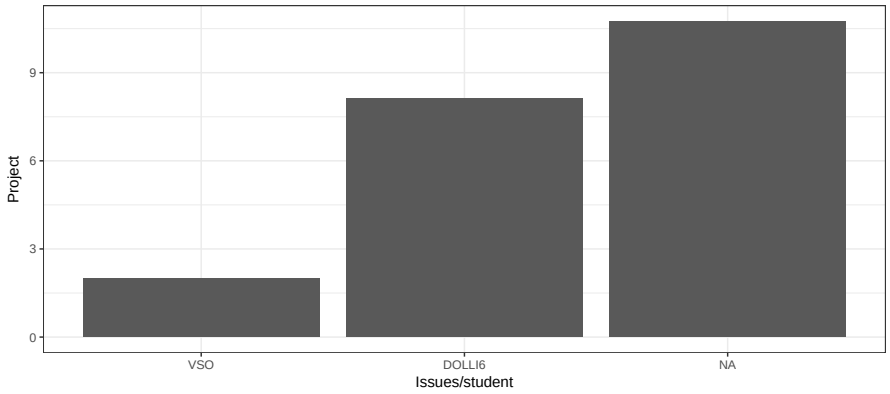

Figure 5.10: Comparison of issues per developer between projects using rationale management

Since the VSO project also explicitly introduced rationale management to participants and also was a similar type of project (a one semester student development project with a real client), the increase in usage of rationale management looks promising.

The results of the survey which followed the second case study indicate that developers did not find the review process any more complicated than ad-hoc reviews, whereas they found the improvement in document quality to be greater for IBMR. They also were more confident of having addressed all feedback from reviews.

6 CONCLUSION

This chapter states the main contributions of this dissertation. We close by presenting ideas for future work.

6.1 SUMMARY

In this dissertation, we have presented a framework called Issue-Based Model Review (IBMR). The main contributions of IBMR are

A MODEL FOR INTEGRATING REVIEWS INTO THE UNIFIED MODEL
: The IBMR model allows for integration with the unified software engineering model, enabling automated checking of simple defect conditions, and simplifying the follow-up on defects.

IMPROVED INTEGRATION OF REVIEWS AND RATIONALE
: Integration of model review documentation and rationale management improves concurrent capture of rationale during the project (instead of documenting rationale after the project is finished), and makes defect status and model maturity traceable and manageable. With IBMR, reviews provide an ongoing source of rationale identification, which can be used to support other rationale capture efforts.

A TAILORABLE PROCESS FOR MODEL-BASED MODEL REVIEWS
: The process for conducting model reviews treats reviews not as special events, but as a workflow supporting continuous checking of models, shortening the time between reviews, and making them more dependent on actual content changes and less dependent on time.

Another contribution of this dissertation are two exploratory case studies in student projects with industry collaboration. The first case study was conducted as part of a large 60 person project. The second case study involved an 8 person project. Both case studies demonstrated the feasibility of the IBMR approach.

In particular, we conducted a survey after the second case study, which indicated that IBMR is not more complex to follow than standard review techniques. Instead it leads to higher confidence among project participants, that all issues from a review have been addressed.

6.2 FUTURE WORK

We successfully conducted two case studies providing anecdotal evidence. Further studies should focus on statistically relevant experiments. Although our case studies had industry partners, they were limited to short term projects. The IBMR approach should also be studied in large, multi-year industrial projects, since these are the type of project which should most strongly benefit from IBMR.

Future work could also address improving the tooling and its integration into the unified model and software lifecycle model. By implementing IBMR in a CASE tool built on a unified software engineering model, review efficiency and effectiveness can be increased by providing better support for conducting and following up on reviews.

Improved tooling would allow for proper implementation of automation support, such as scheduling model elements to be reviewed. This could lead to a reduction in review time, because unchanged model elements would not come up for review.

BIBLIOGRAPHY

Ahtee, Tero and Timo Poranen (Feb. 2009). "Risks in Students' Software Projects." In: *Software Engineering Education and Training, 2009. CSEET '09. 22nd Conference on*. IEEE, pp. 154–157. ISBN: 978-1-4244-3431-2. DOI: 10.1109/CSEET.2009.31.

Alkadhi, Rana Mohammed A (2018). "Rationale in Written Developers' Communications." PhD thesis. Technische Universität München.

Ambler, Scott W (Mar. 2004). *The Object Primer*. English. Agile Model-Driven Development with UML 2.0. Cambridge University Press. ISBN: 0521540186.

Ambler, Scott (2003). *Model Reviews: Best Practice or Process Smell?*

Babar, Muhammad Ali and Ian Gorton (2009). "Software Architecture Review: The State of Practice." In: *Computer* 42.7, pp. 26–32. DOI: 10.1109/MC.2009.233.

