Susanne Jäger

# Entwicklung eines elektronischen Tutors zur Analyse von Projektverläufen

**DIPLOMARBEIT**

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

Studium der Angewandten Informatik

Universität Klagenfurt

Fakultät für Wirtschaftswissenschaften und Informatik

Begutachter: O.Univ.Prof. Mag. DI Dr. Roland Mittermeir
Institut: Informatik-Systeme

April/2003

# Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Schrift verfasst und die mit ihr unmittelbar verbundenen Arbeiten selbst durchgeführt habe. Die in der Schrift verwendete Literatur sowie das Ausmaß der mir im gesamten Arbeitsvorgang gewährten Unterstützung sind ausnahmslos angegeben. Die Schrift ist noch keiner anderen Prüfungsbehörde vorgelegt worden.

Klagenfurt, 23. April 2003

# Danksagung

Viele haben dazu beigetragen, dass diese Arbeit zustande gekommen ist. Besonders bedanken möchte ich mich bei Professor Roland Mittermeir, der mir in zahlreichen Diskussionen wichtige Impulse für meine Diplomarbeit gegeben hat. Weiters möchte ich mich bei meinem Betreuer Andreas Bollin und dem gesamten Projektteam für die angenehme Zusammenarbeit bedanken. Durch die Projektarbeit konnte ich viel über den Softwareentwicklungsprozess lernen und viele Erfahrungen für die Zukunft sammeln. Außerdem möchte ich mich bei Tilmann Hampp bedanken, der mein Ansprechpartner in Modellfragen war und mir mit Rat und Tat zur Seite gestanden ist. Ein spezieller Dank gehört meinem Studienkollegen und Freund Markus Nusser, der mich während des gesamten Studiums begleitet hat und mich motiviert hat, falls dies nötig war. Der größte Dank gebührt jedoch meinen Eltern, die Vertrauen in mich gesetzt und mir dieses Studium überhaupt erst ermöglicht haben. Ihnen sei diese Arbeit gewidmet.

# ZUSAMMENFASSUNG

Die Studenten besitzen zurzeit nur theoretisches Wissen über Projektmanagement. Durch den SESAM-Simulator können sich die Studierenden zum ersten Mal als Projektleiter versuchen und praktische Erfahrungen für zukünftige reale Projekte sammeln. Der Lernerfolg kann jedoch nur dann gewährleistet werden, wenn die Spielverläufe der Spieler vom Tutor genau analysiert werden. Der hohe Betreuungsaufwand des Tutors, macht es aber unmöglich den SESAM-Simulator auch für große Lehrveranstaltungen und für das Selbststudium einzusetzen.
Durch das AMEISE-System werden diese Probleme beseitigt. Die generische Erklärungskomponente ermöglicht eine automatische Auswertung der Spielverläufe und kann dadurch den Tutor stark entlasten. Da zum Analysieren der Bewertungskriterien immer dieselben Fragestellungen beantwortet werden mussten, wurde das Design Pattern „Interpreter“ als Lösungsansatz verwendet. Es wurde eine eigene Grammatik entwickelt, mit deren Hilfe Regeln formuliert werden konnten. Diese Regeln werden im AMEISE-System durch spezielle Hilfsmittelkomponenten repräsentiert. Jede Hilfsmittelkomponente gibt an, welche Daten aus der Datenbank geladen werden müssen, und wie diese zu verarbeiten sind. Die generische Erklärungskomponente fungiert als Regelinterpreter, der diese Repräsentation nutzt, um die Regeln mit den aktuellen Spieldaten zu verknüpfen und auszuwerten. Da die spezielle Hilfsmittelkomponente auch aus mehreren Instanzen bestehen kann, können damit auch komplexere Bewertungskriterien analysiert werden. Dieser generische Ansatz ermöglicht, dass verschiedene Bewertungskriterien unterschiedlicher Modelle damit ausgewertet werden können.
Die Auswertungsergebnisse der generischen Erklärungskomponente werden dem Spieler in Form von Diagrammen und Erklärungstexten präsentiert. In den Erklärungstexten wird der Spieler auf Fehlentscheidungen und deren Auswirkungen auf das Projekt hingewiesen. Der Spieler bekommt aber auch Tipps, die ihm helfen sollen, diese Fehler bei der Durchführung des nächsten Projekts zu vermeiden.
Durch die generische Erklärungskomponente kann das AMEISE-System in Zukunft sowohl für große Lehrveranstaltungen als auch für das Selbststudium eingesetzt werden.

# INHALTSVERZEICHNIS

# ABBILDUNGSVERZEICHNIS

# TABELLENVERZEICHNIS

*Kapitel 1*

# EINLEITUNG UND MOTIVATION

Das erfolgreiche Durchführen eines Software-Entwicklungsprojekts ist eine schwierige Aufgabe, da nicht nur technische, sondern auch organisatorische Probleme [Mandl-Striegnitza] [Mandl-Striegnitzb] gemeistert werden müssen. Junge Projektleiter scheitern häufig bereits an ihrem ersten Projekt, da es ihnen an Erfahrung fehlt. Es wäre darum sinnvoll schon im Rahmen der Ausbildung den Studenten die Möglichkeit zu geben, sich als Projektleiter zu versuchen, ohne dass sie die Angst des Scheiterns verspüren und sich Gedanken über die damit verbundenen Konsequenzen machen müssen.
In Projektmanagement-Vorlesungen werden zwar mögliche Probleme, die beim Managen von Projekten auftreten können, diskutiert, praktische Übungen auf diesem Gebiet konnten aber bis jetzt aus zeitlichen Gründen nicht durchgeführt werden. Da man jedoch am besten aus praktischen Erfahrungen lernen kann [Ludewiga], wurde von der Universität Stuttgart ein Simulator genannt SESAM („Software Engineering Simulation by Animated Models“) entwickelt, mit dessen Hilfe die Ausbildungslücke im praktischen Projektmanagement gefüllt werden kann.

Das AMEISE-System (AMEISE steht für „A MEdia Initiative for Software Engineering“), das diesen so genannten SESAM-Simulator enthält, ermöglicht es den Studenten sich als Projektleiter eines simulierten Software-Entwicklungsprojekts zu versuchen. Mit Hilfe des AMEISE-Systems können zukünftige Projektleiter Projekte in wenigen Stunden durchführen, die normalerweise mehrere Monate dauern würden.
Die Studierenden können somit spielerisch am SESAM-Simulator Projektmanagement-Fähigkeiten erlernen. Sie werden dabei mit typischen Problemen in Projekten konfrontiert und lernen diese zu meistern. Um welche Projekte es sich handelt wird durch das Modell entschieden, das in den Simulator geladen werden muss. Während der Simulation des Projekts werden alle Anweisungen des Studierenden mitprotokolliert. Diese Daten spiegeln am Ende der Simulation den gesamten Projektablauf (im Weiteren auch als Spielver-

lauf bezeichnet) wider. Dieser Spielverlauf kann im Anschluss vom Lehrveranstaltungsleiter[1] (im Folgenden auch als Tutor bezeichnet) genau analysiert werden. Der Tutor kann dem Spieler dadurch Fehlentscheidungen im Projektverlauf aufzeigen und ihn auf die Auswirkungen seiner Entscheidungen auf das Projekt hinweisen.

In meiner Diplomarbeit möchte ich eine Kernkomponente des AMEISE-Systems, die generische Erklärungskomponente, welche ich im Rahmen meiner AMEISE-Projektmitarbeit entwickelt habe, vorstellen. Die generische Erklärungskomponente ist deshalb so wichtig, da ohne ihre Hilfe die Studenten keinen Lernerfolg durch ein Spiel am SESAM-Simulator erzielen würden. Erst durch das Aufzeigen der Fehler kann der Spieler erkennen, welche Auswirkungen seine Entscheidungen auf das Projekt haben und lernen welche Entscheidungen er in welchen Situationen treffen muss, damit das Projekt erfolgreich abgeschlossen werden kann. Die Erklärungskomponente kann eine automatische Auswertung von Spielverläufen durchführen und somit den Lehrveranstaltungsleiter stark entlasten, der bis dahin die Auswertung der Spielverläufe von Hand vornehmen musste.
Es sei hier bereits angemerkt, dass mir die Arbeit großen Spaß gemacht hat, und dass es sehr motivierend war etwas für die Lehre zu entwickeln und damit zur Verbesserung der Ausbildung zukünftiger Projektleiter beizutragen.

An dieser Stelle möchte ich noch auf die Gliederung der Arbeit eingehen. In Kapitel 2 wird das Projekt AMEISE präsentiert und dabei näher auf die Erklärungskomponente eingegangen, die eine Kernkomponente des AMEISE-Projekts darstellt. Der Aufbau von SESAM sowie der Ablauf von SESAM-Spielen wird in Kapitel 3 skizziert. Die existierenden Modelle und die bereits entwickelten Auswertungswerkzeuge werden ebenfalls in diesem Abschnitt vorgestellt.
Mein Vorgehen bei der Analyse von Projektverläufen sowie interessante Aspekte von Spielen werden in Kapitel 4 genauer betrachtet. In Kapitel 5 werden die von mir als wichtig erachteten Bewertungskriterien zusammengefasst, welche bei der Analyse der Projektverläufe in Kapitel 4 entdeckt wurden.
Die Architekturüberlegungen, die bei der Entwicklung der generischen Erklärungskomponente gemacht wurden, werden in Kapitel 6 näher beschrieben. Auf die Architektur, die hinter der generischen Erklärungskomponente steckt, wird in Kapitel 7 genauer eingegangen.
In Kapitel 8 wird ein kurzer Überblick über den bis zu diesem Zeitpunkt

[1] Der einfacheren Formulierung halber verwende ich in dieser Arbeit fast ausschließlich die männlichen Bezeichnungen, sie gelten dem Sinn gemäß jedoch beidgeschlechtlich.

realisierten Umfang der generischen Erklärungskomponente gegeben, um im Kapitel 9 noch genauer auf Erweiterungen und Grenzen der Komponente eingehen zu können. Eine Zusammenfassung der Arbeit wird in Kapitel 10 gebracht.

*Kapitel 2*

# PROJEKT AMEISE

In diesem Kapitel wird das Projekt AMEISE vorgestellt. Nach einer genaueren Beschreibung des SESAM-Simulators, werden die damit erreichbaren Lernziele diskutiert sowie Erweiterungen vorgestellt, die notwendig sind, um diese Lernziele zu erreichen. Der SESAM-Simulator bildet den Kern des AMEISE-Systems und erlaubt es den Studierenden klassische Softwareentwicklungsprojekte zu leiten. Dazu zählen das Einstellen von qualifiziertem Personal, das Zuordnen von Aufgaben und das Kontrollieren von Ergebnissen, um möglichst nahe an die Zielvorgaben, die Budget, Dauer und Qualität der Dokumente betreffen, heranzugelangen. Die Analyse der Spielverläufe erfolgt im Anschluss an das Spiel durch den Tutor und ist sehr zeitaufwändig. Dadurch wird der Einsatz des Systems sehr stark eingeschränkt.
Durch das AMEISE-Projekt soll die Einsetzbarkeit des SESAM-Simulators derart erweitert werden, dass er sowohl in größeren Lehrveranstaltungen als auch für ein Selbststudium eingesetzt werden kann. Welche Erweiterungen dafür notwendig sind, wird am Ende dieses Kapitels erläutert.

## *2.1 Der SESAM-Simulator*

Der SESAM-Simulator ist ein neuer didaktischer Ansatz und wurde von der Arbeitsgruppe um Prof. Jochen Ludewig [LUDEWIGb] an der Universität Stuttgart entwickelt. Mit seiner Hilfe soll die Ausbildung der Studierenden im Softwareprojektmanagement durch interaktive Simulation von Software-Projekten ermöglicht werden, da Studierende nur theoretisches Wissen in diesem Bereich besitzen. Der Prototyp „SESAM“ der Universität Stuttgart erlaubt experimentelles Lernen in einer Simulationsumgebung. Damit kann der Student versuchen, das in Lehrveranstaltungen Erlernte umzusetzen und, so die Idee, auf diese Weise auch „praktische“ Erfahrungen im Bereich Projektmanagement sammeln [DRAPPAc].

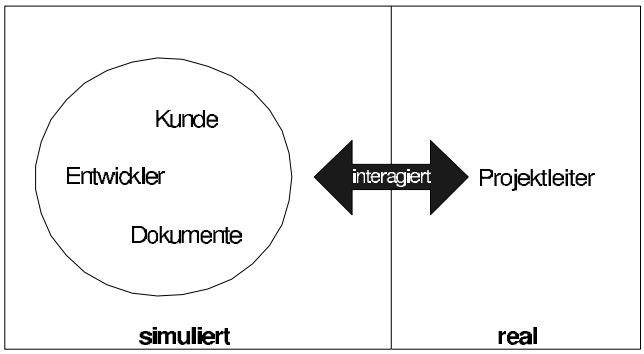



*Abb. 2.1:* Die Grundidee von SESAM

Die Abbildung 2.1 zeigt die realen und simulierten Objekte des SESAM-Simulators. Relevante Objekte eines Projekts wie Kunde, Entwickler und Dokumente werden simuliert, während der Student in die Rolle des Projektleiters schlüpft und als reale Person mit den simulierten Objekten interagiert.

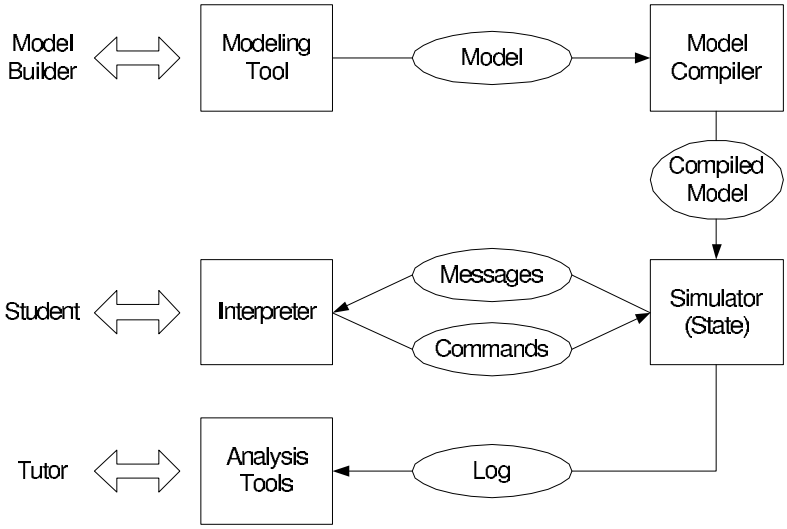



*Abb. 2.2:* Überblick über die SESAM-Komponenten [DRAPPAd]

Abbildung 2.2 gibt einen Überblick über die bereits bestehenden Komponenten und deren Zusammenspiel. Der Modellbauer verwendet ein Modellierungstool, um ein Modell erstellen zu können, welches im Anschluss mit Hilfe eines Compilers übersetzt wird. Sobald das übersetzte Modell in den Simulator eingespeist wurde, kann das Spiel beginnen, das bedeutet, der Stu-

dent kann nun Kommandos absetzen. Die in natürlicher Sprache abgesetzten Kommandos des Spielers müssen jedoch zuvor von einem Interpreter übersetzt werden, bevor sie vom Simulator verarbeitet werden können. Der Simulator verarbeitet die Kommandos, ändert seinen internen Zustand und schickt Nachrichten an den Studenten zurück. Während des Spiels werden alle Kommandos, die vom Spieler abgesetzt werden in einer Log-Datei gespeichert. Der Tutor hat am Ende des Spiels die Möglichkeit diese Log-Datei unter Zuhilfenahme von Auswertungswerkzeugen (SesamScore und SesamAlyzer) zu analysieren, um das Spiel des Studenten bewerten zu können. Genauere Angaben über die bestehenden Auswertungswerkzeuge findet der Leser in Abschnitt 3.4.

## 2.2 Lernziele

Durch den Einsatz dieses Simulators für die Ausbildung von Studierenden sollen zusätzlich zum theoretischen Wissen auch die praktischen Aspekte des Projektmanagements erlernt werden. Die Studenten sollen gegenüber dem Projekt dieselbe Rolle einnehmen wie ein realer Projektleiter. Sie sollen das selbständige Planen und Durchführen von Software-Projekten trainieren und sich dabei wichtige Projektmanagementfunktionen aneignen. Auch die Entscheidungsfähigkeit unter unvollständiger Information soll damit trainiert werden, da Projektleiter ihre Entscheidungen in der Regel bei nur geringer Information über das Projekt treffen müssen. Studierende sollen aber auch mit typischen Schwierigkeiten, die bei der Planung und Kontrolle von Projekten auftreten können, konfrontiert werden und lernen, wie man mit diesen umgehen muss. Das Begreifen des Zusammenhangs zwischen Projektsteuerung und Projektresultaten ist ebenso wichtig, wie das Meistern von schwierigen Situationen, die während eines Projekts auftreten können. Auch das Herausführen von Projekten aus schwierigen Situationen kann mit Hilfe des Simulators trainiert werden und zählt mit zu den wichtigsten Lernzielen, die mit einer Ausbildung der Studierenden am SESAM-Simulator erreicht werden können.

## 2.3 Die Vision

Der SESAM-Simulator eignet sich zurzeit nur für den Einsatz in kleinen Lehrveranstaltungen. Der Grund dafür ist der extrem hohe Betreuungsaufwand des Tutors, da dieser den Projektverlauf jedes Studenten händisch analysieren

muss. Um das Erreichen der Lernziele überprüfen zu können und den Spielern gute und schlechte Entscheidungen und deren Auswirkungen in ihren Projekten aufzeigen zu können, benötigt der Tutor viel Zeit für die Analyse.

Durch Zusammenarbeit (Software Engineering Lab am Institut für Informatik der Universität Stuttgart, Institut für Systemwissenschaften der Universität Linz, Fachhochschule Technikum Kärnten und Fakultät für Wirtschaftswissenschaften und Informatik der Universität Klagenfurt) soll der Einsatz des SESAM-Simulators auch für große Lehrveranstaltungen bzw. für ein Selbststudium möglich gemacht werden. Dazu muss der SESAM-Prototyp jedoch um zusätzliche Komponenten erweitert werden. Hierzu zählen Erklärungskomponente, Kommunikationskomponente und Rollback-Komponente. Die Entwicklung einer Erklärungskomponente ist besonders wichtig, da der Betreuungsaufwand des Tutors unbedingt minimiert werden muss. Hinter der Erklärungskomponente verstecken sich mehrere Aspekte, die für die Bewertung von Spielverläufen denkbar sind.

- **Zwischen- bzw. Endauswertung**
  Die Zwischen- bzw. Endauswertung eines Projektverlaufs soll die Arbeit des Tutors übernehmen und dem Spieler gute und schlechte Entscheidungen sowie deren Auswirkungen auf das Projekt aufzeigen.

- **Unterstützungstools**
  Die Unterstützungstools Ratgeber und „väterlicher Freund“, die in der Diplomarbeit von Markus Nusser [Nusser] beschreiben werden, sollen dem Spieler während des SESAM-Spiels unterstützen. Der Spieler soll durch diese Komponenten die Möglichkeit haben Ratschläge während des Spiels zu bekommen bzw. einzuholen, um in schwierigen Situationen beim Treffen der richtigen Entscheidungen unterstützt zu werden.

- **Vergleich von Spielen**
  Der Vergleich eines Spielverlaufs mit dem Projektverlauf eines anderen Spielers oder sogar einer Gruppe von Spielern kann für die Studierenden als Anreiz für weitere Spiele dienen [Nusser].

Meine Arbeit befasst sich mit dem Teil der Erklärungskomponente, die die Zwischen- bzw. Endauswertung von Projektverläufen vornehmen soll. Um die automatische Analyse von Projektverläufen nicht nur auf ein Modell beschränken zu müssen und somit für jedes neue Modelle ein eigenes Auswertungstool erzeugen zu müssen, habe ich mich für die Entwicklung einer generischen Erklärungskomponente entschieden, die für alle existierenden bzw.

zukünftigen Modelle verwendet werden kann. Eine genaue Beschreibung der Erklärungskomponente wird in Kapitel 6 gegeben. Auf die Grenzen der gewählten Architektur wird in Kapitel 9 eingegangen.

Bevor ich jedoch genauer auf die von mir entwickelte generische Erklärungskomponente eingehe, möchte ich die Grundlagen des SESAM-Spiels beschreiben, um den Spielablauf zu verdeutlichen.

*Kapitel 3*

# SESAM

## *3.1 Aufbau des Simulators*

Der Simulator basiert auf einem Projektmodell, welches von einem Modellbauer erstellt wird. Der Modellbauer hat die Aufgabe, alle in einem realen Projekt existierenden Objekte und Beziehungen auf ein Projektmodell von SESAM abzubilden. Diese Abbildung wird mit Hilfe der Modellbeschreibungssprache SESAM-2 vorgenommen.
Damit bestimmte Lernziele durch den Einsatz dieses Modells erreicht werden können, muss der Modellbauer wichtige Modellanforderungen [DRAPPAd] beachten:

- das Modell muss flexibel sein, so dass Studierende auf ihre Weise versuchen können Probleme zu lösen
- das Modell muss feingranular sein, um die Aktivitäten, die ein Projektleiter in realen Projekten vornimmt abzudecken
- das Modell muss alle Phasen des Softwareentwicklungsprozesses abdecken, damit die Studierende lernen Probleme zu meistern, die erst in späteren Projektphasen auftreten.
- das Modell muss auf empirischen Daten beruhen, damit die Studierende ihre Erfahrungen auch in reale Projekte einfließen lassen können.

Der Simulator besitzt einen internen Zustand, der der aktuellen Situation, welche durch Entitäten und Relationen mit konkreten Attributwerten beschrieben wird, entspricht. Die aktuelle Situation wird in einer globalen Datenstruktur gespeichert, die wir im Weiteren als internen Zustand verstehen. Der Simulator versucht je Zeitschritt, Regeln des Regelmodells auf die aktuelle Situation anzuwenden. Wenn eine Regel ausgeführt wird, dann versucht sie, den Teil der globalen Datenstruktur, die die aktuelle Situation repräsentiert, zu binden. Dadurch kommt es zu einer Änderung des internen Zustands. Der Simulator arbeitet deshalb sowohl ereignis- als auch zeitgesteuert.

- ereignisgesteuert:
  Jedes Ereignis besitzt einen Eintrittszeitpunkt und bewirkt bei seinem Eintritt einen Zustandsübergang.

- zeitgesteuert:
  Bei jedem Simulationsschritt wird die Simulationszeit um eine festgelegte Einheit erhöht. Im klassischen QS-Modell zum Beispiel wird als Zeiteinheit ein Tag verwendet. Nach dem Fortschreiten der Simulation um einen Tag (im Spiel durch den Befehl „Proceed“ ausgelöst), werden alle aufgetretenen Zustandsänderungen seit dem letzten „Proceed“ durchgeführt.

Im Weiteren wird nun näher auf das Projektmodell eingegangen.

## *3.2 Projektmodell*

In einem realen Projekt gibt es zum einen Personen wie Mitarbeiter und Kunden, und zum anderen die Dokumente, die von den Mitarbeitern erzeugt werden (z.B. Spezifikation, Entwurf, Code, usw.). Außerdem weisen sowohl Personen als auch Dokumente bestimmte Merkmale auf. Ein Mitarbeiter hat zum Beispiel einen Namen und bestimmte Fähigkeiten, wohingegen Dokumente durch die Anzahl ihrer Seiten und die enthaltenen Fehler ausgezeichnet werden. Weiters existieren auch Beziehungen zwischen Objekten, zum Beispiel die, dass eine bestimmte Person ein bestimmtes Dokument erstellt. Zusätzlich gibt es eine zentrale Person, den Projektleiter, von dem der Erfolg des Projekts sehr stark abhängt. Dieser Projektleiter wird vom Spieler repräsentiert, während alle anderen Objekte von SESAM simuliert werden.