Bacchelli, Alberto and Christian Bird (2013). "Expectations, Outcomes, and Challenges of Modern Code Review." In: *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, pp. 712–721. ISBN: 978-1-4673-3076-3.

Bernhart, Mario, Andreas Mauczka, and Thomas Grechenig (2010). "Adopting Code Reviews for Agile Software Development." In: *2010 AGILE Conference*. IEEE, pp. 44–47. ISBN: 978-1-4244-7731-9. DOI: 10.1109/AGILE.2010.18.

Black, Kent M (Jan. 1994). "An Industry View of Engineering Education." English. In: *Journal of Engineering Education* 83.1, pp. 26–28. DOI: 10.1002/j.2168-9830.1994.tb00112.x.

Bos, Nathan, Judy Olson, Darren Gergle, Gary Olson, and Zach Wright (2002). "Effects of four computer-mediated communications channels on trust development." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Minneapolis, Minnesota, USA: ACM, pp. 135–140. ISBN: 1-58113-453-3. DOI: 10.1145/503376.503401.

Brown, William J, Hays W Skip McCormick, and Scott W Thomas (Aug. 2000). *AntiPatterns in Project Management*. English. John Wiley & Sons Incorporated.

Bruegge, B, Allen H. Dutoit, and Timo Wolf (Oct. 2006). "Sysiphus: Enabling informal collaboration in global software development." In: *Global Software Engineering, 2006. ICGSE '06. International Conference on*. IEEE, pp. 139–148. ISBN: 0-7695-2663-2. DOI: 10.1109/ICGSE.2006.261227.

Bruegge, Bernd and Allen H. Dutoit (Aug. 2009). "Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd edi-

tion." In: *Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd edition*.

Bruegge, Bernd, Helmut Naughton, and Michaela Gluchow (May 2011). "SLPC++: Teaching software engineering project courses in industrial application landscapes – A tutorial." In: *CSEET '11: Proceedings of the 2011 24th IEEE-CS Conference on Software Engineering Education and Training*. IEEE Computer Society.

Bruegge, Bernd, Stephan Krusche, and Martin Wagner (2012). "Teaching Tornado: From Communication Models to Releases." In: *Proceedings of the 8th Edition of the Educators' Symposium*. New York, NY, USA: ACM, pp. 5–12. ISBN: 978-1-4503-1812-9. DOI: 10.1145/2425936.2425938.

Ciolkowski, Marcus, Oliver Laitenberger, Dieter Rombach, Forrest Shull, and Dewayne Perry (2002). "Software Inspections, Reviews & Walkthroughs." In: *Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, pp. 641–642. ISBN: 1-58113-472-X. DOI: 10.1145/581339.581422.

Ciolkowski, Marcus, Oliver Laitenberger, and Stefan Biffl (2003). "Software reviews, the state of the practice." English. In: *IEEE Software* 20.6, pp. 46–51. DOI: 10.1109/MS.2003.1241366.

Clinical and Laboratory Standards Institute (2009). "Laboratory Instrument Implementation, Verification, and Maintenance, Approved Guideline." In: *CLSI GP31-A*.

Croll, Grenville J. (Jan. 2003). "A typical model audit approach." In: *Integrity and internal control in information systems V*.

DOLLI 6 Project Page (2012).

David, Jörn, Helmut Naughton, Jonas Helming, and Maximilian Koegel (2009a). "Integrating System Modeling with Project Management - A Case Study." In: *33rd Annual IEEE International Computer Software and Applications Conference, 2009. COMPSAC '09.* IEEE Computer Society, pp. 571–578.

David, Jörn, Maximilian Koegel, Helmut Naughton, and Jonas Helming (July 2009b). "Traceability ReARMed." In: *COMPSAC '09: Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference*. IEEE Computer Society.

Doolan, E P (Feb. 1992). "Experience with Fagan's inspection method." English. In: *Software: Practice and Experience* 22.2, pp. 173–182. DOI: 10.1002/spe.4380220205.

Doyle, M and D Straus (1982). *How to Make Meetings Work: The New Interaction Method*. Jove Books. ISBN: 9780515090482.

Dutoit, A H, R McCall, I Mistrik, and B Paech (2007). *Rationale Management in Software Engineering*. Springer Berlin Heidelberg. ISBN: 9783540309987.

Dutoit, Allen H. and Barbara Paech (Apr. 2012). "Rationale Management In Software Engineering." In: *Handbook of Software Engineering and Knowledge Engineering*. World Scientific Publishing

Company, pp. 787–815. ISBN: 978-981-02-4514-6. DOI: 10.1142/9789812389718_0033.

Fagan, Michael E. (1986). "Advances in Software Inspections." In: *IEEE Transactions on Software Engineering* SE-12, pp. 744–751.

— (1976). "Design and Code Inspections to Reduce Errors in Program Development." In: *IBM Systems Journal* 15.3, pp. 182–211.

Fey, Ines and Ingo Stürmer (Apr. 2007). *Quality Assurance Methods for Model-Based Development: A Survey and Assessment*. Tech. rep. Warrendale, PA: SAE International. DOI: 10.4271/2007-01-0506.

Finnigan, David, Elizabeth A. Kemp, and Daniela Mehandjiska (2000). "Towards an ideal CASE tool." In: *Proceedings. International Conference on Software Methods and Tools, 2000. SMT 2000.* IEEE Comput. Soc, pp. 189–197. ISBN: 0-7695-0903-7. DOI: 10.1109/SWMT.2000.890434.

Fisher, Roger and William L Ury (1981). *Getting to YES: Negotiating Agreement Without Giving In*. Penguin Group. ISBN: 978-0-14-015735-2.

Freedman, Daniel P. and Gerald M. Weinberg (1990). *Handbook of walkthroughs, inspections, and technical reviews : evaluating programs, projects, and products*. Little, Brown computer systems series. New York, NY: Dorset House Pub. ISBN: 9780932633194.

Gilb, Tom and Dorothy Graham (1993). *Software Inspection*. Ed. by Susannah Finzi. Addison-Wesley Longman, Amsterdam. ISBN: 0201631814.

Gorschek, Tony and Mikael Svahnberg (2005). "Requirements Experience in Practice: Studies of Six Companies." English. In: *Engineering and Managing Software Requirements*. Berlin/Heidelberg: Springer Berlin Heidelberg, pp. 405–426. ISBN: 978-3-540-25043-2. DOI: 10.1007/3-540-28244-0_18.

Gotel, Orlena C. Z. and Anthony C. W. Finkelstein (1994). "An analysis of the requirements traceability problem." In: *IEEE International Conference on Requirements Engineering*. IEEE Comput. Soc. Press, pp. 94–101. ISBN: 0-8186-5480-5. DOI: 10.1109/ICRE.1994.292398.

Hedberg, Henrik and Jouni Lappalainen (2005). "A preliminary evaluation of software inspection tools, with the DESMET method." In: *Sixth International Conference on Quality Software, 2006. QSIC 2006.* IEEE, pp. 45–52. ISBN: 0-7695-2472-9. DOI: 10.1109/QSIC.2005.7.

Helming, Jonas (2011). "Merging Project Management with System Modeling." PhD thesis. Technische Universität München.

Helming, Jonas, Maximilian Koegel, and Helmut Naughton (May 2009a). "Towards traceability from project management to system models." In: *Traceability in Emerging Forms of Software Engineering, 2009. TEFSE '09. ICSE Workshop on*. IEEE, pp. 11–15. ISBN: 978-1-4244-3741-2. DOI: 10.1109/TEFSE.2009.5069576.

Helming, Jonas, Maximilian Koegel, Helmut Naughton, Jörn David, and Aleksandar Shterev (2009b). "Traceability-Based Change Awareness." English. In: *Model Driven Engineering Languages and Systems*. Ed. by Andy Schürr and Bran Selic. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 372–376–376. ISBN: 978-3-642-04424-3. DOI: 10.1007/978-3-642-04425-0_28.

Hollocker, Charles P (1990). "Software Reviews and Audits Handbook." In:

IEEE (2008). "IEEE Standard for Software Reviews and Audits." In: *IEEE Std 1028-2008*, pp. 1–52. ISSN: 978-0-7381-5769-6. DOI: 10.1109/IEEESTD.2008.4601584.

— (1983). "IEEE standard glossary of software engineering terminology." In: *IEEE Std 729-1983*.

ISO (May 2012). *Information technology - Object Management Group Object Constraint Language (OCL)*. ISO.

ITIL (2016). *The Official IT Infrastructure Library Website*.

Imai, Masaaki (Nov. 1986). *Kaizen: The Key To Japan's Competitive Success*. McGraw-Hill/Irwin. ISBN: 9780075543329.

Jacobson, Ivar, Patrik Jonsson, Magnus Christerson, and Gunnar Overgaard (July 1992). *Object-Oriented Software Engineering*. A Use Case Driven Approach. Addison-Wesley. ISBN: 978-0201544350.

Jetley, R, S Purushothaman Iyer, and P L Jones (Apr. 2006). "A formal methods approach to medical device review." English. In: *Computer* 39.4, pp. 61–67. DOI: 10.1109/MC.2006.113.

Johnson, Philip M (1994). "An instrumented approach to improving software quality through formal technical review." In: *16th International Conference on Software Engineering*. IEEE Comput. Soc. Press, pp. 113–122. ISBN: 0-8186-5855-X. DOI: 10.1109/ICSE.1994.296771.