Durch die Modellbeschreibungssprache SESAM-2 lassen sich Objekte, Beziehungen zwischen Objekten sowie Merkmale der Objekte und Beziehungen abbilden. Außerdem können Aktionen des Projektleiters und Informationen für den Projektleiter sowie eine bestimmte Startsituation abgebildet werden.

Das Projektmodell[1] entspricht somit der Abbildung eines realen Projekts und besteht, wenn wir es gründlich betrachten, aus einem Schemamodell, einem Regelmodell und einem Situationsmodell. Befassen wir uns nun etwas genauer mit den drei Komponenten des Projektmodells.

[1] Genauere Informationen über den Aufbau des Projektmodells kann der Leser dem dazugehörigen Benutzerhandbuch entnehmen.

### 3.2.1 Das Schemamodell

Das Schemamodell beschreibt die Umgebung, in der die Projektmanagementsimulation stattfinden kann. Es definiert alle an einem Projekt beteiligten Typen von Objekten (auch Entitätstypen genannt), alle Beziehungen (auch Relationstypen genannt) zwischen diesen Objekten und die Eigenschaften der Objekte und Beziehungen. Dazu wird eine erweiterte Entity-Relationship Notation verwendet, in der die relevanten Objekte und Beziehungen (Entitäts- und Relationstypen) definiert werden und weitere Beschreibungen dieser Typen durch Attribute möglich sind. Das Schemamodell beschreibt alle möglichen Situationen, die im Verlauf eines Projekts durch Instanziierung des Schemas gewonnen werden können.

Die Entitäts- und die Relationstypen werden im Schemamodell aber nur definiert. Die Instanziierung der Typen erfolgt zum einen zum Startzeitpunkt der Simulation durch das Situationsmodell, und zum anderen während der Simulation durch die Regeln und Benutzerkommandos des Regelmodells.

### 3.2.2 Das Situationsmodell

Das Situationsmodell dient der Beschreibung einer konkreten Projektsituation und besteht aus Instanzen der Entitäts- und Relationstypen. Die Instanzen repräsentieren die simulierten Objekte und Beziehungen. Außerdem besitzen Instanzen konkrete Attributwerte.

Das Situationsmodell repräsentiert den eigentlichen Modellzustand (interner Zustand), auf den das Regelmodell angewendet wird. Mit anderen Worten, das Situationsmodell repräsentiert die aktuelle Situation des Projekts zu einem bestimmten Zeitpunkt. Es müssen allerdings zwei Situationen unterschieden werden. Zum einen die Abbildung der Anfangssituation eines Projekts, und zum anderen die Darstellung der momentanen Situation während der Simulation.

Die Abbildung der Anfangssituation erzeugt konkrete Ausprägungen von im Schemamodell definierten Entitäts- und Relationstypen. Außerdem enthält die Anfangssituation alle Informationen, die dem Spieler zu Beginn der Simulation über das Projekt mitgeteilt werden.

### 3.2.3 Das Regelmodell

Das Regelmodell dient der Beschreibung dynamischer Effekte. Zu den Komponenten des Regelmodells zählen Aktivitäten, Regeln und Benutzerkommandos. Die Ausführung der Komponenten ermöglicht eine Veränderung der aktuellen Situation. Das Regelmodell bildet also den Kern des dynamischen Systems. Die Notation, die dabei verwendet wird, basiert auf der Graphgrammatik [MELCHISEDECH]. Jede Regel besteht aus einem Strukturteil, einem Bedingungsteil und einem Aktionsteil. Im Strukturteil werden die Entitäten und Relationen (im Weiteren formale Komponenten genannt) angegeben, die für die Ausführung der Regel notwendig sind. Im Bedingungsteil muss zwischen Strukturbedingungen und Attributbedingungen unterschieden werden. Strukturbedingungen sind Prädikate über den formalen Komponenten, während Attributbedingungen Prädikate über den Attributen der formalen Komponenten sind. Im Aktionsteil können Komponenten erzeugt und gelöscht werden bzw. einzelne Komponenten können durch Ändern von Attributwerten modifiziert werden.

Wie bereits erwähnt, zählen zu den Komponenten des Regelmodells Aktivitäten, Regeln und Benutzerkommandos. Es muss zwischen allgemeinen Regeln und Regeln mit Feuerungssemantik unterschieden werden. Allgemeine Regeln werden benötigt, um ein Situationsmodell dynamisch an die neue Spielsituation anpassen zu können. Auch das Einfügen, Abfragen und Löschen von Entitäten und Relationen wird von solchen Regeln vorgenommen. Ebenso werden entitäts- und relationsbeschreibende Attributwerte durch diese Regeln geändert. Eine Regel ist immer dann aktiv, wenn mindestens eine Regelinstanz innerhalb des aktuellen Situationsmodells existiert. Regeln mit Feuerungssemantik dagegen sorgen für die zu einem bestimmten Zeitpunkt stattfindenden diskreten Änderungen des Situationsmodells.

Außerdem müssen hier noch Aktivitäten genannt werden, denen das Aktivierungs- und Deaktivierungskonzept zugrunde liegt. Sie ermöglichen sowohl die über einen längeren Zeitraum stattfindenden kontinuierlichen Änderungen des Situationsmodells als auch die hierarchische Strukturierung der Komponenten des Regelmodells.

Die letzte Komponente des Regelmodells bilden die Benutzerkommandos, die parametrisierten Feuerungsregeln entsprechen. Die Benutzerkommandos werden im Gegensatz zu den Regeln nicht automatisch aktiv, sobald ihre Vorbedingungen erfüllt sind, sondern müssen explizit durch den Spieler aufgerufen werden.

### 3.2.4 Konzepte des Regelmodells

*Aktivierungs-/Deaktivierungskonzept*

Das Aktivierungs- und Deaktivierungskonzept bezieht sich nur auf Aktivitäten. Die Aktivität besteht aus vier Teilen, dem Bedingungsteil und drei Aktionsteilen. Der erste Aktionsteil enthält alle Aktionen, die zum Zeitpunkt der Aktivierung ausgeführt werden. Im zweiten Aktionsteil werden die Aktionen beschrieben, die über den Zeitraum der Aktivierung verarbeitet werden, und der dritte Aktionsteil umfasst die Aktionen, die zum Zeitpunkt der Deaktivierung durchgeführt werden.

```
activity_rule Entwickler_spezifiziert

structure
 ein_Entwickler : Entwickler;
 ...
constraints
 exists(soll_produzieren(ein_Entwickler, die_Spezifikation))
declare
 ...
activating_part
 a := LEERE_ANFORDERUNG;
 Send_Player_Message begonnen_zu_spezifizieren(ein_Entwickler);
 ...
active_part
 dt := to_real(del_ta.t) * spezifiziert.ZeitAnteil/100.0;
 ...
deactivating_part
 delete(spezifiziert);
 Send_Player_Message aufgehoert_zu_spezifizieren(ein_Entwickler);

end rule;
```

*Abb. 3.1:* Beispiel einer Aktivität

Abbildung 3.1 zeigt das Beispiel einer Aktivität, die den Namen „Entwickler_spezifiziert“ trägt. Die „activity_rule“ besteht aus einem Strukturteil (structure), einem Bedingungsteil (constraints) und einem Aktionsteil. Im Strukturteil wird die Variable „ein_Entwickler“ angelegt, die den Entitätstyp „Entwickler“ hat. Im Bedingungsteil wird überprüft, ob es bereits eine Relation „soll_produzieren“ mit diesem Entwickler und dem Spezifikationsdokument gibt. Ist dies der Fall, so kann die Regel feuern. Im Deklarationsteil

(declare) können noch zusätzliche Variablen angegeben werden. Kann die Regel feuern, dann muss geprüft werden, ob die Aktivität gerade erst aktiviert wurde, ob sie schon aktiv ist, oder ob sie gerade deaktiviert wurde. Je nachdem welcher Fall zutrifft wird ein anderer Aktionsteil der Regel ausgeführt. Wurde die Aktivität gerade aktiviert, dann wird der „activating_part“ ausgeführt, das bedeutet in unserem Fall, es wird eine Nachricht an den Spieler geschickt, dass der Entwickler begonnen hat zu spezifizieren. War die Aktivität bereits aktiv, dann wird der „active_part“ verarbeitet. In unserem Beispiel wird dann eine Berechnung durchgeführt. Der „deactivating_part“ deckt den Fall ab, dass die Aktivität beendet wurde. In unserem Fall hat der Entwickler die Erstellung des Spezifikationsdokuments abgeschlossen, weshalb die Relation „spezifiziert“ gelöscht werden kann. Außerdem wird dem Spieler in einer Meldung angezeigt, dass der Entwickler aufgehört hat zu spezifizieren.

*Feuerungssemantik*

Das Feuerungskonzept gilt nur für diskrete Regeln und Benutzerkommandos. Diese Regeln bestehen nur aus zwei Teilen, einem Bedingungs- und einem Aktionsteil. Der Aktionsteil wird nur dann ausgeführt, wenn alle Vorbedingungen im Bedingungsteil erfüllt sind.

```
discrete_rule Entwickler_beginnt_zu_analysieren
 ...

structure
 ein_Entwickler   : Entwickler;
 soll_analysieren : soll_produzieren(ein_Entwickler, die_Notizen);
constraints
 not exists_extended(bearbeitet(ein_Entwickler, *));
action_part
 create relation produziert with
 wer := ein_Entwickler; -- Rollen...
 was := die_Notizen;
 end create;
 ...

end rule;
```

*Abb. 3.2:* Beispiel einer diskreten Regel mit Feuerungssemantik

Abbildung 3.2 zeigt das Beispiel einer diskreten Regel mit Feuerungsseman-

tik, die den Namen „Entwickler_beginnt_zu_analysieren“ trägt. Wir sehen hier wieder den Strukturteil (structure), in dem eine Variable „ein_Entwickler“ vom Entitätstyp „Entwickler“ und eine Variable „soll_analysieren“ vom Relationstyp „soll_produzieren“ angelegt werden. Wir können erkennen, dass im Strukturteil definierte Variablen sofort in anderen Deklarationen verwendet werden können, das heißt die Variable „ein_Entwickler“ wird bereits in der Relation „soll_produzieren“ verwendet. Der Bedingungsteil (constraints) überprüft, ob der Entwickler schon an einem anderen Dokument arbeitet. Ist dies der Fall, dann kann er nicht mit der Analyse beginnen, das bedeutet, die Regel kann nicht feuern. Sofern die Bedingungen eingehalten werden, kann zu guter Letzt der Aktionsteil ausgeführt werden. In unserem Beispiel wird eine neue Relation mit dem Namen „produziert“ erzeugt in der festgehalten wird, dass der Entwickler nun mit der Bearbeitung der Analysenotizen beginnt.

Die Beispiele 3.1 und 3.2 zeigen nur sehr einfache Regeln. Eine Vielzahl der Regeln ist aber weitaus umfangreicher und komplexer. Das Auffinden von Eckdaten im Regelwerk des QS-Modells und die damit verbundene Analyse einiger Regeln haben sich als schwierig und ausgesprochen aufwändig herausgestellt. Bei der Analyse einiger Regeln musste die Hilfe der Modellbauer in Stuttgart in Anspruch genommen werden.

## *3.3 Modelle und ihre Eigenschaften*

Um Softwareentwicklungsprojekte simulieren zu können werden Modelle benötigt. Leider gibt es nicht das eine Modell das alles kann, sondern es werden spezifische Modelle benötigt, welche die interessanten Teilbereiche von Softwareentwicklungsprojekten abdecken. Der Simulator kann aber nicht nur für Modelle, die Softwareentwicklungsprojekte realisieren, eingesetzt werden, sondern auch für jeden beliebigen anderen Bereich, in welchem Projekte durchgeführt werden müssen (z. B. in der Betriebswirtschaftslehre). Die Probleme, die in diesen Projekten auftreten sind zwar andere, aber auch hier kann die Ausbildung der Studierenden am Simulator für die erfolgreiche Durchführung von Projekten in der Praxis entscheidend sein.

Die Modelle, die am Simulator eingesetzt werden, sind empirisch gehaltvoll und helfen dadurch den Studierenden sich auf mögliche Situationen in realen Projekten vorzubereiten. In diesem Abschnitt werden die bestehenden Modelle kurz beschrieben. Außerdem wird erklärt, warum im Weiteren nur das Qualitätssicherungsmodell verwendet wurde.

### 3.3.1 Qualitätssicherungsmodell (QS-Modell)

Das QS-Modell ist das erste vollständig entwickelte Modell, das für die SESAM-Simulation eingesetzt wurde. Es beruht auf empirischen Daten [JONES] und Effekten [BOEHM], die in [DRAPPAb] zusammengefasst wurden. In der Diplomarbeit von Michael Joos [JOOS] wird die Konzeption und die Realisierung[2] dieses Projektmodells beschrieben.
Das QS-Modell bildet kleine bis mittlere Auftragsprojekte mit einem Umfang von 200 bis 1200 AFPs (Adjusted Function Points) ab, das heißt es realisiert die für diese Projektklasse relevanten Objekte (z.B. Entität Kunde), Beziehungen und Aktivitäten. Die Effekte, die im QS-Modell integriert wurden, entsprechen realen Problemen, mit denen Projekte dieser Größenordnung zu kämpfen haben. Das Modell konzentriert sich hier vor allem auf konstruktive, analytische und korrigierende Maßnahmen. Dadurch ist sowohl die Durchführung von Qualitätssicherungsmaßnahmen als auch gutes Personalmanagement entscheidend für den erfolgreichen Projektabschluss.

Als Software-Prozessmodell wurde das Wasserfallmodell gewählt, welches von Winston Royce 1970 definiert wurde [ROYCE]. Das Modell überlässt dem Spieler die Entscheidung, wann er welche Tätigkeiten anordnet.

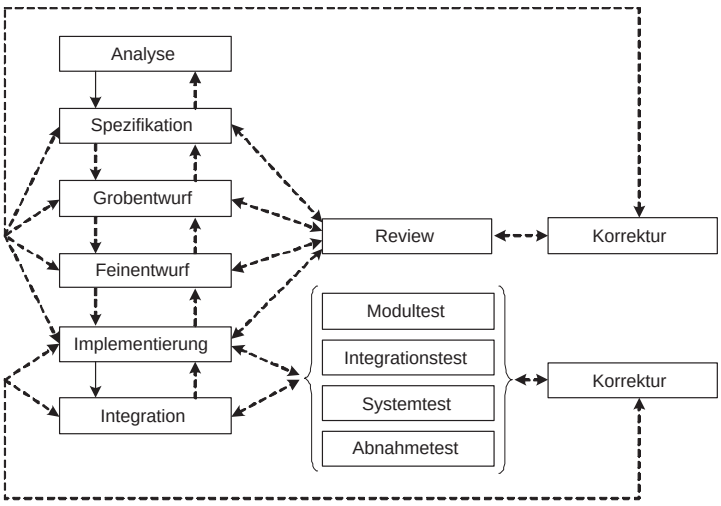



*Abb. 3.3:* Mögliche Abläufe im QS-Modell

[2] Änderungen vor allem in der Quantifizierung des Regelmodells wurden in einem separaten Dokument „Beschreibung der Implementierung des QS-Modells“ [DRAPPAa] festgehalten.

Das Wasserfallmodell wurde um zusätzliche konstruktive und analytische Maßnahmen erweitert. Die möglichen Abläufe innerhalb dieses, so genannten QS-Modells, werden in Abbildung 3.3 dargestellt. Die gestrichelt gezeichneten Pfeile zwischen den Aktivitäten weisen darauf hin, dass der Spieler nicht streng nach dem Wasserfallmodell vorgehen muss. Es gibt zwar eine vorgegebene Sequenz zwischen den Phasen, der Spieler muss diese aber nicht einhalten. Er wird allerdings zumindest dazu gezwungen ein Analysedokument zu erstellen und die Implementierung durchzuführen. Alle anderen Phasen zwischen Analyse und Implementierung darf er überspringen. Somit kann der Spieler nicht davon abgehalten werden den Code-and-Fix Ansatz auszuprobieren. Weiter ist zu erkennen, dass Rücksprünge in frühere Phasen ermöglicht werden. Diese Rücksprünge sind jedoch nur solange möglich, solange das Dokument zu dem gesprungen werden soll noch Fehler enthält. Fehlerhafte Dokumente können beliebig oft geprüft und korrigiert werden. Wichtig ist jedoch, dass die Korrektur eines Reviews erfolgreich abgeschlossen werden muss, bevor eine neue Prüfung desselben Dokuments durchgeführt werden kann. Nach der Korrektur können beliebige Aktivitäten ausgeführt werden.

*Lernziele des QS-Modells*

Wir haben bereits bei den Lernzielen des Simulators im Kapitel 2 festgehalten, dass das Hauptanliegen dieses Ansatzes im Trainieren von Projektmanagementfunktionen liegt. Das Erlernen von Projektmanagementfähigkeiten steht daher in allen Modellen im Vordergrund. Die Studierenden sollen lernen, dass Projekte im Voraus gründlich geplant und vorbereitet werden müssen. Weitere Probleme ergeben sich im Bereich Personaleinsatz. Die Spieler sollen merken, wie wichtig es ist die richtigen Mitarbeiter aus dem Personalangebot auszuwählen und sie dann entsprechend ihrer Qualifikation einzusetzen. Ebenso soll der zukünftige Projektleiter lernen, welche Entscheidungen in welchen Situationen getroffen werden müssen, damit auch kritische Projekte noch zu einem erfolgreichen Abschluss gebracht werden können.

Mit Hilfe des QS-Modells sollen die Studenten aber nicht nur Projektmanagementfunktionen trainieren, sondern auch den Nutzen verschiedener Qualitätssicherungsmaßnahmen erfahren und lernen, diese sinnvoll einzusetzen.

Diese Arbeit wird sich nur auf das klassische QS-Modell beziehen, da es zum Zeitpunkt dieser Arbeit das einzige praktisch erprobte Modell war, das bereits in diversen Lehrveranstaltungen zum Einsatz gekommen ist. Dadurch hatte ich für dieses Modell auch genügend Spielverläufe für eine genauere

Analyse zur Verfügung.

### 3.3.2 QS-Modell verfeinerter Ansatz (QSVA-Modell)

Ein Modell, das bereits vollständig entwickelt wurde, ist das QSVA-Modell, eine verfeinerte Variante der QS-Modells, das in [HAMPP] näher beschrieben wird. In diesem Modell werden die Reviewphasen noch genauer unterteilt.

Im QS-Modell kann der Student zwei oder drei Mitarbeiter anweisen einen Review durchzuführen. In den genauen Ablauf des Reviews kann der Spieler allerdings nicht mehr eingreifen. Die Mitarbeiter erstellen völlig selbständig ihre Befunde über die Mängel im Prüfobjekt, die in mehreren Reviewsitzungen besprochen und in den Reviewbericht aufgenommen werden. Der Spieler erfährt nur wie viele Fehler und wie viele fehlende AFPs in den einzelnen Reviewsitzungen gefunden wurden und bekommt nach erfolgreichem Abschluss des Reviews eine Zusammenfassung über die gefundenen Mängel in Form eines Reviewberichts zurück.
In der verfeinerten Variante gibt es mehrere Kommandos für die Durchführung eines Reviews. Der Spieler muss für die einzelnen Sitzungen Termin, Moderator, Schriftführer und Gutachter festlegen. Außerdem muss er den Umfang des Prüfobjekts (den Teil des Dokuments) bestimmen, der in den einzelnen Sitzungen behandelt werden soll. Das Prüfobjekt wird dann wie ein Arbeitspaket geschnürt. Die Gutachter bereiten sich auf die Sitzung vor und notieren sich ihre Befunde, welche inhaltliche Fehler und Verluste im Prüfobjekt beinhalten. In der Reviewsitzung werden die Befunde alle Gutachter zusammengetragen und in einem Bericht festgehalten. Aufgrund dieses Reviewberichts wird zuletzt eine Empfehlung für das Prüfobjekt abgegeben.

#### *Lernziele des QSVA-Modells*

Der Spieler kann mit Hilfe dieses Modells lernen, wie Reviewsitzungen organisiert werden müssen, und welcher Umfang des Prüfobjekts in einer Sitzung sinnvoll bearbeitet werden kann. Außerdem lernt er, welche Auswirkungen gut bzw. schlecht gewählte Moderatoren, Schriftführer und Gutachter auf die Effektivität des Reviews haben.
Nachteilig ist jedoch, dass sich durch diese Verfeinerungen der Aufwand für den Spieler drastisch erhöht. Ein Spiel mit dem QSVA-Modell ist kaum noch an einem Tag spielbar, wohingegen ein Spiel mit dem QS-Modell innerhalb von vier Stunden beendet werden kann.

### 3.3.3 Motivation im QS-Modell

Eine andere Verfeinerung des klassischen QS-Modells beschäftigt sich mit Effekten des menschlichen Verhaltens in Softwareprojekten, genauer in der Diplomarbeit von Axel Dudler [DUDLER] nachzulesen.

*Lernziele des QS-Modells mit Motivationseffekten*

Der Spieler soll anhand dieses Modells lernen, welche Auswirkung die Motivation der Mitarbeiter auf den Projekterfolg hat und von welchen Faktoren die Motivation positiv oder negativ beeinflusst wird. Kann der Projektleiter seine Entwickler motivieren, dann steigt auch die Produktivität dieser Mitarbeiter. Das bedeutet aber nicht, dass eine schlechte Motivation nur zu einer schlechteren Produktivität führen kann. Unmotivierte Entwickler können sich mit ihrer Tätigkeit nicht mehr identifizieren und das kann, sofern der Projektleiter dies nicht früh genug erkennt, sogar zur Kündigung des Mitarbeiters führen.

### 3.3.4 Weitere Modelle

Es gibt laufend neue Ideen für zukünftige Modelle, von denen aber nur wenige bis zum heutigen Tag vollständig realisiert wurden.

*Erweiterung des QS-Modells um ein Wartungsmodell*

Im Moment gibt es auch Anstrengungen im Bezug auf eine Erweiterung des QS-Modells um ein anschließendes Wartungsmodell. Damit soll der Spieler lernen, dass das Projekt mit der Auslieferung des Systems an den Kunden noch lange nicht endet. Während des Einsatzes der Software kommt es zum Auftreten bisher noch nicht entdeckter Fehler, die unter Berücksichtigung des Änderungsdrucks so schnell wie möglich aus dem System entfernt werden müssen. Außerdem können zusätzliche Anforderungen auftreten, die in das bestehende System integriert werden müssen.

Während des Einsatzes der Software treten einerseits noch nicht entdeckter Fehler auf, die unter Berücksichtigung des Änderungsdrucks so schnell wie möglich aus dem System entfernt werden müssen, und andererseits zusätzliche Anforderungen, die in das System integriert werden müssen.

## 3.4 Auswertungswerkzeuge

Alle Kommandos des Spielers werden in einer Log-Datei festgehalten, die nach der Beendigung der Simulation vom Tutor ausgewertet werden kann. Für die Auswertung dieser Datei wurden spezielle Auswertungswerkzeuge entwickelt, die im Weiteren beschrieben werden.

### 3.4.1 SesamScore

Wie bereits erwähnt lassen sich die Interaktionen zwischen Spieler und System während des Spielverlaufs in einer Protokolldatei abspeichern, das heißt diese Datei enthält die Aktionen des Spielers mit Angabe der Parameter und entsprechend die Nachrichten des Systems. Zur Auswertung dieser Protokolldatei wurde das Werkzeug SesamScore[3] entwickelt. Mit Hilfe dieses Tools können Protokolldateien nach bestimmten Aktionen und Reaktionen des Systems und deren Parametern durchsucht werden. Die Auswertungsergebnisse können von SesamScore graphisch oder in Tabellen aufbereitet und dargestellt werden. Auch die Gegenüberstellung mehrerer Spielverläufe in einem Diagramm ist möglich.