Kamm, Daniel (May 2005). *An Introduction to Risk/Hazard Analysis for Medical Devices*. Tech. rep.

Kemerer, Chris F. and Mark C. Paulk (2009). "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data." In: *IEEE Transactions on Software Engineering* 35.4, pp. 534–550. DOI: 10.1109/TSE.2009.27.

Koegel, Maximilian (2011). "Operation-based Model Evolution." PhD thesis. Technische Universität München.

Kruchten, Philippe (2004). *The Rational Unified Process*. English. 3rd. An Introduction. Addison-Wesley Professional. ISBN: 9780321197702.

Kunz, Werner and Horst WJ Rittel (1970). *Issues as elements of information systems*. Vol. 131. Institute of Urban and Regional Development, University of California Berkeley, California.

Laitenberger, Oliver and Jean-Marc DeBaud (2000). "An Encompassing Life Cycle Centric Survey of Software Inspection." English. In: *J. Syst. Softw.* 50.1, pp. 5–31. DOI: 10.1016/S0164-1212(99)00073-4.

Landsberger, Henry A (1958). *Hawthorne revisited : management and the worker : its critics, and developments in human relations in industry*. Ithaca, N.Y.: Cornell University.

MacLean, Allan, Richard M Young, Victoria M E Bellotti, and Thomas P Moran (1991). "Questions, Options, and Criteria: Elements of Design Space Analysis." In: *Hum.-Comput. Interact.* 6.3, pp. 201–250. DOI: 10.1207/s15327051hci0603{\&}4_2.

Naughton, Helmut (2008). "Meeting management in the Unified Model." MA thesis. Technische Universität München.

Nguyen, David T and John Canny (2009). "More than face-to-face: empathy effects of video framing." In: *the SIGCHI Conference*, pp. 423–432. DOI: 10.1145/1518701.1518770.

Ott, Daniel and Alexander Raschke (2012). "Review improvement by requirements classification at Mercedes-Benz: Limits of empirical studies in educational environments." In: *2012 IEEE Second International Workshop on Empirical Requirements Engineering (EmpiRE)*. IEEE, pp. 1–8. ISBN: 978-1-4673-4364-0. DOI: 10.1109/EmpiRE.2012.6347677.

Parnas, David L and David M Weiss (1985). "Active design reviews: principles and practices." In: *Proceedings of the 8th International Conference on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 132–136. ISBN: 0-8186-0620-7.

Popper, Karl R (1934). *Logik der Forschung*. Wien: Springer.

Porter, Adam A and Lawrence G Votta Jr (1997). "What makes inspections work?" In: *Software, IEEE* 14.6, pp. 99–102. DOI: 10.1109/52.636690.

Rösler, Peter, Maud Schlich, and Ralf Kneuper (2013). *Reviews in der System- und Softwareentwicklung: Grundlagen, Praxis, kontinuierliche Verbesserung*. Dpunkt.Verlag GmbH. ISBN: 9783864900945.

Salger, Frank (2013). "Requirements Reviews Revisited: Residual Challenges and Open Research Questions." In: *Proceedings RE 2013*. IEEE, pp. 250–255.

Selic, Bran (Sept. 2003). "The pragmatics of model-driven development." English. In: *IEEE Software* 20.5, pp. 19–25. DOI: 10.1109/MS.2003.1231146.

Shaw, M E (1971). *Group dynamics, the psychology of small group behavior*. McGraw-Hill series in psychology. McGraw-Hill.

Shirey, Glen C (1992). "How inspections fail." In: *Proceedings of the 9th International Conference on Testing Computer Software*, pp. 151–159.

Shum, Simon Buckingham (1996). "Analyzing the usability of a design rationale notation." In: *Design rationale: Concepts, techniques, and use*, pp. 185–215.

Slater, Philip E (June 1958). "Contrasting Correlates of Group Size." In: *Sociometry* 21.2, p. 129. DOI: 10.2307/2785897.

Votta Jr, Lawrence G (1993). "Does Every Inspection Need a Meeting?" In: *Proceedings of the 1St ACM SIGSOFT Symposium on*

Foundations of Software Engineering. New York, NY, USA: ACM, pp. 107–114. ISBN: 0-89791-625-5. DOI: 10.1145/256428.167070.

Wheeler, David A, Bill Brykczynski, and Reginald N Meeson Jr (1996). *Software Inspection: An Industry Best Practice for Defect Detection and Removal*. IEEE Computer Society Press.

Wolf, Timo (2007). "Rationale-based Unified Software Engineering Model." PhD thesis. Technische Universität München.

Yin, Robert K (2014). *Case study research : design and methods*. 5. ed. London: SAGE. ISBN: 978-1-4522-4256-9.

iOS Praktikum 2013 Project Page (2013).

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of July 1, 2018 (`draft`).