Informationen aus einer oder mehreren Protokolldateien können somit als Diagramm dargestellt werden. Dazu werden Auswertungsschemata benötigt, die beschreiben, wie eine oder mehrere Protokolldateien ausgewertet werden sollen. Die für das Diagramm benötigten Informationen werden als Anfrage angegeben, mit der bestimmt wird, welche Informationen dargestellt werden sollen. Die Art und Weise wie diese Daten angezeigt werden sollen, wird über das Layout beschrieben. Anfrage und Layout bilden somit ein Schema.

Mit SesamScore können Tabellen, Balken-, Linien- und Ganttdiagramme erstellt werden. Es gibt bereits vordefinierte Schemata, die den Tutor bei der Analyse von Projektverläufen bzw. bei der Gegenüberstellung mehrerer Spielverläufe unterstützen. Welche dieser vordefinierten Schemata ich für die Analyse von Spielverläufen verwendet habe, und was diese Schemata aussagen, werde ich in Kapitel 4 genauer ausführen.

[3] Genauere Informationen zu diesem Werkzeug findet der Leser im dazugehörigen Handbuch.

### 3.4.2 SesamAlyzer

Das Auswertungswerkzeug SesamAlyzer[4] bietet die Möglichkeit, die in einer Protokolldatei abgespeicherten Zustandsänderungen auszuwerten und graphisch aufzubereiten. Es lassen sich beliebige Werte aus dem Situationsmodell und ihre Verläufe über die Projektzeit ausgeben. Dieses Tool ermöglicht mehrere Darstellungen wie zum Beispiel die Darstellung und den Vergleich interner Zustände des simulierten Projekts, oder die Darstellung von Attributwertverläufen in Liniendiagrammen. Auch die Darstellung von Attributwerten zu bestimmten Zeitpunkten in Balkendiagrammen und Tabellen ist möglich.

Da dieses Tool aber zum Zeitpunkt meiner Projektarbeit noch nicht verfügbar war, konnte ich mich nicht näher damit auseinandersetzen. Ich habe meine Aufmerksamkeit stattdessen SesamScore gewidmet, dass zu diesem Zeitpunkt bereits erprobt war und mir helfen konnten, die Bewertungskriterien für das QS-Modell zu finden.

Beide Werkzeuge sind sehr komplex, da es sich um generische Auswertungswerkzeuge handelt, die für alle Modelle eingesetzt werden können. Sowohl SesamScore, als auch der SesamAlyzer holen sich zuerst Informationen über das Modell. Der SesamAlyzer bekommt diese Informationen aus der Datei, welche die Daten über den Spielverlauf enthält. In dieser Datei steht zu Beginn das Schemamodell, in dem alle Attributs-, Entitäts- und Relationstypen definiert sind. SesamScore dagegen benötigt eine zusätzliche Datei, in der die Namen der vorhandenen Kommandos und Nachrichten definiert sind. Es verwendet diese Daten dann, um zu prüfen, ob die Auswertungsschemata und die Protokolldatei des Spielverlaufs zueinander passen.
Für beide Werkzeuge gilt selbstverständlich, dass die Auswertungsschemata zum Modell passen müssen.

Der große Nachteil dieser Auswertungswerkzeuge liegt darin, dass sie die „interessanten“ Daten nur aufbereiten und anzeigen können. Das bedeutet, dass es weiterhin die Arbeit des Tutors ist, anhand der Diagramme positive bzw. negative Effekte in den Spielverläufen zu erkennen und deren Auswirkungen auf andere Bereiche des Projekts dem Spieler begreifbar zu machen. Die neue generische Erklärungskomponente soll dem Lehrveranstaltungsleiter diese Arbeit abnehmen.

[4] Genauere Informationen zu diesem Werkzeug findet der Leser im dazugehörigen Handbuch.

*Kapitel 4*

# ANALYSE VON PROJEKTVERLÄUFEN

Bevor näher auf die generische Erklärungskomponente eingegangen werden kann, müssen Bewertungskriterien identifiziert werden, anhand derer die Erklärungskomponente eine Auswertung von Projektverläufen vornehmen kann. Dieses Kapitel beschreibt interessante Aspekte von Spielen, die durch die Analyse von Spielverläufen aufgedeckt wurden und zur Identifikation der in Kapitel 5 aufgelisteten Bewertungskriterien geführt haben.

## *4.1 Datenbasis*

Bei der Datenbasis, die für die Identifikation der relevanten Bewertungskriterien herangezogen wurde, handelt es sich um Spiele, die in der Lehrveranstaltung „Systementwicklungsprozess“ im Sommersemester 2002 durchgeführt wurden. An dieser Lehrveranstaltung nahmen 21 Studenten teil, die sich alle im dritten Studienabschnitt befanden und bereits ein viermonatiges Praktikum außerhalb der Universität absolviert hatten. Sie wurden in zehn Zweiergruppen und einer Einzelspielergruppe (trägt im Folgenden die Bezeichnung sep_105) eingeteilt. Da es sich für den Einzelspieler bereits um das dritte Spiel handelte, stand für die Auswertung somit auch der Projektverlauf eines „erfahrenen“ Spielers zur Verfügung, der mit denen von „Neulingen“ verglichen werden konnte. Gespielt wurde das klassische QS-Modell mit 200 AFPs. Damit die Spieler nicht den Tücken des Simulators ausgeliefert waren, wurden diese vor dem Spiel vom Lehrveranstaltungsleiter aufgezeigt. Besonders gute Projekte im Sinne der Zielerreichung (siehe Tabelle 4.1) wurden von den Gruppen sep_102, sep_105 und sep_111 durchgeführt. Gruppe sep_108 bzw. sep_103 haben mehrere Zielvorgaben nicht erfüllt. Anhand dieser fünf Projektverläufe werde ich im Abschnitt 4.3 richtige bzw. falsche Entscheidungen und ihre Auswirkungen auf das Projekt aufzeigen.

## 4.2 Auswertung von Projektverläufen

Der Tutor hat es im Falle des QS-Modells nicht leicht den Erfolg oder Misserfolg eines Projekts festzustellen. Er kann es sich zwar einfach machen und die Auswertung der Spiele nur aufgrund der erreichten Zielvorgaben vornehmen, dadurch wir jedoch der Lernerfolg der Studenten auf beinahe Null reduziert. Um den Spielern ihre Fehler im Projektmanagement aufzeigen zu können, und auf diese Weise den Lernerfolg der Studenten zu maximieren, muss der Tutor jeden Projektverlauf genau analysieren, denn der Erfolg bzw. Misserfolg eines Projekts hängt von vielen sich überlagernden Effekten ab, die genau untersucht werden müssen. Je genauer der Tutor jeden Projektverlauf analysiert, desto mehr Informationen kann er den Spielern zu ihren Projekten geben, und desto höher ist der Lernerfolg der einzelnen Studenten [NOTTER].
Natürlich sollte der Tutor auch den Projektplan jeder Gruppe analysieren und diesbezügliche Missstände aufzeigen und Tipps für ein besseres Vorgehen geben. Leider konnte festgestellt werden, dass der Projektplan der Spieler kaum vom Tutor näher betrachtet wird. Möglicherweise würde eine Verpflichtung der Abgabe des Projektplans bereits zu Verbesserungen auf diesem Gebiet führen, da bei vielen Spielern eine eher schlampige Projektplanung beobachtete werden konnte. Allerdings wird beim SESAM-Spiel eine schriftliche Projektplanung nicht erzwungen, es wird nur darauf hingewiesen, dass ein Projektplan nützlich sein kann. Es liegt also im Ermessen des Tutors den Spielern das Erstellen eines Projektplans nahe zu legen.

Im folgenden Abschnitt werden die Erfolgsfaktoren anhand von „interessanten“ Aspekten einzelner Spielverläufe gezeigt. Dabei wird so vorgegangen, wie der Tutor bisher die Auswertung vornehmen musste. Dieser Weg wurde aus zwei Gründen gewählt. Zum einen, um den Analyseaufwand des Tutors zu verdeutlichen, und zum anderen, um die interessanten Effekte leichter anhand von SesamScore-Diagrammen aufzeigen bzw. beschreiben zu können.
Im Kapitel 5 werden die Bewertungskriterien aufgezeigt, die aus diesen „interessanten“ Aspekten gewonnen werden konnten.

## 4.3 Interessante Aspekte in Projektverläufen

Für die händische Analyse von Projektverläufen hat es sich als hilfreich herausgestellt, sich vor genauerer Betrachtung jedes einzelnen Spielverlaufs einen Überblick über sämtliche Spielverläufe zu verschaffen. Mit Hilfe des Werkzeugs „SesamScore“ ist es möglich Daten übersichtlich anzeigen zu lassen.

Dazu kann auf bereits bestehende Schemata zurückgegriffen werden, die die Daten anschaulich präsentieren. Die Diagramme helfen dem Tutor dabei Anhaltspunkte zu finden, die er dann anhand der Projektdaten genauer untersuchen muss, um so eine Bewertung des Spiels vornehmen zu können.

Fasst man diese Diagramme zu einem Dokument zusammen, dann kann dieses Dokument einerseits dazu verwendet werden, um dem Spieler einen Überblick über seinen Projektverlauf zu geben, andererseits dient es aber auch als Diskussionsgrundlage für den Tutor, der damit die guten und schlechten Entscheidungen der Spieler in den einzelnen Projektverläufen aufzeigen und erklären kann. In Anhang F findet der Leser ein derartiges Dokument, das einen Überblick über den Projektverlauf der Gruppe sep_108 gibt. Anhand von ausgesuchten SesamScore-Diagrammen werden nun „interessante“ Aspekte von Spielverläufen gezeigt und diskutiert. Um die Vielzahl von Diagrammen leichter überschauen zu können, wurden sie wie folgt gruppiert:

- Zielvorgaben: Gibt eine Zusammenfassung über die erreichten bzw. nicht erreichten Zielvorgaben aller Gruppen.
- Mitarbeiter: Dieser Abschnitt beschäftigt sich mit Diagrammen bzw. Tabellen, die den Mitarbeitereinsatz betreffen.
- Projektverlauf: Diese Gruppe enthält Diagramme, die etwas über den gesamten Projektverlauf, oder über Teile davon aussagen.
- Qualitätssicherung: Diagramme und Tabellen dieser Gruppe beschäftigen sich mit dem Einsatz von Qualitätssicherungsmaßnahmen.
- Dokumente: Dieser Abschnitt enthält Diagramme, die verschiedene Aspekte von Dokumenten genauer untersuchen.

### *4.3.1 Zielvorgaben*

Die Tabelle „Zielvorgaben“ gibt einen Überblick darüber, welche Zielvorgaben von den einzelnen Gruppen erreicht bzw. nicht erreicht wurden. Es werden Dauer, Kosten, erreichter Umfang in AFPs für Code und Handbuch sowie die Anzahl der Fehler pro KLOC und pro Seite im Handbuch angegeben.

Die Tabelle 4.1 zeigt eine Gegenüberstellung aller Spiele. Die erreichten Zielvorgaben wurden grün, die nicht erreichten wurden rot markiert.
Es fällt auf, dass nur die Gruppe sep_102 alle Zielvorgaben erfüllt hat. Am

| Gruppe | Dauer | Kosten | AFPs im Code in % | Fehler\KLOC | AFPs im HB in % | Fehler/Seite im HB |
|---|---|---|---|---|---|---|
| sep_101 | 283 | 464280 | 95,94 | 14,08 | 96,52 | 0,32 |
| sep_102 | 269 | 428760 | 98,14 | 8,07 | 96,53 | 0,39 |
| sep_103 | 280 | 607640 | 90,96 | 29,26 | 97,41 | 0,34 |
| sep_104 | 269 | 561080 | 92,66 | 19,73 | 96,09 | 0,41 |
| sep_105 | 233 | 303960 | 96,74 | 12,92 | 96,68 | 0,21 |
| sep_108 | 259 | 415560 | 93,72 | 16,04 | 93,56 | 0,43 |
| sep_109 | 212 | 469560 | 96,75 | 12,50 | 93,30 | 0,57 |
| sep_110 | 214 | 474760 | 96,62 | 11,25 | 93,14 | 0,61 |
| sep_111 | 245 | 475080 | 97,48 | 11,36 | 95,33 | 0,34 |
| sep_112 | 282 | 455000 | 97,10 | 9,00 | 96,31 | 0,28 |

*Tab. 4.1:* Zielvorgaben

häufigsten wurden die Zielvorgaben „Kosten“ und „Fehler/KLOC“ nicht eingehalten. Die geforderten Zielvorgaben für das Handbuch hingegen wurden von vielen Gruppen erreicht.

Warum sich der Tutor bei der Bewertung von Spielen nicht nur an der Anzahl der erreichten Zielvorgaben orientieren sollte, wird durch genauere Betrachtung der Gruppe sep_105 deutlich. Bei dieser Gruppe muss entscheiden werden, wie schwerwiegend es ist, dass die eine Zielvorgabe von 12 Fehlern pro KLOC von dieser Gruppe um nur 0,92 Fehler verfehlt wurde. Dabei muss berücksichtigt werden, dass das Projekt schon 37 Tage früher abgegeben werden konnte, und mehr als 140 000 DM gespart wurden.
Die Gruppe hätte demnach noch genügend Zeit und finanzielle Mittel zur Verfügung gehabt, um den Code noch zu verbessern. Es ist jedoch denkbar, dass der Kunde bereits mit dem aktuellen Produkt zufrieden wäre, da er weniger dafür bezahlen muss und es auch schon viel früher einsetzen kann.

Vor allem der Zeit- bzw. der Kostenfaktor kann vom Spieler in der Simulation sehr schwer abgeschätzt werden, da die Mitarbeiter keine Abschätzung abgeben, wie lange sie ungefähr für eine Tätigkeit benötigen.
Dieses Problem wird anhand des folgenden Beispiels deutlich: Nehmen wir an, ein Spieler hat bereits alle Zielvorgaben bis auf die „Fehler/KLOC“ erreicht. Der Spieler weiß, dass er noch einige Tage und genügend finanzielle Mittel zur Verfügung hat, um die Qualität des Codes noch mit Hilfe eines letzten Tests zu verbessern. Er führt also den Test durch und korrigiert die gefundenen Fehler. Danach muss er jedoch feststellen, dass er dadurch die

vorgegebene Zeit von 270 Tagen um einige Tage überschritten hat. Der zusätzliche Test kann zu zwei verschiedenen Resultaten geführt haben:

1. Der Spieler hat die Zielvorgabe „Fehler/KLOC“ erfüllt, aber dafür die Zielvorgabe „Dauer“ nicht erreicht. In diesem Fall hätte sich durch die Entscheidung für den letzten Test nichts an der Bewertung des Spiel geändert, es sei denn, der Tutor gewichtet die einzelnen Bewertungskriterien verschieden schwer.

2. Der Spieler hat die Zielvorgabe „Fehler/KLOC“ trotzdem nicht erfüllt und hat daduch zwei, statt einer nicht erfüllten Zielvorgabe. In diesem Fall würde der Spieler schlechter beurteilt werden, als zuvor.

Das Problem für den Spieler liegt demnach darin, das Projekt zum richtigen Zeitpunkt zu beenden, auch wenn er nicht alle Zielvorgaben erreicht hat.
Deshalb sollten bei der Beurteilung von Spielen nicht nur erfüllte bzw. nicht erfüllte Zielvorgaben bewertet, sondern Fehlentscheidungen des Spielers im Projektverlauf gesucht und gewichtet werden.
Die faire Bewertung von Spielen ist aber nicht das Ziel dieser Arbeit und wird darum nicht weiter diskutiert. Der Tutor kann aber die in Kapitel 5 zusammengefassten Bewertungskriterien als Bewertungsgrundlage für Spiele verwenden. Wie die einzelnen Kriterien gewichtet werden, liegt jedoch in seinem/ihrem Ermessen.

Betrachten wir im Weiteren, wie es zu den Ergebnissen in der Tabelle 4.1 gekommen ist, und welche Probleme in den einzelnen Projektverläufen aufgetreten sind.

### *4.3.2 Mitarbeiter*

Im folgenden Abschnitt werden Diagramme besprochen, die den Mitarbeitereinsatz behandeln. Außerdem werden positive und negative Auswirkungen des Mitarbeitereinsatzes auf den Projektverlauf diskutiert.

#### *Anzahl eingesetzter Mitarbeiter*

Dieses Diagramm gibt an, wie viele Mitarbeiter in den einzelnen Phasen (Spezifikation, Entwurf, Modulspezifikation, Code, Handbucherstellung, Modultest, Integrationstest, Systemtest) eingesetzt wurden. Wurden in einer Phase

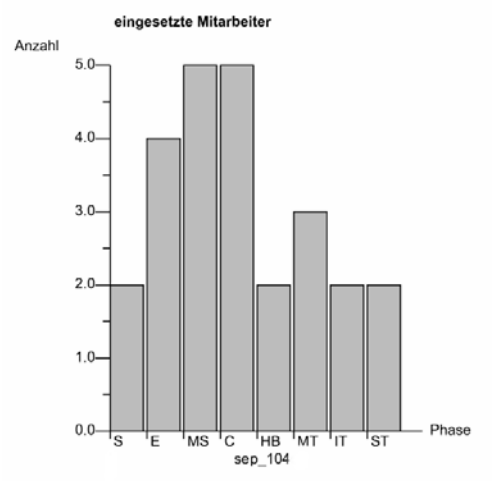
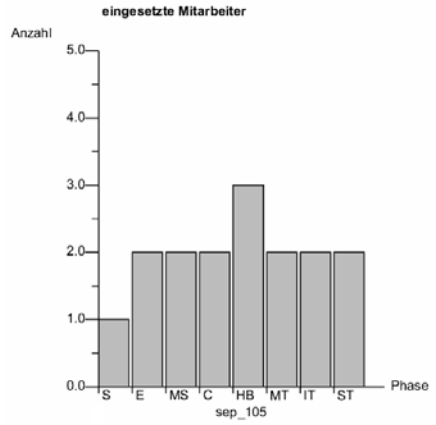



*Abb. 4.1:* Anzahl eingesetzter Mitarbeiter (sep_104)

mehr als zwei Mitarbeiter eingesetzt, dann können daraus Rückschlüsse auf die Kosten gezogen werden. Je mehr Mitarbeiter in einer Phase eingesetzt werden, desto höher ist der Kommunikationsaufwand, und desto länger benötigt jeder einzelne Mitarbeiter für seine Tätigkeit. Das bedeutet, dass die Phase umso länger dauert, je mehr Mitarbeiter an ihr beteiligt sind. Längere Phasen bedeuten natürlich auch höhere Projektkosten.

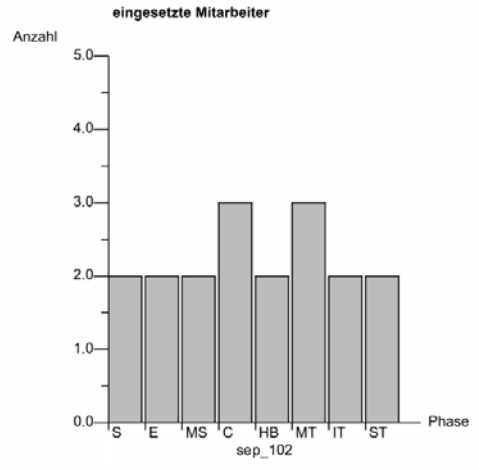
Die Abbildung 4.1 zeigt, dass die Gruppe sep_104 in den Phasen Modulspezifikation und Code je fünf Mitarbeiter eingesetzt hat. Dies ist ein Grund dafür, dass das benötigte Budget dieser Gruppe mit 561 080 DM, wie wir es in der Tabelle 4.1 gesehen haben, im Vergleich zu der geforderten Zielvorgabe von 450 000 DM, sehr hoch lag.

Die Gruppe sep_105 hat, laut Abbildung 4.2, für die Spezifikationsphase einen Mitarbeiter, für die Handbucherstellung drei und in allen anderen Phasen immer zwei Mitarbeiter eingesetzt. Dadurch blieb der Kommunikationsaufwand gering und die Kosten des Projekts betrugen nur 303 960 DM.
Natürlich hängen die Kosten auch noch von anderen Faktoren ab, wie der Qualifikation der Mitarbeiter, den Lücken im Personalmanagement und vielen mehr. Diese Aspekte werden aber noch besprochen.




| | |
|---|---|
| S: Spezifikation | HB: Handbuch |
| E: Entwurf | MT: Modultest |
| MS: Modulspezifikation | IT: Integrationstest |
| C: Codierung | ST: Systemtest |

*Abb. 4.2:* Anzahl eingesetzter Mitarbeiter (sep_105)




| | |
|---|---|
| S: Spezifikation | HB: Handbuch |
| E: Entwurf | MT: Modultest |
| MS: Modulspezifikation | IT: Integrationstest |
| C: Codierung | ST: Systemtest |

*Abb. 4.3:* Anzahl eingesetzter Mitarbeiter (sep_102)

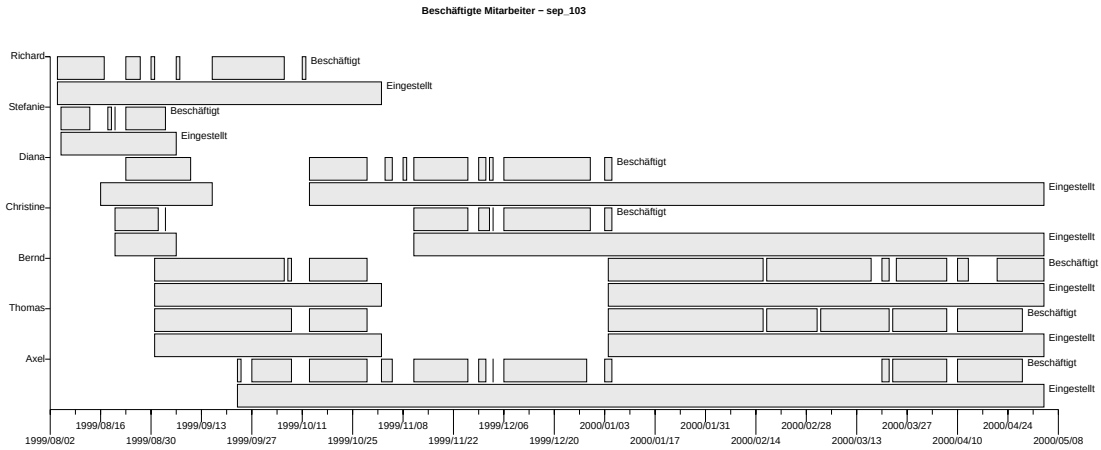



*Abb. 4.4:* Beschäftigte Mitarbeiter (sep_103)

Auch die Gruppe sep_102 ist, wie in Abbildung 4.3 zu sehen, bei der Zuteilung von Mitarbeitern richtig vorgegangen. Lediglich in der Codierungsphase und für die Modultests wurden drei Mitarbeiter eingesetzt. Alle anderen Phasen wurden von zwei Mitarbeitern durchgeführt. Dadurch blieb der Kommunikationsaufwand und somit auch die Kosten für die Erstellung der Dokumente gering. Betrachtet man jedoch die Zielvorgaben (Tabelle 4.1), so erkennt man, dass die Gruppe trotzdem 428 760 DM benötigt hat. Dies hat aber nichts mit der Anzahl der eingesetzten Mitarbeiter zu tun. Wir werden später anhand der Projektverlaufsdiagramme erkennen, dass ein Großteil der Kosten auf das intensive Testen der Gruppe zurückzuführen ist.

*Beschäftigte Mitarbeiter*

Dieses Diagramm zeigt, für jeden Mitarbeiter, der zum Projekt hinzugezogen wurde, zwei Balken an. Der untere Balken gibt an, in welchen Zeiträumen der Mitarbeiter eingestellt war. Der obere Balken hingegen sagt aus, zu welchen Zeitpunkten er tatsächlich beschäftigt war. Mit Hilfe dieses Diagramms können in der Spielanalyse Lücken im Personaleinsatz feststellt.

Die unproduktive Zeit von Mitarbeitern ist eines der Hauptprobleme mit denen die Spieler zu kämpfen haben. Das „Vergessen“ auf Mitarbeiter, die eingestellt sind, aber denen keine Tätigkeit zugewiesen wurde, kann fatale Folgen haben, wie man am Projektverlauf der Gruppe sep_103 in Abbil-


*Abb. 4.5:* Beschäftigte Mitarbeiter (sep_102)

dung 4.4 sehen kann. Nachdem Diana und Christine ihre Aufgabe Anfang Jänner beendet hatten, wurden sie für keine weiteren Tätigkeiten eingesetzt. Ein Grund dafür könnte sein, dass sie für die weiteren Projektphasen nicht qualifiziert genug waren. Die Gruppe hat aber darauf vergessen die beiden, anscheinend nicht mehr benötigten Mitarbeiterinnen, aus dem Projekt zu entlassen. Dies hat Diana und Christine somit ungewollt ganze vier Monate bezahlten Urlaub ermöglicht. Auch Axel blieb beinahe drei Monate untätig und kassierte immer noch sein Gehalt. Dies führte zu erheblichen Kosten, die eigentlich leicht vermieden hätten werden können.
Die Projektkosten dieser Gruppe von 607 640 DM (Tabelle 4.1) spiegeln diesen fatalen Fehler wieder, der auf einen schlampigen Projektplan zurückzuführen ist. Man sollte in diesem Fall den Projektplan dieser Gruppe genauer analysieren, und die Gruppe auf Fehler in der Projektplanung hinweisen.

Es muss an dieser Stelle angemerkt werden, dass manche Lücken im Personaleinsatz unvermeidlich sind. Deshalb ist es notwendig, den Grund für jede einzelne Lücke genauer zu analysieren und zu prüfen, ob der Spieler zu einem bestimmten Zeitpunkt im Projekt überhaupt eine andere Wahl gehabt hätte die Lücke sinnvoll zu füllen. Nur dann kann man dem Spieler eine genaue Erklärung dafür geben, welche Auswirkungen seine Entscheidungen im Projektverlauf gehabt haben.

Das Beschäftigungsdiagramm der Gruppe sep_102 in Abbildung 4.5 zeigt sehr gut, dass die Mitarbeiter am selben Tag an dem sie ihre letzte Tätigkeit

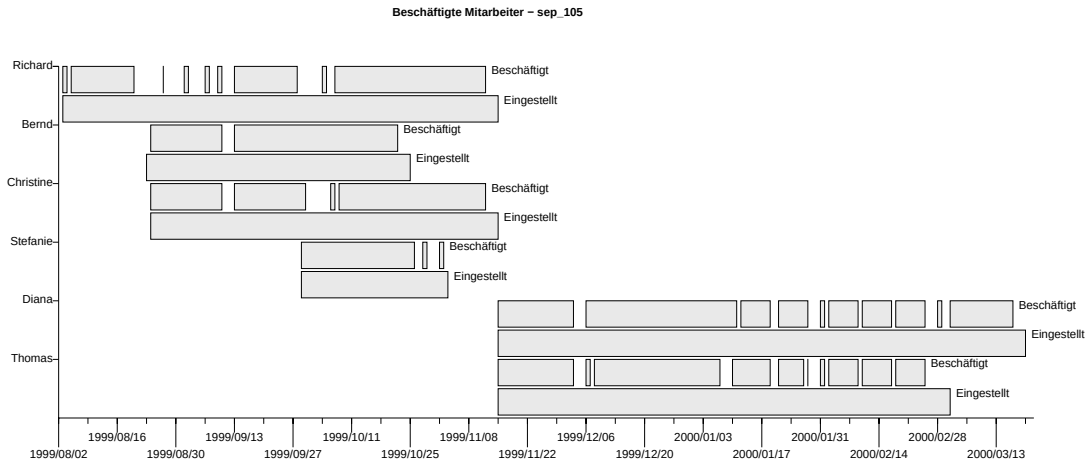



*Abb. 4.6:* Beschäftigte Mitarbeiter (sep_105)

beendet hatten, bereits entlassen wurden, da sie im zukünftigen Projektverlauf nicht mehr benötigt wurden. Diana und Axel wurden, wie im Diagramm ersichtlich noch am selben Tag entlassen, dies lässt auf eine gute Projektplanung schließen.

Abbildung 4.6 zeigt einen anderen interessanten Aspekt. Die Gruppe sep_105 hat das Projekt von zwei unterschiedlichen Projektteams durchführen lassen. Die Spieler haben die qualifiziertesten Mitarbeiter für die Spezifikations- und Entwurfsphase angestellt und sie nach Erfüllung ihrer Pflichten entlassen. Dann hat sich die Gruppe die besten Programmierer vom Markt geholt und das Projekt sehr schnell (bereits 37 Tage früher) und günstig (verbrauchtes Budget von nur 303960 DM) zu einem frühen Ende gebracht. Die Erfahrungen des ersten Projektteams gingen dabei jedoch vollständig verloren.

*Einfluss unproduktiver Zeit auf die Kosten*

Diese Tabelle stellt die Gesamtkosten, dem Gesamtaufwand und den Kosten pro Mannmonat gegenüber.

Die Kosten pro Mannmonat waren bei der Gruppe sep_103 in Tabelle 4.2 mit 31 995,50 am größten. Einer der Gründe war wohl die unproduktive Zeit der Mitarbeiter, die bei dieser Gruppe extrem hoch war. Ich verweise an diesem Punkt nochmals auf die Abbildung 4.4, in der wir die großen Lücken

**Einfluss unproduktiver Zeit**

| | sep_103 |
|---|---|
| K | 607640.0 |
| | |
| A | |
| | 18.99 |
| k | |
| | 31995.5 |

K: Kosten
A: Personenmonate
k: Kosten/Personenmonat

*Tab. 4.2:* Einfluss der unproduktiven Zeit (sep_103)

im Personaleinsatz entdeckt und analysiert haben.

Gruppe sep_102, deren Daten in der Tabelle 4.3 aufgelistet sind, benötigte sieben Personenmonate für die Fertigstellung des Projekts und verursachte dabei 15 181,30 DM Kosten pro Personenmonat. Dies kann einerseits auf eine gute Auslastung der Beschäftigung der Mitarbeiter zurückgeführt werden, andererseits hat die Gruppe in den einzelnen Phasen nie mehr als drei Mitarbeiter eingesetzt, wie wir in Abbildung 4.3 gesehen haben.

Die verursachten Kosten pro Personenmonat waren, wie aus der Tabelle 4.4 hervorgeht, bei der Gruppe sep_105 mit 19 717 DM etwas höher. Allerdings benötigte die Gruppe nur sechs Personenmonate um das Projekt zu beenden. Das verbrauchte Budget war bei dieser Gruppe mit nur 303 960 DM das Geringste. Betrachten wir nochmals die Erreichung der Zielvorgaben in der Tabelle 4.1, dann kann weiters festgestellt werden, dass die Gruppe sep_105 trotzdem schon 36 Tage vor der Gruppe sep_102 das fertige Projekt an den Kunden übergeben konnte.

*Anzahl der Inspektionen*

Dieses Diagramm veranschaulicht, wie oft der Spieler sich nach dem aktuellen Stand bzw. dem Fortschritt der Dokumente erkundigt hat, und wie oft er

**Einfluss unproduktiver Zeit**

| | sep_102 |
|---|---|
| K | 428760.0 |
| | |
| A | |
| | 28.24 |
| k | |
| | 15181.3 |

K: Kosten
A: Personenmonate
k: Kosten/Personenmonat

*Tab. 4.3:* Einfluss der unproduktiven Zeit (sep_102)

**Einfluss unproduktiver Zeit**

| | sep_105 |
|---|---|
| K | 303960.0 |
| | |
| A | |
| | 15.42 |
| k | |
| | 19717.0 |

K: Kosten
A: Personenmonate
k: Kosten/Personenmonat

*Tab. 4.4:* Einfluss der unproduktiven Zeit (sep_105)

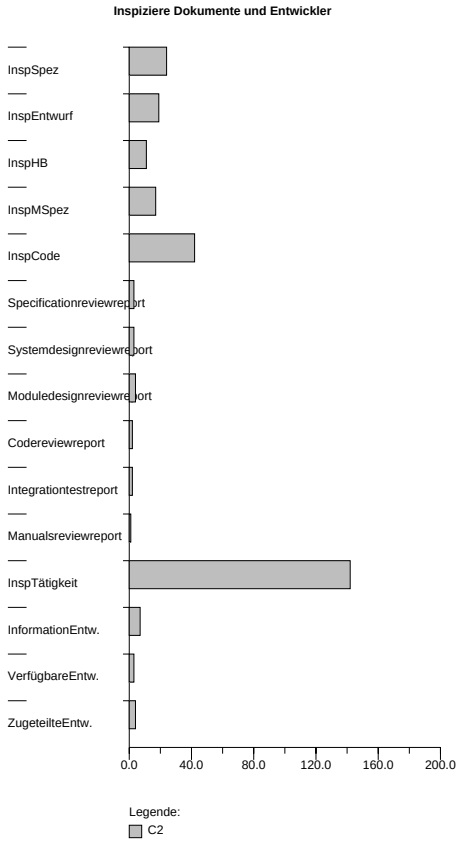



*Abb. 4.7:* Anzahl der Inspektionen (sep_102)

Informationen über verfügbare und dem Projekt zugeteilte Entwickler eingeholt hat. Die Anzahl der jeweiligen Inspektionen kann anhand der x-Achse abgelesen werden.
Wurde bei den Dokumenten häufig der aktuelle Zustand überprüft, könnte das darauf hindeuten, dass der Spieler den Überblick über sein Projekt verloren hat. Daraus kann abgeleitet werden, dass er keinen oder einen sehr schlechten Projektplan für sein Projekt aufgestellt hat. Auch das häufige Nachfragen, ob Mitarbeiter noch beschäftigt sind, deutet auf eine chaotische Projektplanung hin.
Die Gruppe sep_102 hat, wie in Abbildung 4.7 ersichtlich, mehr als 140mal die Tätigkeit ihrer Mitarbeiter überprüft und schnitt trotzdem am besten ab. Es ist anzunehmen, dass diese Gruppe jedes Mal wenn sie einer Person eine Tätigkeit zugeteilt hat, auch überprüft hat, ob die Person tatsächlich mit ihrer Arbeit begonnen hat. In diesem Fall können die vielen Inspektionen

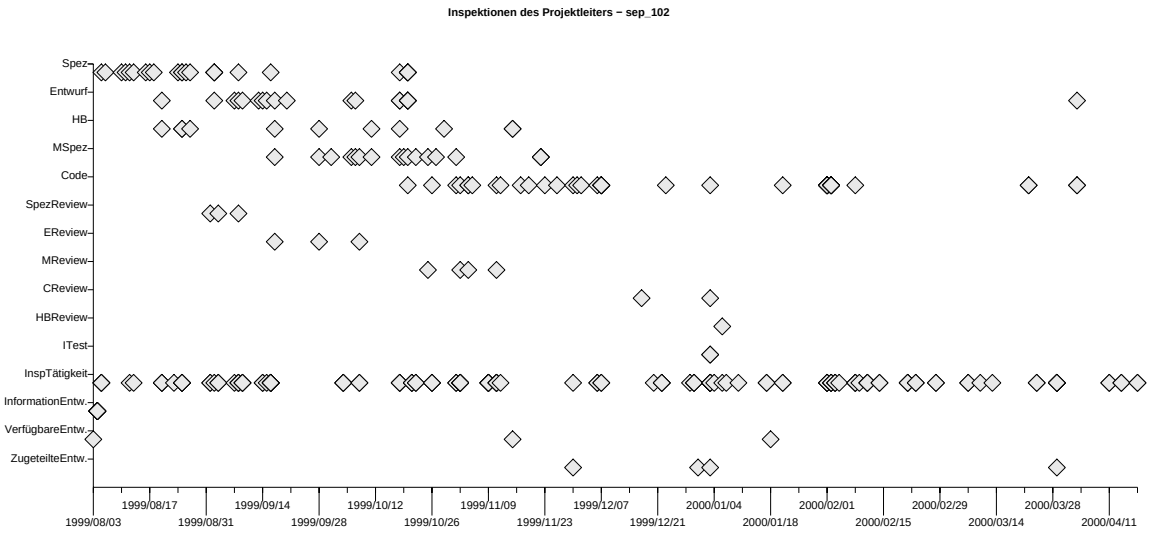



*Abb. 4.8:* Inspektionen des Projektleiters (sep_102)

nicht auf einen schlampigen Projektplan zurückgeführt werden.
Es muss daher noch darauf hingewiesen werden, dass die Anzahl der Inspektionen noch nichts über den Projekterfolg aussagt.

*Inspektionen des Projektleiters*

In diesem Diagramm werden alle Inspektionen des Projekts gezeigt. Im Gegensatz zum Diagramm „Anzahl der Inspektionen“, wird hier die Zeit als zweite Dimension angegeben. Der Zeitpunkt zu dem eine Inspektion stattfand, wird im Diagramm durch eine Raute dargestellt. Dadurch kann man feststellen, zu welchen Zeitpunkten im Projekt der Spieler besonders oft die Dokumente inspiziert hat.

Die Abbildung 4.8 zeigt ein solches Diagramm. Stellen, an denen punktuell viele Inspektionen durchgeführt wurden, können auf chaotische Projektplanung hinweisen.

### *4.3.3 Projektverlauf*

Die Diagramme in diesem Abschnitt beschäftigen sich alle mit dem Projektverlauf. Im Diagramm „Einfluss Konsistenz“ wird der Projektverlauf grob

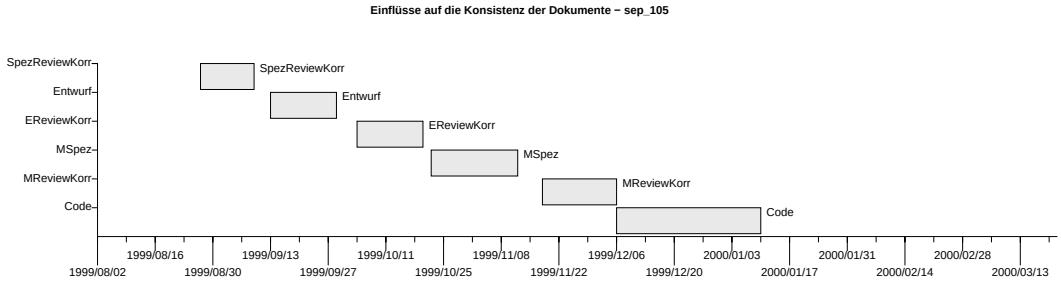



*Abb. 4.9:* Einfluss Konsistenz (sep_105)

dargestellt. Eine genauere Aufschlüsselung der Aktivitäten der einzelnen Phasen kann dem jeweiligen Phasendiagramm entnommen werden. Das letzte Diagramm in diesem Abschnitt ist die „Aufwandsverteilung“. Sie gibt einen Überblick über die Verteilung des Aufwands über die einzelnen Projektphasen.
Die Projektverlaufsdiagramme sind alle auf die gleiche Art und Weise aufgebaut. Auf der y-Achse werden die Phasen und die Subphasen (Review, Korrektur) dargestellt, die im jeweiligen Diagramm näher untersucht werden. Die x-Achse hingegen repräsentiert die Projektzeit.

*Einfluss Konsistenz*

Dieses Diagramm zeigt den genaueren Projektverlauf. Anhand der Überschneidungen von Reviews und anschließenden Phasen kann ermittelt werden, ob der Spieler zu früh mit der nächsten Phase begonnen hat, und somit Fehler aus der früheren Phase in die nächste Phase übernommen hat. Ein Alleskorrektureintrag weist darauf hin, dass der Spieler nachträglich versucht hat, seine Fehler doch noch zu beseitigen und die Dokumente dafür einer zusätzlichen Korrektur unterzogen hat.

Beim Verlauf der Gruppe sep_105, in Abbildung 4.9, ist aufgefallen, dass sie sich genau an das Wasserfallmodell gehalten hat, das heißt die nächste Phase wurde erst begonnen, nachdem die Vorgängerphase bereits vollständig abgeschlossen war. Dies hat sich positiv auf die Vollständigkeit der Dokumente ausgewirkt. Außerdem war das Durchführen von Alleskorrekturen nicht notwendig, da es keine Überschneidungen von Phasen gegeben hat.

So streng können die Phasen aber in der Praxis nicht voneinander getrennt

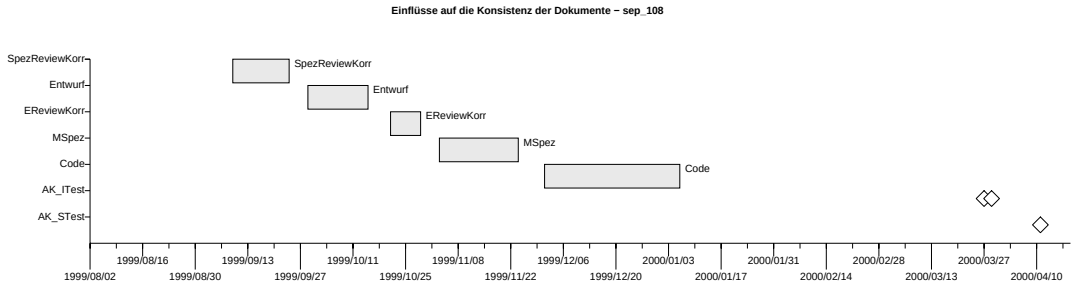



*Abb. 4.10:* Einfluss Konsistenz (sep_108)

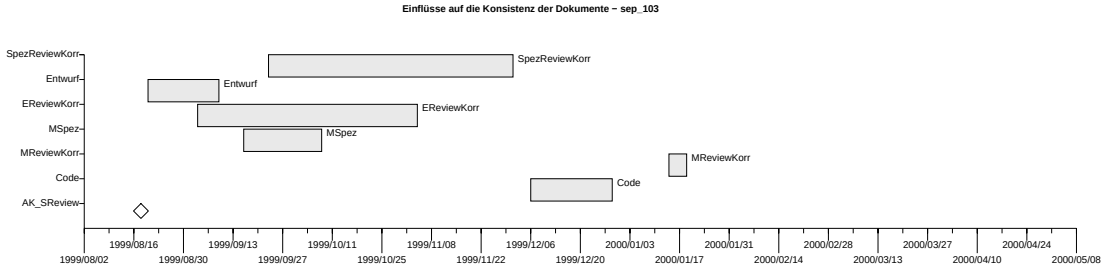



*Abb. 4.11:* Einfluss Konsistenz (sep_103)

werden. Wahrscheinlicher ist, dass Teile von Phasen parallel durchgeführt werden und die Dokumente einer Alleskorrektur unterzogen werden müssen.

Auch Gruppe sep_108 hat sich, wie die Abbildung 4.10 zeigt, an das klassische Wasserfallmodell gehalten. Allerdings fällt bei diesem Projektverlauf auf, dass die Gruppe auf die Korrektur der Modulspezifikation vergessen hat. Dadurch wurden viele Fehler in den Code übernommen, die erst später durch intensive Tests lokalisiert und eliminiert werden konnten. Der hohe Modultestaufwand dieser Gruppe, der aus den Projektdaten hervorgeht, spiegelt diesen Fehler wider. Die Rauten im Diagramm markieren den Zeitpunkt zu dem die Alleskorrektur des Integrations- bzw. des Systemtests durchgeführt wurde. Die Gruppe sep_108 hat also versucht alle Dokumente aufgrund der in den Tests gefundenen Fehlern zu verbessern.

Der Projektverlauf der Gruppe sep_103 in Abbildung 4.11 zeigt ein ähnliches Bild. Sie hat zwar nicht auf die Korrektur der Modulspezifikation vergessen, aber sie hat die Codephase bereits abgeschlossen, bevor sie mit der

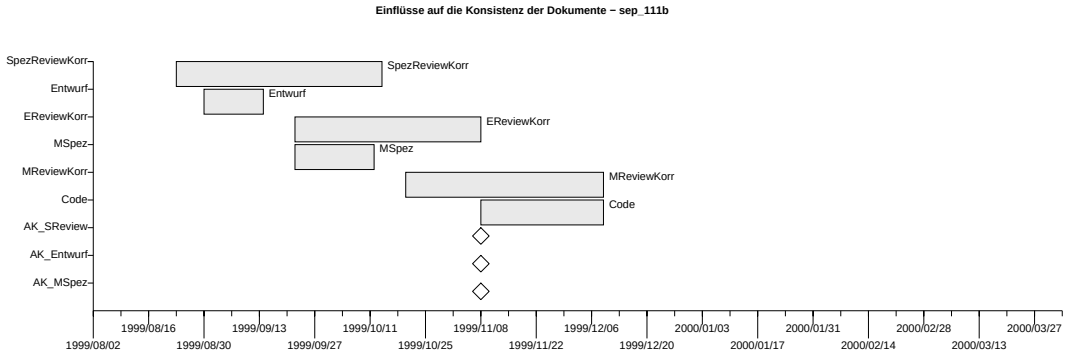



*Abb. 4.12:* Einfluss Konsistenz (sep_111)

Korrektur der Modulspezifikation begonnen hat. Auch in diesem Fall sind alle Fehler, die im Modulspezifikation-Review gefunden wurden, in den Code übernommen worden. Das anschließende Korrigieren der Modulspezifikation hat zwar zu einem bessern Modulspezifikationsdokument geführt, die in den Code übernommenen Fehler wurden aber dadurch nicht ausgemerzt. Diese Fehler sind nur durch intensives Testen oder durch die Durchführung einer Alleskorrektur wieder gut zu machen. Die Gruppe sep_103 hat sich für das erstere entschieden. Eine genaue Analyse des Testaufwands werden wir später, im Abschnitt 4.3.3 („Test und Korrektur (Code)“), vornehmen.

Gruppe sep_111 hingegen, deren Verlauf in Abbildung 4.12 gezeigt wird, hat immer zu früh mit der nächsten Phase begonnen. Wie man im Projektverlauf sehen kann, wurde die Entwurfsphase beendet bevor die Korrektur der Spezifikation erfolgreich beendet werden konnte. Das parallele Beginnen von Entwurfsreview-Korrektur und Modulspezifikationsphase führte dazu, dass nur einige Fehler im Vorgabedokument verbessert werden konnten, bevor es für die Modulspezifikation herangezogen wurden. Dadurch wurden viele Fehler, die im Entwurf zwar gefunden, aber noch nicht verbessert werden konnten, in die Modulspezifikation übernommen.
Allerdings hat diese Gruppe die Alleskorrektur von Entwurf und Modulspezifikation vorgenommen, um noch möglichst viele der übernommenen Fehler ausmerzen zu können. Der Zusatzaufwand, der dadurch entstanden ist, hätte jedoch vermieden werden können, wenn die Gruppe erst nach Beendigung der Korrektur des Vorgabedokuments mit der nächsten Phase begonnen hätte. Die Fehler, die in den Code übernommen wurden, wurden nicht mehr verbessert, da die Codephase aber nicht gleichzeitig mit der Korrektur der

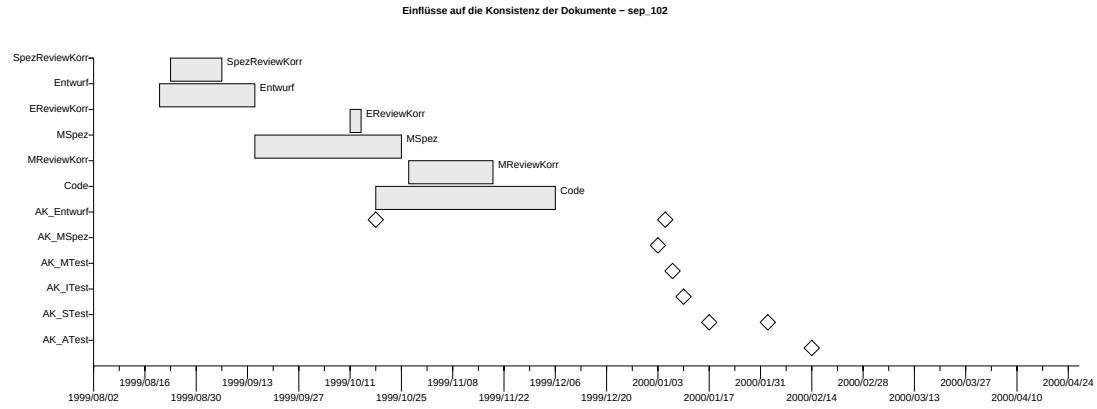



*Abb. 4.13:* Einfluss Konsistenz (sep_102)

Modulspezifikation begonnen hat, konnte das Übernehmen einiger Fehler in den Code verhindert werden. Ein paar der übernommenen Fehler konnten in der anschließenden Testphase aufgedeckt und korrigiert werden.

Bei der Gruppe sep_102 ist in Abbildung 4.13 genau zu sehen, dass jede neue Phase bereits während der Korrektur der alten begonnen wurde. In diesem Fall wurden jedoch alle Dokumente auch dementsprechend nochmals durch eine Alleskorrektur korrigiert.

*Spezifikation und Entwurf*

Das Diagramm zeigt die Phasen Spezifikation und Grobentwurf. Dargestellt werden jeweils die Dauer der gesamten Phase und zusätzlich die Tätigkeiten für die einzelnen Mitarbeiter. Review und Korrektur der Spezifikation und des Systemdesigns (Grobentwurf) werden ebenso gezeigt.

Betrachten wir nun die frühen Phasen des Projektverlaufs (Abbildung 4.14) der Gruppe sep_105 genauer. Die grauen Balken stellen die Projektphasen bzw. -subphasen dar, während die weißen Balken, die einzelnen Aktivitäten innerhalb der Phase zeigen und angeben welcher Mitarbeiter die Aktivität ausgeführt hat.
Das Projektverlaufsdiagramm zeigt uns, dass die Gruppe sep_105 für die Spezifikation den qualifiziertesten Mitarbeiter (Richard) eingesetzt hat. Der anschließende Review wurde richtigerweise von unabhängigen Gutachtern

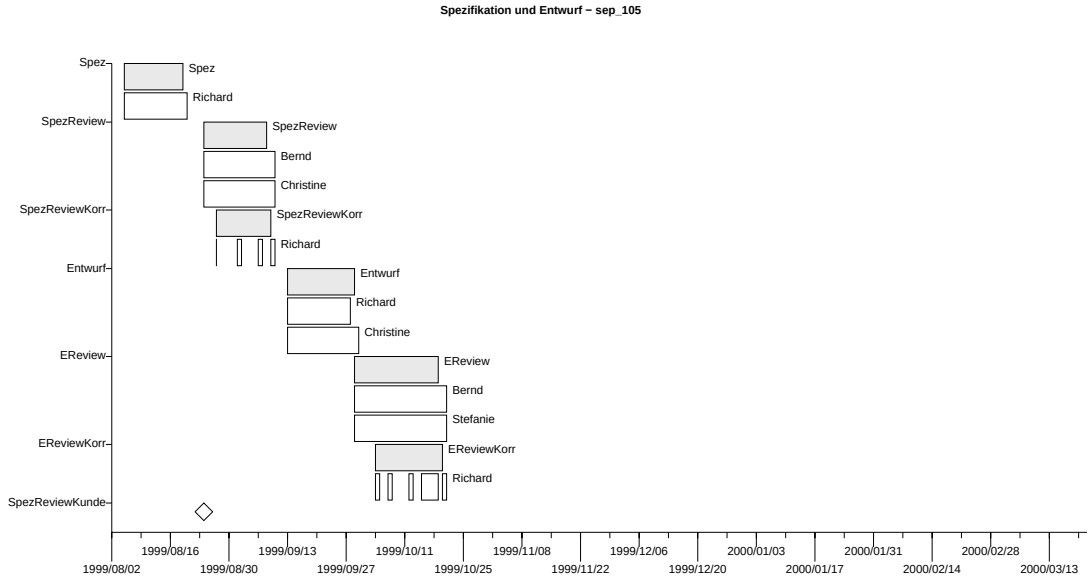



*Abb. 4.14:* Spezifikation und Entwurf (sep_105)

und zusammen mit dem Kunden vorgenommen, dies können wir anhand der Raute im Diagramm erkennen. Während des Reviews wurden bereits die identifizierten Fehler vom Autor der Spezifikation verbessert.

Auch für die Erstellung des Grobentwurfs wurden fähige Entwickler (Richard und Christine) eingesetzt. Die Korrektur des Entwurfs wurde allerdings nur von Richard durchgeführt. Es wäre vorteilhafter gewesen, zusätzlich zu Richard auch Christine mit der Korrektur des Entwurfs zu beauftragen. Sie hätte die Fehler, die ihren Teil des Dokuments betreffen, schneller und korrekter verbessern können als Richard.

Die Spezifikation wurde im Falle der Gruppe sep_103 (Abbildung 4.15) von Stefanie und Richard durchgeführt. Stefanie ist aber besser für das Prüfen von Dokumenten geeignet. Ihre Mitarbeit an der Spezifikation hat zu zusätzlichen Fehlern im Dokument geführt. Betrachten wir die Phase Spezifikation-Review genauer, so können wir feststellen, dass die Gruppe fünf Reviews durchgeführt hat, von denen nur drei erfolgreich waren. Der erste Review wurde von Bernd und Thomas vorgenommen. Die zwei kurzen Balken danach geben an, dass der Projektleiter versucht hat dieselben Mitarbeiter für einen weiteren Review einzusetzen. Da aber die Korrektur der ersten Prüfung noch nicht

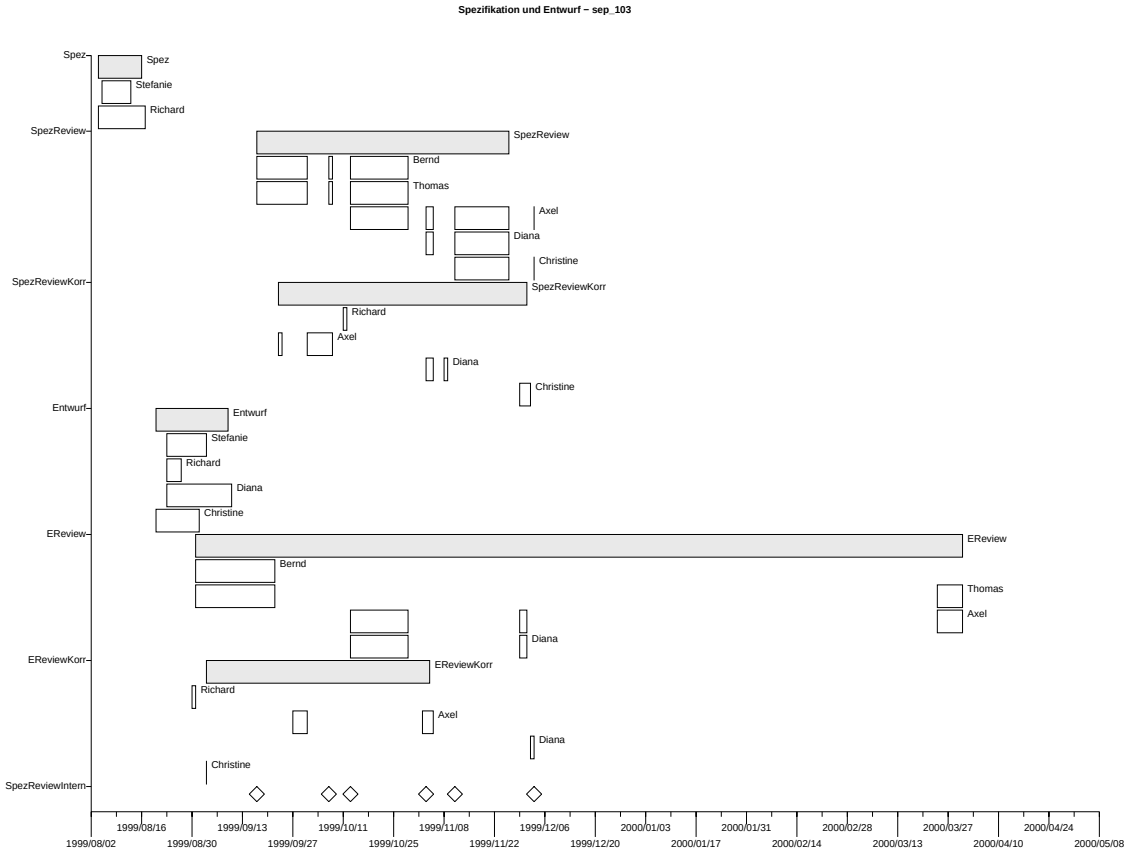



*Abb. 4.15:* Spezifikation und Entwurf (sep_103)

beendet war, konnte der Review nicht beginnen. Der nächste Versuch war erfolgreich, da die Korrektur des vorherigen Reviews bereits abgeschlossen war. Der vierte Review war wieder nicht erfolgreich. Aufgrund des Wochenendes haben die Entwickler aber erst etwas später gemeldet, dass der Review bereits beendet wurde. Auch in diesem Fall war die Korrektur des vorherigen Reviews noch nicht beendet worden. Die letzte Prüfung wurde von Axel, Diana und Christine erfolgreich durchgeführt und im Anschluss von Christine korrigiert.
Wie ebenfalls aus dem Diagramm hervorgeht, wurde der Kunde zu keinem der Reviews eingeladen. Die Rauten zeigen an, dass alle Reviews intern, also ohne den Kunden, durchgeführt wurden. Der Einsatz von Axel, Diana und Christine in der Korrekturphase hat zu neuen Fehlern im Dokument geführt, da es sich bei diesen Mitarbeitern nicht um die Autoren des Dokuments handelt.

Für die Erstellung des Entwurfs wurden vier Mitarbeiter eingesetzt. Dadurch stieg der Kommunikationsaufwand enorm an. Der Balken für die Entwurfsreviewphase ist bei dieser Gruppe extrem lang. Es waren jedoch nur zwei Reviews erfolgreich. Der erste Review wurde von Bernd und Thomas, der zweite von Axel und Diana vorgenommen. Korrigiert wurden die Fehler des ersten Reviews von Richard und Axel, während die zweite Korrektur nur mehr von Axel durchgeführt wurde. Ende November 1999 und Ende März 2000 wollte die Gruppe einen weiteren Review des Entwurfsdokuments vornehmen. Diese Reviews konnten jedoch nicht erfolgreich durchgeführt werden, da die Korrektur des zweiten Reviews vorzeitig abgebrochen wurde. Dies kann allerdings nur mit Hilfe der Tabelle „Gefundene Fehler in Prüf- und Korrekturmaßnahmen“, die im Abschnitt 4.3.4 beschrieben wird, gezeigt werden, da es dort einen Eintrag gibt, der anzeigt, ob die Korrektur vollständig durchgeführt oder abgebrochen wurde.

Für Reviews gilt demnach die folgende Faustregel: Ein weiterer Review kann erst dann durchgeführt werden, wenn die Korrektur des letzten Reviews erfolgreich beendet wurde.

Auch Gruppe sep_111 hat, wie aus Abbildung 4.16 hervorgeht, Richard für die Spezifikation eingesetzt. Es wurden von dieser Gruppe drei Spezifikationsreviews durchgeführt, wobei an zwei Reviews auch der Kunde teilnahm. Dadurch konnte, wie wir anhand der „Vollständigkeit der Dokumente“ in Tabelle 4.13 noch sehen werden, diese Gruppe das vollständigste Spezifikationsdokument anfertigen. Auch der Entwurf wurde von Richard erstellt, jedoch bereits bevor die Korrektur der Spezifikation beendet wurde. Die beiden anschließenden Reviews wurden von unabhängigen Gutachtern durchgeführt und anschließend vom Autor und einigen anderen Mitarbeitern vorgenommen, was sich nachteilig auf die Korrektheit des Dokuments ausgewirkt hat.

Gruppe sep_102 hat, wie die Abbildung 4.17 zeigt, die Spezifkation von Thomas und Richard erstellen lassen. Der anschließende Review wurde von Stefanie und Christine vorgenommen. Die im Review gefundenen Fehler wurden jedoch nur von Richard verbessert. Es wäre in diesem Fall besser gewesen, wenn auch Thomas für die Korrektur eingesetzt worden wäre, da dann jeder Autor seinen Teil des Dokuments verbessern hätte können. Auch in der Entwurfsphase ist die Gruppe auf diese Weise vorgegangen.

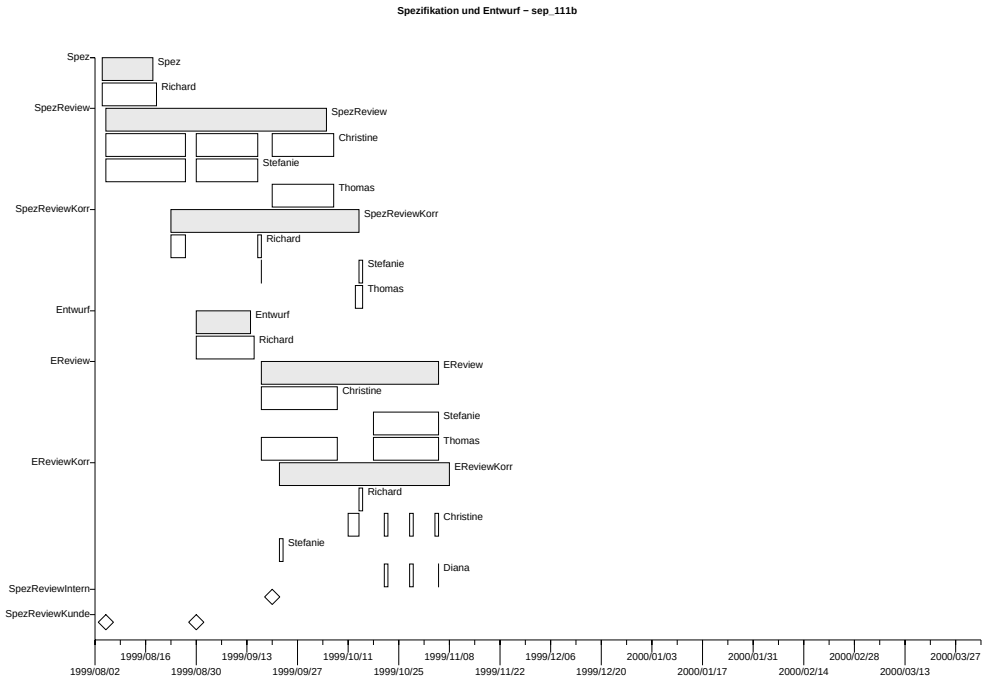



*Abb. 4.16:* Spezifikation und Entwurf (sep_111)

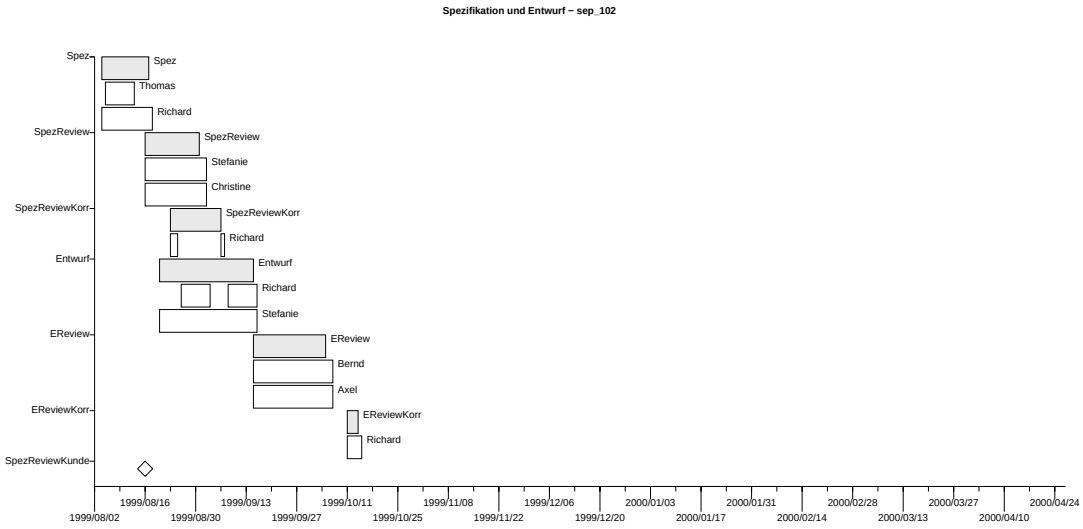



*Abb. 4.17:* Spezifikation und Entwurf (sep_102)


*Abb. 4.18:* Modulspezifikation und Codierung (sep_102)

*Modulspezifikation und Codierung*

In diesem Diagramm werden Feinentwurf und Codierung dargestellt, jeweils für die gesamte Phase und aufgeteilt nach beteiligten Entwicklern. Modulspezifikation-Review und Codereview werden jeweils mit Korrektur gezeigt. Mittels dieses Diagramms kann der Projektverlauf in diesem Abschnitt genauer untersucht werden.

Im Falle der Gruppe sep_102 ist aus Abbildung 4.18 erkennbar, dass für den Modulspezifikation-Review unabhängige Gutachter eingesetzt wurden. Auch bei der Korrektur ist die Gruppe richtig vorgegangen und hat die Autoren der Modulspezifikation die gefundenen Fehler korrigieren lassen. Ein leichtes Überschneiden der Modulspezifikations- und der Codierungsphase kann festgestellt werden. Betrachtet man jedoch die Korrekturphase für die Modulspezifikation genauer, so kann man erkennen, dass das Dokument bereits vor Beginn der Korrekturphase als Vorgabedokument für den Code eingesetzt wurde. Dadurch wurden viele Fehler in den Code übernommen.

Am Projektverlauf von Gruppe sep_103 in Abbildung 4.19 fällt auf, dass der Modulspezifikation-Review und die damit verbundene Korrektur erst nach der Fertigstellung des Codes durchgeführt wurde. Die in dem Modulspezifikation-Review gefundenen Fehler wurden dadurch in den Code übernommen und konnten dort erst mittels Tests entdeckt und verbessert werden.

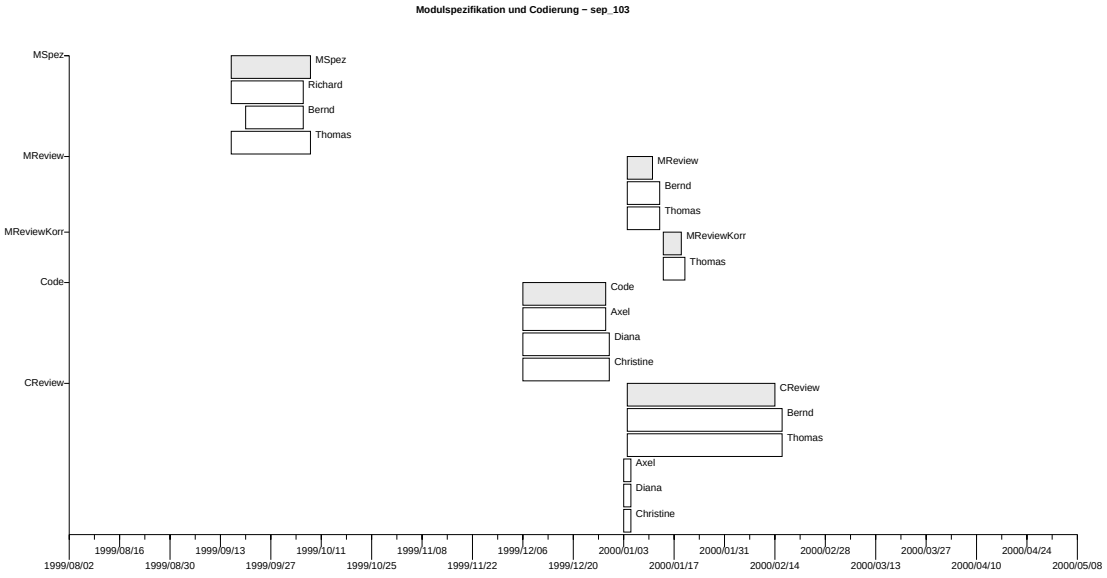



*Abb. 4.19:* Modulspezifikation und Codierung (sep_103)

Wie aus dem Diagramm „Einfluss Konsistenz“ (Abbildung 4.11) ersichtlich war, wurde der Code nicht nachträglich aufgrund der im Review gefundenen Fehler korrigiert. Hätte man diese Korrektur vorgenommen, hätte man einen vollständigeren und korrekteren Code für die Testphase zur Verfügung gehabt. Betrachten wir den Codereview genauer, so kann festgestellt werden, dass die Gruppe sep_103 Axel, Diana und Christine in einem ersten Review einsetzten wollten. Da es sich bei diesen drei Mitarbeitern aber um die Autoren des Codes handelt, wurden keine Fehler gefunden und der Review nach zwei Tagen wieder beendet.

Gruppe sep_105 hat sich, wie bereits berichtet, eisern an das Wasserfallmodell gehalten. Das Diagramm in Abbildung 4.20 zeigt jedoch, dass die Korrektur der Modulspezifikation parallel zum Review durchgeführt wurde. Das Projektverlaufsdiagramm zeigt weiters, dass die Gruppe sep_105 qualifizierte Mitarbeiter sowohl für die Erstellung der Modulspezifikation als auch für die Implementierung eingesetzt hat.

Zuletzt wollen wir noch den Projektverlauf von Gruppe sep_111 in Abbildung 4.21 näher betrachten. Es wurden zwei Modulspezifikation-Reviews durchgeführt, die Korrektur der gefundenen Fehler wurde jedoch in beiden Fällen nicht von den Autoren vorgenommen. Da sich die Codephase

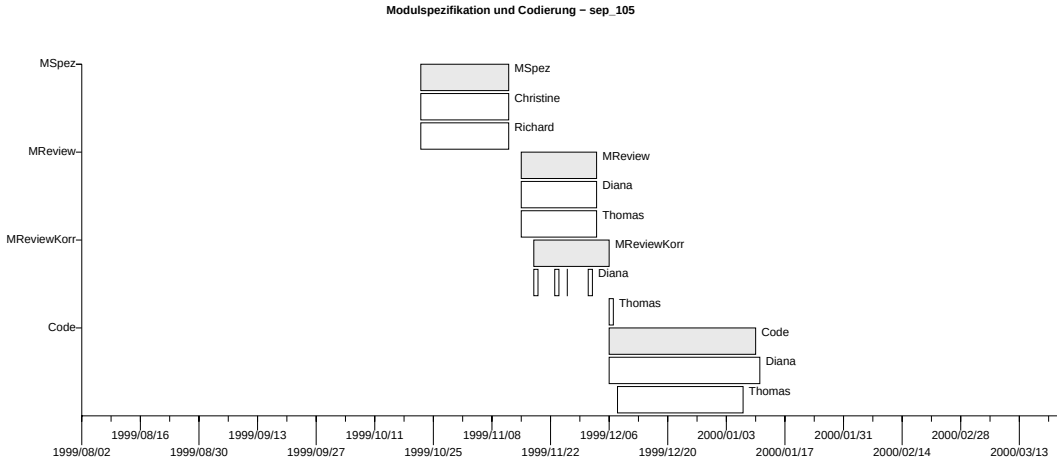



*Abb. 4.20:* Modulspezifikation und Codierung (sep_105)

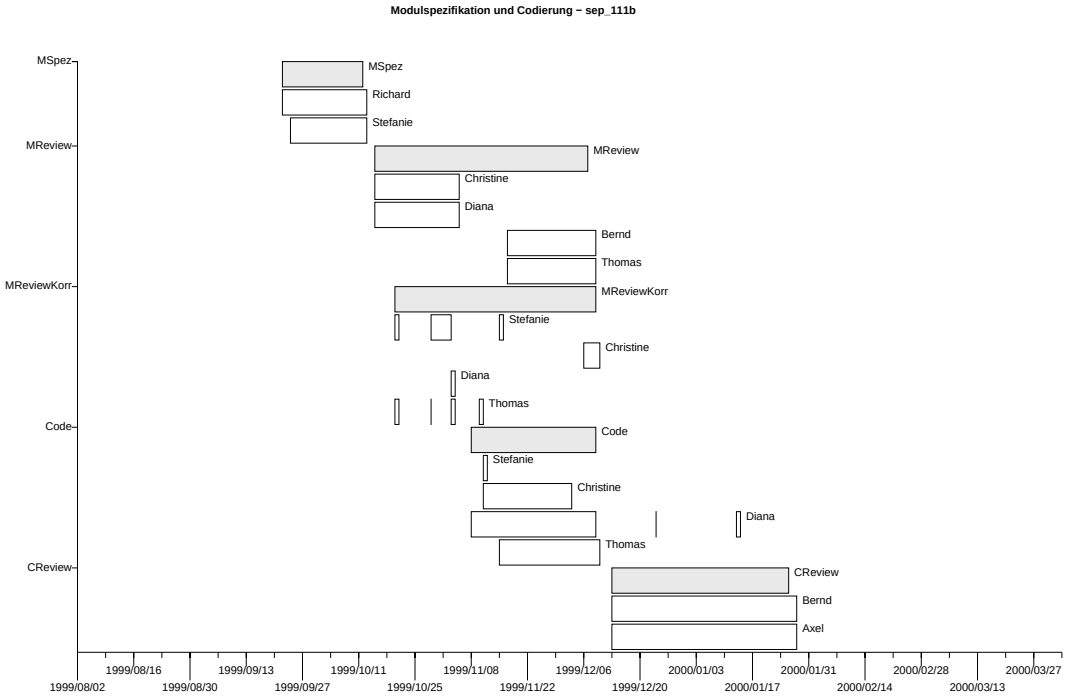



*Abb. 4.21:* Modulspezifikation und Codierung (sep_111)

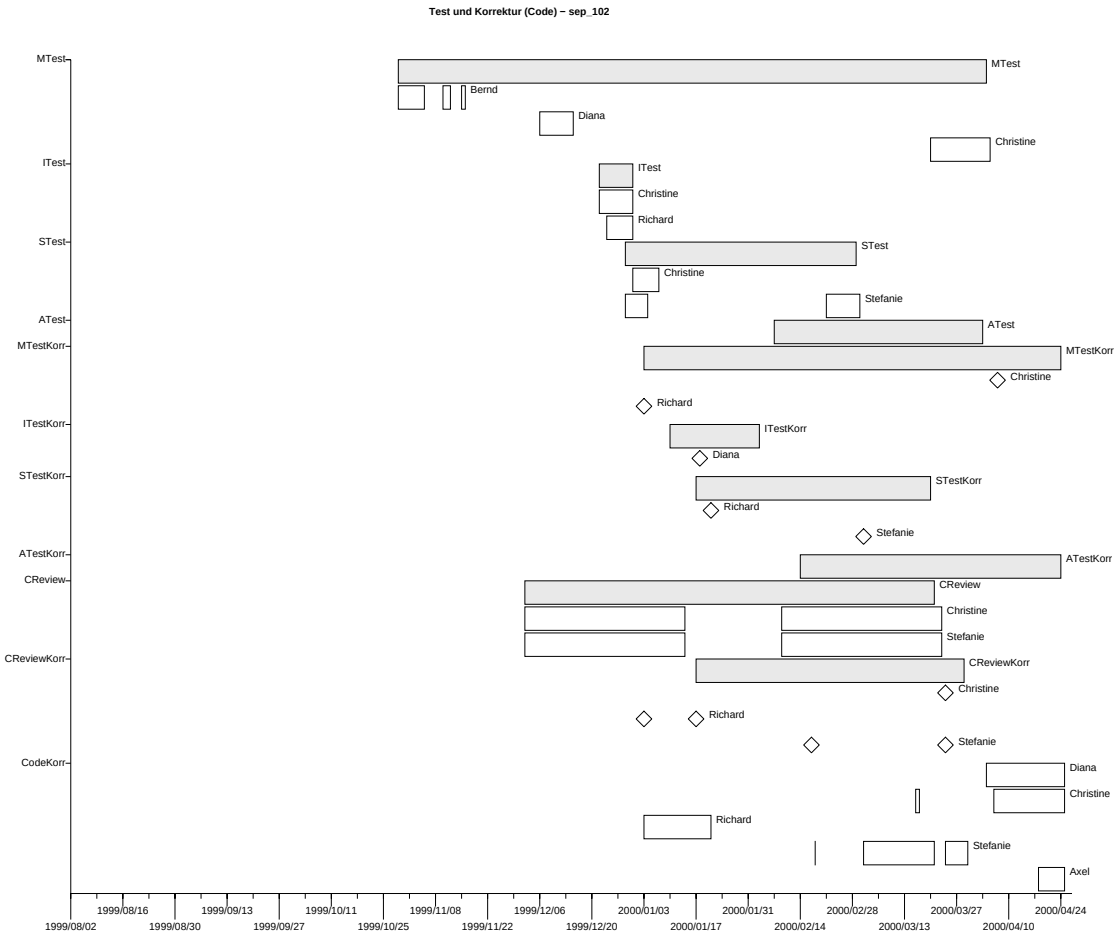



*Abb. 4.22:* Test und Korrektur Code (sep_102)

mit der Korrekturphase der Modulspezifikation überschneidet, wurden einige der Fehler in den Code übernommen. Für den Codereview hat die Gruppe sep_111 vernünftigerweise unabhängige Gutachter eingesetzt.

### *Test und Korrektur (Code)*

Das Diagramm zeigt alle Tests des Codes (Modultest, Integrationstest, Systemtest, Abnahmetest). Die Tätigkeiten werden jeweils für die gesamte Phase und aufgeteilt nach Entwicklern dargestellt. Mit Hilfe dieses Diagramms kann der Projektverlauf in diesem Abschnitt genauer untersucht werden.

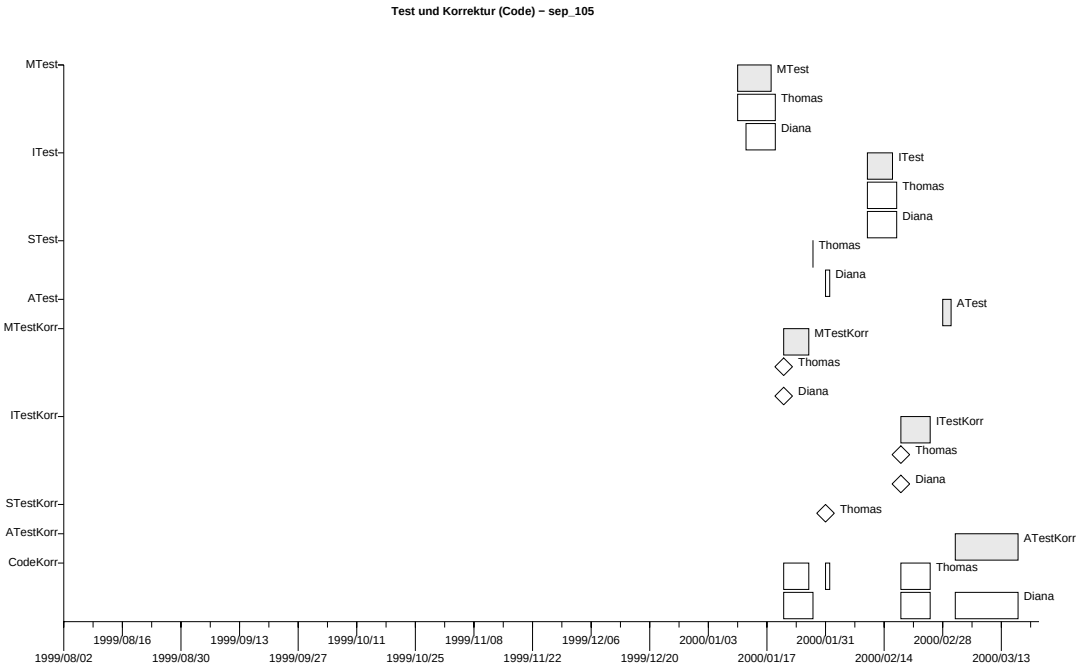



*Abb. 4.23:* Test und Korrektur Code (sep_105)

Der Projektverlauf der Gruppe sep_102 in Abbildung 4.22 zeigt, dass zu früh mit dem Modultest begonnen wurde. Außerdem überschneidet sich die Korrektur des Modultest mit der Korrektur der gefundenen Fehler im Codereview. Man findet in diesem Fall viele gleiche Fehler, das führt zu einem unnötigen Zusatzaufwand.
Weiters wurden der Integrationstest und der Systemtest bereits beendet bevor die Module vollständig getestet waren. Dadurch wurde die Durchführung eines zweiten Systemtests und eines zweiten Abnahmetests notwendig. Dies kann mit Hilfe der Tabelle „Gefundene Fehler in Test- und Korrekturmaßnahmen“, die in Abschnitt 4.3.4 gezeigt wird, festgestellt werden.

Gruppe sep_105 hingegen hat auch die Tests schrittweise durchgeführt. Wie man im Projektverlauf in Abbildung 4.23 erkennen kann, wurde nach jedem Test auf die Fertigstellung der Korrektur gewartet bis mit dem nächsten Test begonnen wurde. Diese Gruppe hat nur jeweils einen Test durchgeführt und dabei auch die qualifiziertesten Mitarbeiter eingesetzt. Allerdings kann anhand des Diagramms auch festgehalten werden, dass diese Gruppe keinen Systemtest veranlasst hat.

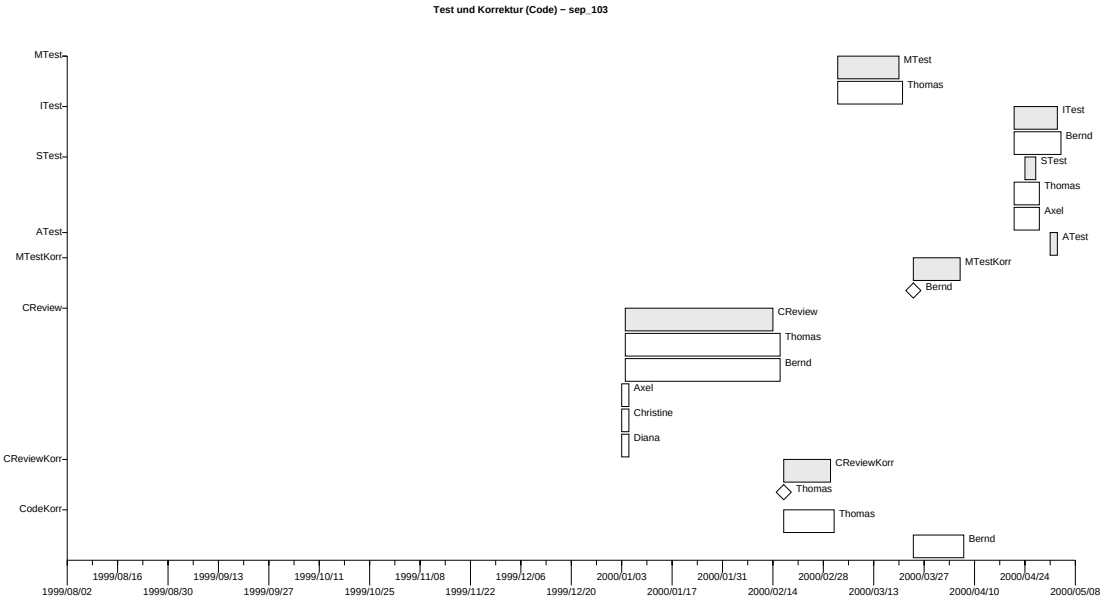



*Abb. 4.24:* Test und Korrektur Code (sep_103)

Bei nochmaliger Betrachtung der Zielvorgaben in Tabelle 4.1 kann festgestellt werden, dass die Zielvorgabe „Fehler/KLOC“ von der Gruppe sep_105 nicht erreicht wurden. Die Durchführung eines weiteren Tests oder des fehlenden Systemtests hätte zu einer Minimierung der „Fehler/KLOC“ geführt. Es hätte aber auch das Gegenteil eintreten können: wenn der qualifizierteste Mitarbeiter zu diesem Zeitpunkt nicht mehr für das Projekt verfügbar gewesen wäre und der Spieler einen anderen (schlechter qualifizierteren) Mitarbeiter für diesen Test einsetzen hätte müssen. In diesem Fall wäre es zu einer Verschlechterung dieses und anderer Werte gekommen.

Betrachten wir nun den Projektverlauf von Gruppe sep_103 in Abbildung 4.24. Hier fällt auf, dass die Gruppe zuerst einen Codereview veranlasst hat, bevor die Tests durchgeführt wurden. Die dabei gefunden Fehler wurden verbessert, bevor mit dem Modultest begonnen wurde. Alle weiteren Tests wurden zwar noch durchgeführt, allerdings kam es nicht mehr zu einer Korrektur der gefunden Fehler. Man würde zuerst behaupten, dass auf die Korrektur vergessen wurde, bei genauerer Analyse des verbrauchten Projektbudgets von 607 640 DM kann aber sofort erkannt werden, dass diese Gruppe zu diesem Zeitpunkt bereits weit über der Grenze von 450 000 DM lag, und sich da-

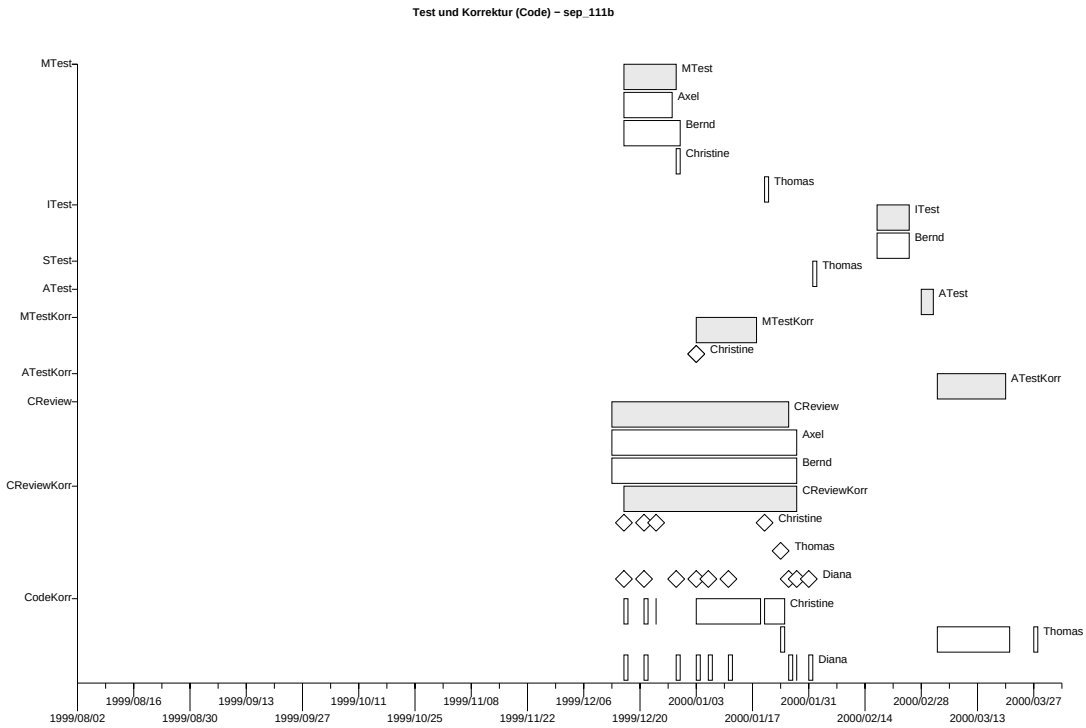



*Abb. 4.25:* Test und Korrektur Code (sep_111)

durch eine Korrektur des Codes nicht mehr leisten konnte.

Im Diagramm 4.25 der Gruppe sep_111 fällt auf, dass der Modultest und der Codereview parallel durchgeführt wurden. Das hat keine negativen Auswirkungen, da im Codereview andere Fehler gefunden werden, als beim Testen von Modulen. Die Gruppe hat außerdem einen Integrationstest durchgeführt, der jedoch nicht mehr korrigiert wurde. Auf einen Systemtest wurde verzichtet. Betrachtet man nochmals die Zielvorgaben (Tabelle 4.1), dann kann festgestellt werden, dass die Gruppe möglicherweise bereits zu diesem Zeitpunkt im Projekt die Kostengrenze von 450 000 DM erreicht hatte. Um so wenig zusätzliche Kosten wie möglich zu verursachen, wurde nur noch ein Abnahmetest veranlasst und die Fehler, die der Kunde dabei gefunden hat, verbessert.

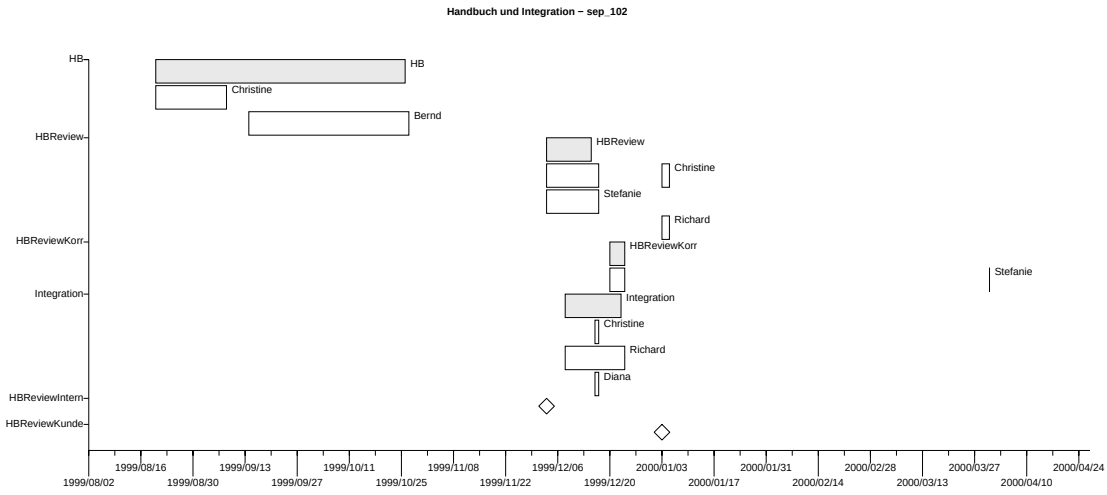



*Abb. 4.26:* Handbuch und Integration (sep_102)

*Handbuch und Integration*

In diesem Diagramm werden die Phasen Integration und Erstellung des Handbuchs gezeigt, für die gesamte Phase und aufgeteilt nach Entwicklern. Für das Handbuch wird zusätzlich Review und Korrektur dargestellt. Mittels dieses Diagramms kann der Projektverlauf in diesem Abschnitt genauer untersucht und Besonderheiten entdeckt werden, die anhand der folgenden Beispiele aufgezeigt werden.

Das Vorgehen der Gruppe sep_102 wird in Abbildung 4.26 gezeigt. Wie man aus dem Projektverlauf erkennen kann, wurde das Handbuch gleich nach Abschluss der Spezifikationsphase erstellt. Es wurden zwar zwei Reviews durchgeführt, aber nur der erste interne Review konnte auch erfolgreich beendet werden. Es wäre sehr wichtig gewesen, den Kunden gleich zum ersten Review einzuladen, da er noch viele Analysefehler bzw. fehlende Funktionalität entdecken kann, die den Gutachtern nicht auffallen. Diese fehlende Funktionalität kann zu einem frühen Zeitpunkt noch recht günstig in das Projekt integriert werden.

Mit dem Handbuch hat die Gruppe sep_103 viel zu spät begonnen, das geht aus dem Projektverlauf in Abbildung 4.27 hervor. Sie hat auch den Kunden nicht zu dem Handbuchreview eingeladen. Die wesentlichen Analysefehler, die der Kunde gefunden hätte, und die für die Erfüllung der gewünschten Funk-

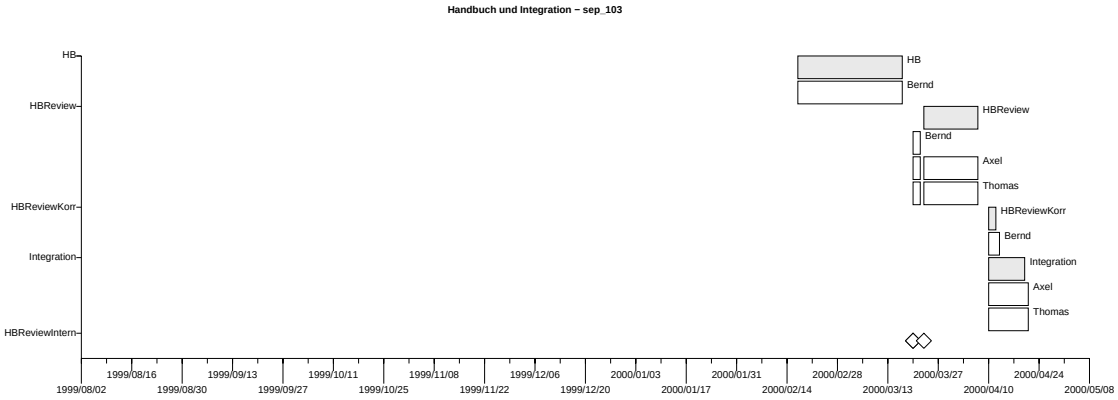



*Abb. 4.27:* Handbuch und Integration (sep_103)

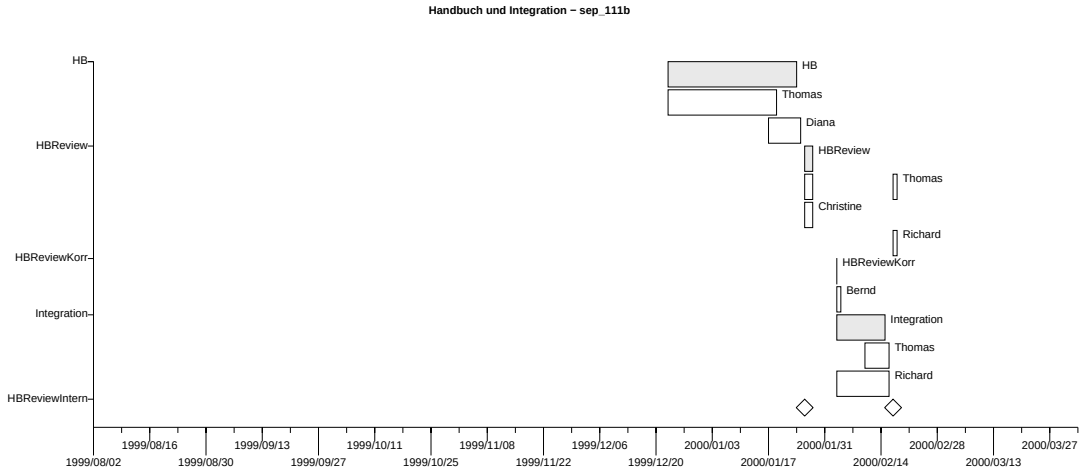



*Abb. 4.28:* Handbuch und Integration (sep_111)

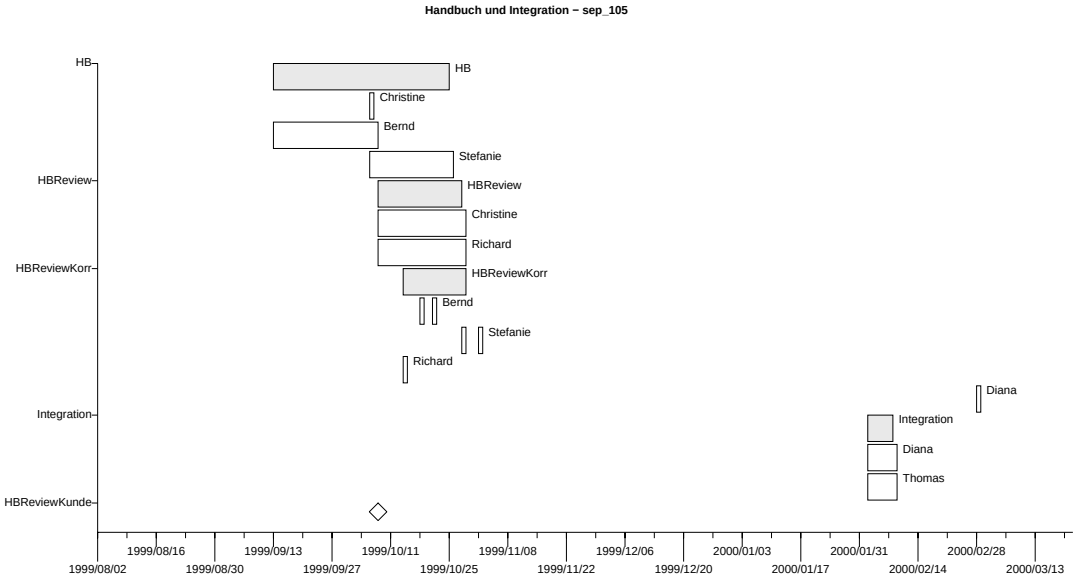



*Abb. 4.29:* Handbuch und Integration (sep_105)

tionalität notwendig gewesen wären, wurden nicht vom internen Reviewteam entdeckt. Betrachtet man die „Restfehler in den Dokumenten“ (Tabelle 4.12) kann man erkennen, dass das Handbuch der Gruppe sep_103 nach der Beendigung des Projekts noch 56 Analysefehler aufwies. Viele dieser Fehler hätten durch einen Review mit dem Kunden aufgedeckt werden können.

Die Gruppe sep_111 hingegen hat, wie in Abbildung 4.28 ersichtlich, zwar mit einem Handbuchreview begonnen, diesen aber abgebrochen und somit keine Prüfung des Handbuchs, weder intern noch mit dem Kunden durchgeführt.

Der Projektverlauf der Gruppe sep_105 in Abbildung 4.29 zeigt wieder einen besseren Verlauf. Die Erstellung des Handbuchs wurde recht früh begonnen. Durch die Teilnahme des Kunden an einem Handbuchreview wurden Analysefehler bzw. fehlende Funktionalität entdeckt, die somit noch in das Projekt einfließen konnten.

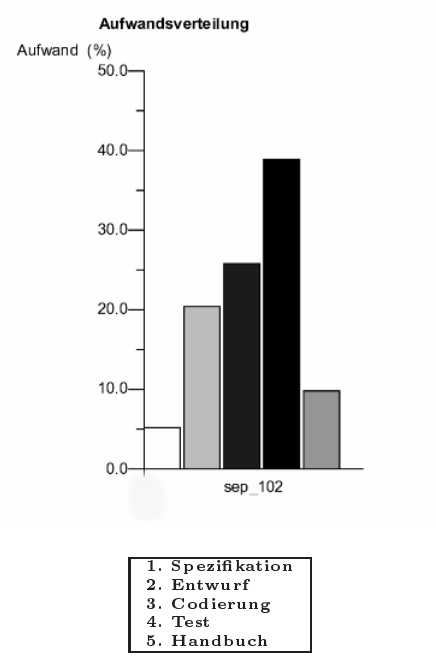



*Abb. 4.30:* Aufwandsverteilung (sep_102)

*Aufwandsverteilung über alle Phasen*

Dieses Diagramm zeigt die Aufwandsverteilung in Prozent für alle Phasen. Die einzelnen Balken entsprechen den Phasen Spezifikation, Entwurf, Codierung, Test und Handbucherstellung. Mit Hilfe dieses Diagramms kann festgestellt werden, ob der Spieler in die einzelnen Phasen zu viel oder zu wenig Aufwand investiert hat.

Das Aufwandsverteilungsdiagramm von Gruppe sep_102 in Abbildung 4.30 zeigt, dass der Aufwand für die Spezifikationsphase recht gering ausgefallen ist. Bei genauerer Betrachtung der Projektdaten erkennt man, dass das Spezifikationsdokument nach Beendigung des Projekts aber nur wenige Restfehler enthielt. Da der Aufwand für diese Phase aber trotzdem so gering ist, kann darauf geschlossen werden, dass diese Gruppe die richtigen Mitarbeiter als Autoren bzw. Gutachter eingesetzt hat und auch die Korrektur des Spezifikationsdokuments von dem/den Autor(en) durchgeführt wurde. Wie wir bereits in Abbildung 4.17 gesehen haben, war dies der Fall.
Die wenigen Restfehler des Dokuments weisen zusätzlich darauf hin, dass zu dem Spezifikationsreview auch der Kunde eingeladen wurde. Auch das haben wir bei der Betrachtung der Abbildung 4.17 festgestellt.

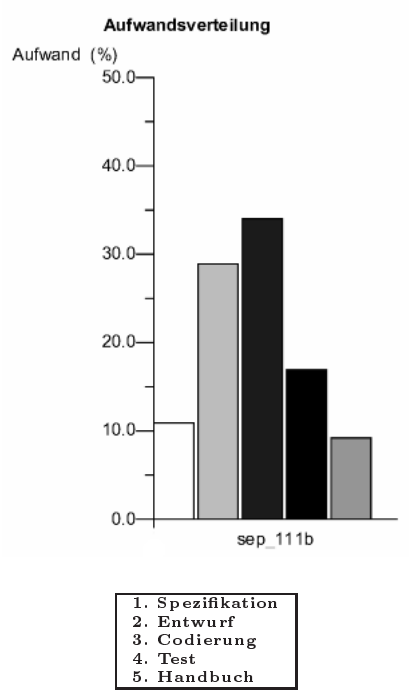



*Abb. 4.31:* Aufwandsverteilung (sep_111)

Das Aufwandsverteilungsdiagramm in Abbildung 4.30 zeigt aber auch, dass der Aufwand für die Testphase sehr hoch ist. Dies deutet auf viele Fehler im Code bzw. in den Entwurfsdokumenten hin, die durch die Tests erst gefunden und eliminiert werden mussten. In Abschnitt 4.3.4 werden wir sehen, dass die Gruppe sep_102 tatsächlich intensiv getestet hat.

Bei der Gruppe sep_111, deren Aufwandsverteilung in Abbildung 4.31 zu sehen ist, fallen die Phasen Spezifikation, Entwurf und Codierung stärker aus. Betrachtet man die einzelnen Phasen genauer, so kann man den Grund dafür recht schnell entdecken. Diese Gruppe hat, wie wir in Abbildung 4.16 gesehen haben, das Spezifikationsdokument drei Reviews unterzogen. Das ist auch der Grund warum diese Gruppe, wie wir später (Tabelle 4.13) noch sehen werden, das vollständigste (200,38 AFPs) Spezifikationsdokument erstellt hat. Aus den Projektverlaufsdiagrammen (Abbildung 4.16, 4.21, 4.25) geht hervor, dass die Dokumente Grobentwurf, Feinentwurf und Codierung jeweils zwei Reviews unterzogen wurden. Bei den Tests wurden nur Modultest, Integrationstest und Abnahmetest durchgeführt, wobei nur die Fehler aus dem Modultest korrigiert wurden. Darum fällt der Aufwand für diese

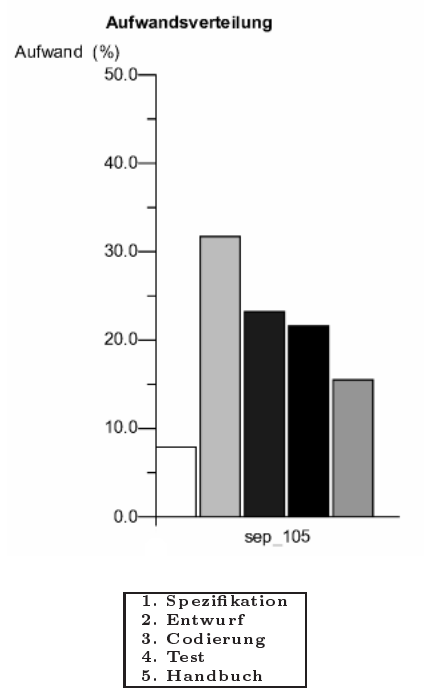



*Abb. 4.32:* Aufwandsverteilung (sep_105)

Phase geringer aus.

Die Aufwandsverteilung der Gruppe sep_105 in Abbildung 4.32 zeigt einen anderen Verlauf. Der Aufwand wurde vor allem in die ersten Phasen gesteckt, das heißt vor allem in die Spezifikation und den Entwurf, aber auch in das Handbuch, das auch dazu beitragen kann, dass möglichst viele Analysefehler zu Beginn des Projekts aufgedeckt und korrigiert werden können. Die Testphase ist geringer ausgefallen. Wenn wir das Projektverlaufsdiagramm „Test und Korrektur (Code)“ in Abbildung 4.23 nochmals betrachten, können wir erkennen, dass die Gruppe nur die Durchführung eines Modultests, eines Integrationstests und eines Abnahmetests veranlasst hat. Auf einen Systemtest wurde vollständig verzichtet.

### *4.3.4 Qualitätssicherung*

Die Tabellen bzw. Diagramme in diesem Abschnitt geben Auskunft darüber, welche Qualitätssicherungsmaßnahmen der Spieler eingesetzt hat, und wie erfolgreich diese durchgeführt wurden.

Gefundene Fehler in Prüf- und Korrekturmaßnahmen

| | sep_102 | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | C | D | E | F | G | H |
| SpezReview | 67.63 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 68 | | |
| | | | | | | 0.0 | |
| SpezReviewKorr | 62.22 | 0.0 | 0.0 | 0.0 | | | true |
| EReview | 36.14 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 37 | | |
| | | | | | | 0.0 | |
| EReviewKorr | 35.36 | 0.0 | 0.0 | 0.0 | | | true |
| MReview | 56.74 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 57 | | |
| | | | | | | 0.0 | |
| MReviewKorr | 53.12 | 0.0 | 0.0 | 0.0 | | | true |
| CReview | 87.52 | 18.04 | 0.0 | 0.0 | | | |
| | | | | | 106 | | |
| | | | | | | 0.0 | |
| CReviewKorr | 41.5 | 10.9 | 0.0 | 0.0 | | | true |
| HBReview | 31.56 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 32 | | |
| | | | | | | 157.694 | |
| HBReviewKorr | 19.84 | 0.0 | 0.0 | 0.0 | | | false |

B: Anzahl Fehler 1. Prüfung
C: Anzahl Fehler 2. Prüfung
D: Anzhl Fehler 3. Prüfung
E: Anzahl Fehler 4. Prüfung
F: Anzahl Fehler gesamt
G: geprüfter Umfang in AFPs
a H: vollständige Durchführung der Korrektur

*Tab. 4.5:* Gefundene Fehler in Prüf- und Korrekturmaßnahmen (sep_102)

*Gefundene Fehler in Prüfungen und Korrekturmaßnahmen*

Diese Tabelle zeigt für Reviews verschiedener Dokumente an, wie viele Fehler in den einzelnen Reviews gefunden wurden, und wie viele Fehler insgesamt durch Reviews entdeckt wurden. Außerdem wird aufgelistet, wie viele Fehler von den in Reviews gefunden Fehlern korrigiert wurden. Auch der geprüfte Umfang in AFPs und die vollständige Durchführung der Korrektur können daraus abgelesen werden. Anhand dieser Tabelle kann festgestellt werden, ob und wie oft der Spieler seine Dokumente geprüft hat und, ob die dabei gefundenen Fehler auch korrigiert wurden.

Die gefundenen Fehler in Prüf- und Korrekturmaßnahmen der Gruppe sep_102 sind in der Tabelle 4.5 dargestellt. Man kann dadurch recht einfach feststellen, wie oft der Spieler Reviews der Dokumente veranlasst hat. Gruppe

sep_102 hat nur für den Code zwei Reviews durchführen lassen, alle anderen Dokumente wurden nur einmal geprüft. Weiters hat man die Möglichkeit festzustellen, ob die Korrektur der gefundenen Fehler vollständig durchgeführt wurde, und ob es Gründe dafür gibt, anzunehmen, dass nicht der/die Autor(en) mit der Verbesserung des Dokuments beauftragt wurde(n).
Betrachtet man die durchgeführten Codereviews von Gruppe sep_102 genauer, kann man erkennen, dass von den 87 gefundenen Fehlern im ersten Review nur 41 verbessert wurden. Bei genauerer Analyse der Projektverlaufsdiagramme („Modulspezifikation und Codierung“ und „Test und Korrektur (Code)“) kann das eingesetzte Personal identifiziert werden. Die Gruppe sep_102 hat für die Erstellung des Codes Richard, Axel und Diana eingesetzt. Im ersten Review wurden Stefanie und Christine damit beauftragt das Dokument zu prüfen. Die Verbesserung der 87 gefundenen Fehler im Review wurde von Richard, Christine und Stefanie vorgenommen. Allerdings handelt es sich nur bei Richard um einen Autor. Christine und Stefanie kennen das Dokument nicht so gut und können daher nicht alle Fehler richtig verbessern. Außerdem fügen beide durch ihre Unwissenheit noch zusätzliche Fehler in das Dokument ein. Durch diese Fehlentscheidung konnten viele gefundene Fehler nicht richtig korrigiert werden.

Um den Studenten veranschaulichen zu können, warum manche Reviews effektiver durchgeführt wurden als andere, hat es sich als hilfreich erwiesen die Gutachterauswahl verschiedener Spieler gegenüberzustellen und zu vergleichen. Da die Probleme der Gutachterwahl in allen Phasen gleich sind, genügt es im Allgemeinen im Rahmen einer Feedback-Lehrveranstaltung eine genauere Analyse für nur eine Phase vorzunehmen. Die interessanten Aspekte, die man dabei aufzeigen kann, werden im folgenden Abschnitt am Beispiel Spezifikation gezeigt.

Zuerst muss ermittelt werden, welche Autoren die Spezifikation erstellt, welche Gutachter das Dokument geprüft und welche Mitarbeiter die in den Reviewsitzungen gefundenen Fehler im Spezifikationsdokument verbessert haben. Außerdem muss der Aufwand für die einzelnen Tätigkeiten ermittelt werden. Danach stellt man, wie in der Tabelle 4.6, die Gruppen gegenüber.

Nun kann man mit der genaueren Analyse beginnen. Es kann beim Betrachten der ersten zwei Spalten festgestellt werden, dass eine Person (wie bei Gruppe sep_111) zwar länger für das Erstellen des Spezifikationsdokuments braucht als zwei Personen (wie bei Gruppe sep_103 und sep_102), aber dass es auch auf die Qualifikation der Mitarbeiter ankommt, wie schnell sie ihre Tätigkeit beenden, denn man kann erkennen, dass Stefanie und Richard bes-

| Grp. | Spez. Team | Spez. Aufwand | Spez.Rev. Team | Spez.Rev. Aufwand | Spez.Korr. Team | Spez.Korr. Aufwand |
|---|---|---|---|---|---|---|
| sep102 | Thomas<br>Richard | 72 | Christine<br>Stefanie | 41,4 | Richard | 12,6 |
| sep103 | Stefanie<br>Richard | 64 | Bernd<br>Thomas<br>Axel<br>Diana<br>Christine | 131 | Richard<br>Axel<br>Diana<br>Christine | 31,8 |
| sep111 | Richard | 80 | Christine<br>Stefanie<br>Thomas | 83,7 | Richard<br>Stefanie<br>Thomas | 13 |
| sep110 | Thomas<br>Diana<br>Richard | 64 | Bernd<br>Stefanie | 40,4 | Thomas<br>Bernd | 11,7 |

*Tab. 4.6:* Autoren der Spezifikation

ser im Spezifizieren sind als Thomas und Richard.
Man kann auch feststellen, dass drei Personen (sep_110) für das Erstellen des Dokuments bereits gleich viel Zeit benötigen wie zwei Mitarbeiter (sep_103). Die Begründung ist einfach. Der Kommunikationsaufwand ist bei drei Personen wesentlich höher als bei zwei. Wie bereits beschrieben wurde, hat der Kommunikationsaufwand aber auch Auswirkungen auf andere Bereiche des Projekts wie zum Beispiel auf das Budget. Je mehr Mitarbeiter also an einer Phase beteiligt sind, desto mehr müssen sie ihre Aufgaben mit denen der anderen Mitarbeiter absprechen, und desto länger braucht jeder einzelne für die Fertigstellung seiner Tätigkeit.

Betrachtet man nun den Reviewbereich, so kann festgestellt werden, dass Gruppe sep_103 viele verschiedene Gutachter eingesetzt hat, wobei Gruppe sep_102 und Gruppe sep_110 nur zwei Personen beauftragt haben.

Um weitere Aussagen über die Reviewsitzungen treffen zu können, müssen alle vorhandenen Daten für die Reviews ermittelt und gegenübergestellt werden. Erst dann kann geprüft werden, wie viele Reviews die einzelnen Gruppen durchgeführt haben, und wie oft der Spieler dasselbe Reviewteam eingesetzt hat, denn wie wir sehen werden, hat auch das Auswirkungen auf die Anzahl der gefunden Fehler.

Diese Auswirkungen werden in den nächsten vier Tabellen veranschaulicht.

| Grp. | Spez. | Fehler | Review | gef. F | Korrektur | korr. F | Restf. |
|---|---|---|---|---|---|---|---|
| sep101 | Christine Richard | 121 | Christine Richard Stefanie | 53,25 | Christine Richard | 48,99 | 72 |
| sep109 | Christine Richard | 123 | Diana Stefanie | 74,14 | Christine Richard | 68,21 | 48 |
| | | | Diana Stefanie | 7,38 | Christine | 6,79 | |

*Tab. 4.7:* Gutachter in Spezifikationsreviews (1)

Der Aufbau dieser Tabellen ist immer der gleiche. Auch weiterhin untersuchen wir nur das Spezifikationsdokument. Die erste Spalte enthält die Autoren des Spezifikationsdokuments. In der zweiten Spalte werden die Fehler, die das Dokument vor der Reviewsitzung enthielt, angeführt. Spalte drei zeigt die Gutachter der einzelnen Reviews, gefolgt von der Anzahl der von ihnen gefundenen Fehler. Die nächsten zwei Spalten listen die für die Korrektur eingesetzten Mitarbeiter und die von ihnen korrigierten Fehler auf. Die Restfehler des Spezifikationsdokuments am Ende des Projekts werden in der letzten Spalte aufgezeigt.

Die Tabelle 4.7 zeigt, wie wichtig es ist, dass der Autor eines Dokuments nicht als Gutachter eingesetzt wird. Man kann erkennen, dass bei der Gruppe sep_101 eigentlich nur Stefanie Fehler findet. Christine und Richard finden keine oder kaum Fehler, weil anzunehmen ist, dass sie als Autoren ihr Dokument bereits geprüft haben und für sie das Dokument fehlerfrei ist. Sollten sie während der Reviewsitzung doch Fehler entdecken, dann ist es eher unwahrscheinlich, dass sie diese auch vor den anderen Gutachtern zugeben.

Bei Gruppe sep_109 werden wesentlich mehr Fehler gefunden. Bei dieser Gruppe wäre es hingegen klüger gewesen, im zweiten Reviews andere Gutachter einzusetzen, die hätten eine größere Anzahl von Fehlern finden können.

Bei der Korrektur kann man feststellen, dass sowohl bei der Gruppe sep_101 als auch bei der Gruppe sep_109 die Autoren für die Korrektur des Dokuments eingesetzt wurden. Das ist von Vorteil, denn es ist anzunehmen, dass der Autor sich am besten in seinem Dokument auskennt, und dass dadurch die Fehler sehr rasch korrigiert werden können. Außerdem kann davon ausgegangen werden, dass der Autor während der Korrektur weniger neue Fehler

| Grp. | Spez. | Fehler | Review | gef. F | Korrektur | korr. F | Restf. |
|---|---|---|---|---|---|---|---|
| sep102 | Thomas Richard | 101 | Christine Stefanie | 67,63 | Richard | 62,22 | 38 |
| sep110 | Thomas Diana Richard | 130 | Bernd Stefanie | 42,59 | Thomas Bernd | 34,75 | 95 |

*Tab. 4.8:* Gutachter in Spezifikationsreviews (2)

in das Dokument einfügt.

Das Spezifikationsdokument der Gruppe sep_109 weist wesentlich weniger Restfehler auf als das der Gruppe sep_101. Die Gruppe sep_101 hätte besser noch einen zweiten Review des Dokuments durchführen sollen, um die Restfehler zu senken, denn diese Fehler pflanzen sich über die Projektdauer hinweg fort. Je später ein Fehler gefunden wird, desto teurer wird das Beseitigen dieses Fehlers, da in diesem Fall auch alle Nachfolgedokumente korrigiert werden müssen.

Wie aus Tabelle 4.8 deutlich hervorgeht, kann sich der Einsatz eines dritten Autors nachteilig auf die Korrektheit eines Dokuments auswirken. Gruppe sep_110 hat durch den Einsatz von Thomas, Diana und Richard ein Spezifikationsdokument, das 130 Fehler enthält. Im Fall von Gruppe sep_102 hingegen haben Thomas und Richard nur 101 Fehler eingefügt. Je mehr Mitarbeiter an der Erstellung eines Dokuments teilnehmen, desto mehr Fehler wird das Dokument enthalten, da jeder Mitarbeiter einige zusätzliche Fehler in das Dokument einfügt.

Bei einem Vergleich der Reviewteams fällt auf, dass das Team von Gruppe sep_110 wesentlich weniger Fehler gefunden hat als das von Gruppe sep_102. Stefanie ist in beiden Fällen eingesetzt worden. Sie zählt zu den am besten qualifizierten Mitarbeitern in diesem Bereich. Christine hat im Vergleich zu Bernd noch wesentlich mehr Fehler gefunden. Der Einsatz von Bernd als Gutachter hat sich nachteilig auf die Korrektheit des Spezifikationsdokuments ausgewirkt.

Untersuchen wir nun noch die Korrekturphase. Gruppe sep_102 hat einen der Autoren für die Korrektur der gefundenen Fehler eingesetzt. Allerdings hat Richard nur einige der Fehler im Dokumentteil von Thomas lokalisieren und verbessern können. Wäre auch Thomas mit der Korrektur seines Teils

| Grp. | Spez. | Fehler | Review | gef. F | Korrektur | korr. F | Restf. |
|---|---|---|---|---|---|---|---|
| sep105 | Richard | 116 | Bernd Christine | 65,99 | Richard | 60,71 | 55 |
| sep111 | Richard | 117 | Christine Stefanie | 70,94 | Richard | 65,26 | 29 |
| | | | Christine Stefanie | 23,8 | Richard Stefanie | 21,9 | |
| | | | Christine Thomas | 0,19 | Stefanie Thomas | 0,18 | |

*Tab. 4.9:* Gutachter in Spezifikationsreviews (3)

beauftragt worden, dann hätte er noch mehr Fehler verbessern können. Gruppe sep_110 hat einen Autor (Thomas) zum Verbessern der Fehler eingesetzt. Als zweiter Mitarbeiter wurde Bernd mit der Korrektur des Dokuments beauftragt. Da Thomas hauptsächlich die Fehler, die seinen Teil des Dokuments betreffen, verbessert, bleibt für Bernd der Teil von Diana und Richard übrig. Er kennt sich aber im Dokument nicht aus und kann daher nur sehr wenige Fehler verbessern. Außerdem wird er durch seine Unwissenheit einige zusätzliche Fehler in das Dokument einfügen.

In der Tabelle 4.9 sind zwei Gruppen gegenübergestellt, die beide Richard als Autor für das Spezifikationsdokument eingesetzt haben. Wie man in der zweiten Spalte erkennen kann, hat Richard in beiden Fällen beinahe gleich viele Fehler gefunden. Gruppe sep_105 hat dann Bernd und Christine als Gutachter eingesetzt, während sich Gruppe sep_111 für Christine und Stefanie entschieden hat. Christine und Stefanie haben auch tatsächlich mehr Fehler gefunden, als Bernd und Christine. Die Korrektur des Dokuments hat in beiden Fällen wieder der Autor vorgenommen, wodurch beinahe alle gefundenen Fehler beseitigt werden konnten.

Betrachten wir die letzte Spalte, die die Restfehler des Spezifikationsdokuments aufzeigt, dann wäre es für die Gruppe sep_105 besser gewesen, noch eine weitere Prüfung des Dokuments zu veranlassen, um zumindest einige der 55 verbleibenden Analysefehler noch zu beseitigen.

Untersuchen wir nun die weiteren Reviews von Gruppe sep_111 genauer. Auch im zweiten Review wurden wieder Christine und Stefanie als Gutach-

| Grp. | Spez. | Fehler | Review | gef. F | Korrektur | korr. F | Restf. |
|---|---|---|---|---|---|---|---|
| sep103 | Stefanie Richard | 124 | Thomas Bernd | 45,91 | Richard Axel | 29,56 | 66 |
| | | | Bernd Thomas Axel | 25,44 | Diana | 21,07 | |
| | | | Diana Christine Axel | 10,59 | Christine | 6,82 | |

*Tab. 4.10:* Gutachter in Reviews

ter eingesetzt. Sie fanden jedoch noch einige Fehler, die von Richard und Stefanie verbessert wurden. Der Einsatz von Stefanie in der Korrekturphase könnte sich nachteilig auf die Anzahl der Restfehler im Dokument ausgewirkt haben. Da Stefanie nicht so gut wie Richard mit dem Spezifikationsdokument vertraut ist, kann angenommen werden, dass sie bei der Verbesserung von Fehlern zusätzliche Fehler in das Dokument eingefügt hat. Im dritten Review der Gruppe sep_111 wurden Stefanie und Thomas als Prüfer eingesetzt. Wie man erkennen kann, wurden keine Fehler mehr gefunden. Das Spezifikationsdokument wurde von dieser Gruppe gründlich geprüft und auch die 29 Restfehler im Dokument weisen auf ein sehr korrektes Dokument hin. Natürlich muss hier angemerkt werden, dass die Gruppe sep_111 durch die Durchführung von drei Reviews auch wesentlich mehr Budget bzw. Zeit verbraucht hat, als Gruppe sep_105.

Gruppe sep_103 hat, wie aus Tabelle 4.10 hervorgeht, Stefanie und Richard für die Erstellung der Spezifikation beauftragt. Im ersten Review haben Thomas und Bernd das Dokument geprüft. Da nicht die qualifiziertesten Mitarbeiter (Stefanie ist aufgrund ihrer Angaben am qualifiziertesten für das Prüfen von Dokumenten) für das Review eingesetzt wurden, wurden nur 45 Fehler gefunden. Für die Korrektur des Dokuments wurde nur einer der Autoren eingesetzt. Dies führte dazu, dass Richard nur die Fehler in seinem Teil des Dokuments verbessert hat. Die Fehler im Teil von Stefanie wurden durch Axel verbessert, der sich jedoch im Dokument nicht gut auskennt, und dadurch neue Fehler in das Dokument eingefügt hat. Es wurden dadurch nur 29 der 45 Fehler verbessert. Auch der Einsatz von Diana und Christine für die Korrektur der Fehler, die in Review zwei und drei gefunden wurden, führte zu zusätzlichen Fehlern im Spezifikationsdokument.

Die Gruppe sep_103 hat als einzige Gruppe Reviews mit drei Gutachtern

durchgeführt. Wenn wir das zweite Review von sep_103 mit dem zweiten Review von sep_111 in Tabelle 4.9 vergleichen, können wir erkennen, dass auch von drei schlecht gewählten Gutachtern nicht mehr Fehler gefunden werden als von zwei Gutachtern.

Es konnte festgestellt werden, dass drei unabhängige Gutachter nur unwesentlich mehr Fehler in der Reviewsitzung finden, als zwei. Das liegt daran, dass viele Fehler von mehreren Gutachtern gefunden werden und jeder Gutachter nur ein paar neue Fehler findet. Das bedeutete je höher die Anzahl der Gutachter, die an einem Review teilnehmen, gewählt wird, desto höher ist die Anzahl der Fehler die von mehreren Gutachtern gefunden werden.

Im SESAM-Spiel können nur Reviews mit zwei oder drei Gutachtern durchgeführt werden. Es hat sich aber als günstiger erwiesen nur Reviews mit zwei Gutachtern (gegebenenfalls mit Kunde) durchzuführen, da die anfallenden Kosten eines Reviews mit drei Gutachtern in keiner Relation zum Nutzen stehen.

*Gefundene Fehler in Test- und Korrekturmaßnahmen*

Diese Tabelle zeigt für die unterschiedlichen Tests jeweils die gefunden und korrigierten Fehler, den geprüften Umfang in AFPs und, ob die Korrektur vollständig durchgeführt wurde.

Betrachten wir als Beispiel die Gruppe sep_102 in der Tabelle 4.11, so können auch in diesem Fall, wie bei der Tabelle „Gefundene Fehler in Prüf- und Korrekturmaßnahmen“, falsch gewählte Gutachter entlarvt bzw. das Abbrechen der Korrekturphase erkannt werden.

Anhand der Tabelle 4.11 kann man gut erkennen, welche Auswirkungen das Abbrechen der Korrektur haben kann. Betrachtet man nochmals das Projektverlaufsdiagramm „Test und Korrektur (Code)“ in Abbildung 4.22 genauer, so stellt man fest, dass die Gruppe sep_102 Bernd, Diana und Christine für die Durchführung der Modultests eingesetzt hat. Die Vorgängerphase (Codeerstellung) wurde am 18.10.1999 begonnen. Die Gruppe sep_102 hat bereits gegen Ende Oktober versucht Bernd mit dem Modultest zu beauftragen. Da aber zu diesem Zeitpunkt noch nicht 50 % des Codes fertig gestellt waren, konnte Bernd nicht mit dem Modultest beginnen. Erst als die Gruppe Anfang Dezember Diana mit dem Modultest beauftragt, liegt das Vorgabedokument vor und der Test kann durchgeführt werden. Diana findet im ersten Mo-

Gefundene Fehler in Test– und Korrekturmaßnahmen

| | sep_102 | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | C | D | E | F | G | H |
| MTest | 117.86 | 11.95 | 0.0 | 0.0 | | | |
| | | | | | 130 | | |
| | | | | | | 197.166 | |
| MTestKorr | 100.82 | −7.0 | 0.0 | 0.0 | | | false |
| ITest | 111.26 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 112 | | |
| | | | | | | 0.0 | |
| ITestKorr | 49.38 | 0.0 | 0.0 | 0.0 | | | true |
| STest | 114.86 | 25.77 | 0.0 | 0.0 | | | |
| | | | | | 141 | | |
| | | | | | | 0.0 | |
| STestKorr | 47.18 | 19.74 | 0.0 | 0.0 | | | true |
| ATest | 31.9 | 13.98 | 0.0 | 0.0 | | | |
| | | | | | 46 | | |
| | | | | | | 197.166 | |
| ATestKorr | 25.69 | 2.02 | 0.0 | 0.0 | | | false |

B: Anzahl Fehler 1. Prüfung
C: Anzahl Fehler 2. Prüfung
D: Anzahl Fehler 3. Prüfung
E: Anzahl Fehler 4. Prüfung
F: Anzahl Fehler gesamt
G: geprüfter Umfang in AFPs
H: vollständige Durchführung der Korrektur

*Tab. 4.11:* Gefundene Fehler in Test- und Korrekturmaßnahmen (sep_102)

dultest 117 Fehler, davon können 100 Fehler von Richard auch verbessert werden. Ein zweiter Modultest, der von Christine durchgeführt wird, kann weitere 11 Fehler aufdecken. Christine wird auch mit der Korrektur dieser Fehler beauftragt. Wie wir in der Tabelle erkennen können, wurde die Korrektur vorzeitig abgebrochen. Christine hatte dadurch keine Möglichkeit alle gefundenen Fehler zu verbessern. Welche Auswirkungen diese Entscheidung verursacht hat, kann man an der Anzahl der korrigierten Fehler im zweiten Review erkennen. Christine hat demnach keinen der 11 gefundenen Fehler verbessert, sondern stattdessen 7 neue Fehler in das Dokument eingefügt. An diesem Beispiel sehen wir, wie wichtig es ist, die Autoren für das Korrigieren des eigenen Dokuments einzusetzen.

Der Code wurde von Richard, Axel und Diana erstellt. Richard, der Au-

tor des Codes ist, konnte viele der im ersten Modultest gefundenen Fehler verbessern. Christine kennt sich im Code nicht aus, da sie nicht Autor ist. Darum benötigt sie viel mehr Zeit, um die Fehler zu lokalisieren und auszubessern. Jedoch fügt sie aufgrund ihrer Unwissenheit dabei zusätzliche Fehler in den Code ein. Durch den Abbruch der Korrektur und das Ausbleiben von weiteren Modultests, verblieben die gefundenen und noch nicht verbesserten Fehler sowie die Fehler, die Christine zusätzlich eingefügt hat, im Code und haben somit die Vollständigkeit und Korrektheit des Dokuments verschlechtert.

Sollte aus der Tabelle hervorgehen, dass der Spieler die Korrektur gar nicht begonnen hat, dann muss dies nicht bedeuten, dass er darauf vergessen hat. Wie bereits erwähnt wurde, ist es vorgekommen, dass die Projektkosten eine erneute Korrektur nicht mehr erlaubt hätten. Dies ist jedoch eher in der Testphase vorgekommen, da zu diesem Zeitpunkt das Projekt schon recht weit fortgeschritten war. Ob es allerdings besser ist, die gefundenen Fehler im Dokument zu lassen, oder stattdessen die Fehler zu verbessern und höhere Projektkosten in Kauf zu nehmen, muss individuell geprüft werden.

Um den Studierenden veranschaulichen zu können, warum manche Tests effektiver durchgeführt wurden als andere, wird die Testerwahl verschiedener Spieler gegenübergestellt. Dabei wird gleich vorgegangen, wie es im Abschnitt „Gefundene Fehler in Prüf- und Korrekturmaßnahmen“ gezeigt wurde.
Bei den Tests gibt es jedoch nicht die Einschränkung wie bei Reviews, dass die Prüfung des Dokuments nicht durch die Autoren erfolgen soll. Dieser Effekt wurde im QS-Modell nicht realisiert. Zum Testen eines Dokuments kann jeder Mitarbeiter eingesetzt werden. Es muss allerdings darauf geachtet werden, dass es auch in diesem Bereich qualifizierte und weniger qualifizierte Mitarbeiter gibt.

In der Praxis sollte der Test aber immer von einem unabhängigen Mitarbeiter durchgeführt werden, da der Autor sein Dokument kennt, und den Test dadurch gewollt oder ungewollt beeinflussen kann.
Für die Korrektur des Dokuments gelten die gleichen Einschränkungen wie bei Reviews. Die in den Tests gefundenen Fehler sollten immer vom Autor verbessert werden, da der Autor sein Dokument kennt und die Fehler rasch korrigieren kann.

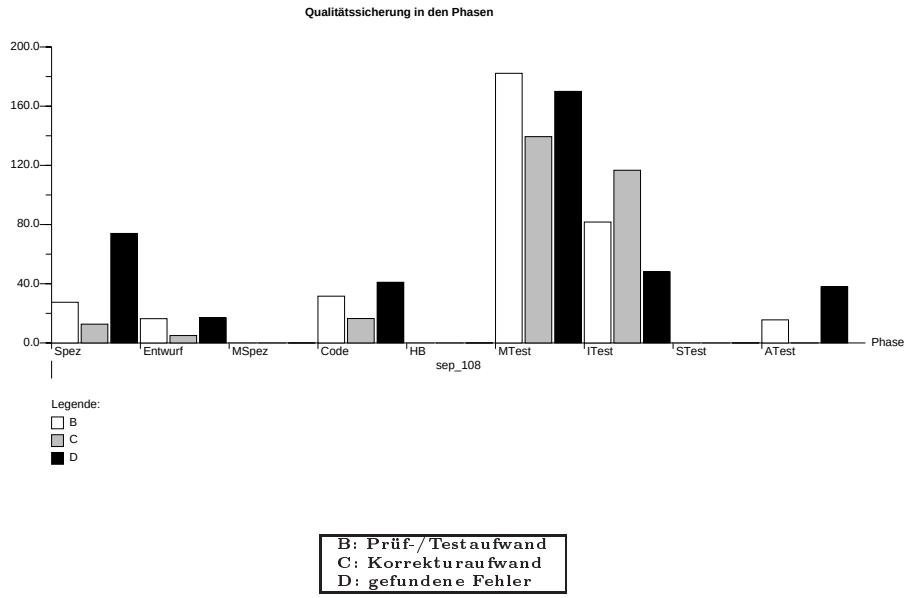



*Abb. 4.33:* Qualitätssicherung in den Phasen (sep_108)

*Qualitätssicherung in den Phasen*

In diesem Diagramm werden der Aufwand (in Stunden) für die Prüfung durch Reviews/Tests, der Aufwand für die Korrektur und die gefundenen Fehler für jede einzelne Phase dargestellt.
Dieses Diagramm zeigt, ob sich der eingesetzte Aufwand für das Finden von Fehlern gelohnt hat, oder ob der Spieler einen zu hohen Aufwand im Verhältnis zu den gefundenen Fehlern benötigt hat.
Das Diagramm liefert zusätzliche Anhaltspunkte, wenn besonders viele oder besonders wenige Fehler gefunden wurden. Ein Grund dafür, dass die Gutachter besonders wenig Fehler gefunden haben, liegt darin, dass die ausgewählten Gutachter nicht qualifiziert genug waren, das bedeutet, der Spieler hat vielleicht die Autoren des Dokuments als Prüfer eingesetzt. Ein erhöhter Korrekturaufwand hingegen würde auf schlecht ausgewählte Mitarbeiter hinweisen. Es sollte in diesem Fall geprüft werden, ob für die Korrektur andere Mitarbeiter als die Autoren des Dokuments eingesetzt wurden.

Anhand der Abbildung 4.33 kann festgestellt werden, in welchen Phasen die Gruppe sep_108 Qualitätssicherungen vorgenommen hat. Man kann erkennen, dass weder die Modulspezifikation noch das Handbuch einem Review unterzogen wurden. Welche Auswirkungen diese Entscheidungen mit

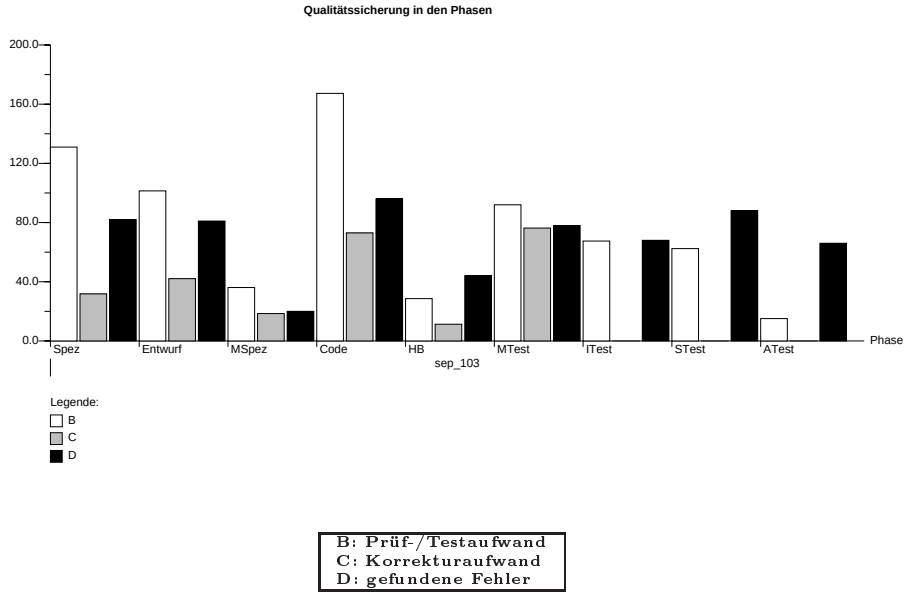



*Abb. 4.34:* Qualitätssicherung in den Phasen (sep_103)

sich brachten, erkennen wir, wenn wir nochmals die Zielvorgaben (Tabelle 4.1), die von dieser Gruppe nicht erreicht wurden, betrachten. Die geforderten AFPs für den Code und für das Handbuch wurden aufgrund der fehlenden Qualitätssicherung nicht erreicht.
Weiters kann dem Diagramm entnommen werden, dass die Gruppe viel Aufwand in den Modultest und den Integrationstest gesteckt hat. Es wurde jedoch kein Systemtest vorgenommen. Der Grund dafür ist unklar, da die Gruppe noch genügend finanzielle Mittel zur Verfügung gehabt hätte. Außerdem wurde darauf verzichtet, die Fehler zu korrigieren, die der Kunde im Abnahmetest gefunden hat.

Das Diagramm der Gruppe sep_103 in Abbildung 4.34 zeigt, dass die Gruppe alle Dokumente einer Prüfung unterzogen hat. Auffällig ist aber, dass Integrations-, System- und Abnahmetest zwar durchgeführt, die gefundenen Fehler jedoch nicht mehr korrigiert wurden. Vergleichen wir wieder die Zielvorgaben in Tabelle 4.1, dann wird klar, dass diese Gruppe bei den Kosten bereits zu diesem Zeitpunkt im Projekt weit über der vorgeschriebenen Grenze von 450 000 DM lag, und daher keine finanziellen Mittel mehr zur Verfügung hatte, um diese Korrekturen vorzunehmen.

| Gruppe | Fehlerart | Spez. | Entwurf | Mspez | Code | HB |
|---|---|---|---|---|---|---|
| sep_103 | Analysefehler | 66 | 93 | 106 | 106 | 56 |
| | Grobentwurfsfehler | | 43 | 52 | 42 | |
| | Feinentwurfsfehler | | | 76 | 51 | |
| | Implementierungsfehler | | | | 63 | |
| | Handbuchfehler | | | | | 31 |
| sep_111 | Analysefehler | 29 | 29 | 36 | 30 | 27 |
| | Grobentwurfsfehler | | 15 | 18 | 13 | |
| | Feinentwurfsfehler | | | 33 | 18 | |
| | Implementierungsfehler | | | | 48 | |
| | Handbuchfehler | | | | | 59 |

*Tab. 4.12:* Restfehler in den Dokumenten

### *4.3.5 Dokumente*

Der folgende Abschnitt beschäftigt sich mit Tabellen, die verschiedene Kriterien von Dokumenten genauer betrachten.

#### *Restfehler der Dokumente*

Die Tabelle zeigt die Restfehler in den verschiedenen Dokumenten nach Abschluss des Projekts. Die Fehler werden nach Fehlerart (Analyse-, Grobentwurfs-, Feinentwurfs-, Implementierungs- und Handbuchfehler) aufgeschlüsselt gezeigt.

Eine Analyse dieser Fehler zeigt, ob der Spieler bereits am Anfang des Projekts auf das Verbessern von Fehlern geachtet hat. Außerdem kann festgestellt werden, ob genügend Prüfungen für das Dokument veranlasst wurden, oder ob zu früh mit der nächsten Phase begonnen wurde und dadurch viele Fehler erst später oder gar nicht entdeckt und verbessert wurden.

Stellen wir Gruppe sep_103 und Gruppe sep_111 in der Tabelle 4.12 gegenüber, so kann man feststellen, dass Gruppe sep_103 bereits nach der Spezifikationsphase noch 66 Restfehler im Dokument hatte. Im Code gab es bereits 106 Analysefehler, wohingegen die Fortpflanzung der Fehler bei Gruppe sep_111 gering war bzw. im Code wieder abnahm. Bei der Gruppe sep_103 wurden die in Tests gefundenen Fehler nicht mehr verbessert, dadurch zeichnet sich keine Verbesserung von der Modulspezifikation zur

| Gruppe / AFP | Spez | Entwurf | Mspez | Code | | HB |
|---|---|---|---|---|---|---|
| sep_101 | 197,12 | 193,74 | 187,30 | 192,73 | | 193,89 |
| sep_102 | 198,73 | 197,05 | 195,42 | 197,14 | | 193,92 |
| sep_103 | 199,98 | 192,28 | 183,45 | 182,71 | | 195,68 |
| sep_104 | 197,64 | 194,75 | 185,67 | 186,14 | | 193,02 |
| sep_105 | 198,03 | 195,62 | 191,81 | 194,33 | | 194,22 |
| sep_108 | 197,83 | 193,26 | 188,17 | 188,27 | | 187,94 |
| sep_109 | 199,07 | 197,76 | 195,46 | 194,34 | | 187,43 |
| sep_110 | 196,95 | 193,65 | 189,16 | 194,10 | | 187,10 |
| sep_111 | 200,38 | 198,80 | 195,77 | 195,82 | | 191,50 |
| sep_112 | 198,78 | 193,33 | 184,53 | 195,05 | | 193,47 |

*Tab. 4.13:* Vollständigkeit

Codierung ab wie bei der Gruppe sep_111.

*Vollständigkeit der Dokumente*

Diese Tabelle zeigt den Umfang in AFPs für die verschiedenen Dokumente (Spezifikation, Entwurf, Modulspezifikation, Codierung, Handbucherstellung). Damit kann die Vollständigkeit der Dokumente geprüft und interessante Aspekte entdeckt werden.

In der Tabelle 4.13 wurden die Daten aller Spieler zusammengefasst, um einen Überblick über die Vollständigkeit der Dokumente geben zu können. Bei Gruppe sep_103 ist die Vollständigkeit der Dokumente im Code wieder gesunken. Während sie sonst immer zunahm. Wie wir bereits gesehen haben, hatte diese Gruppe keine finanziellen Mittel mehr zur Verfügung, um die in den Tests gefundenen Fehler noch zu korrigieren.

Bei Gruppe sep_111 konnte bereits zu Beginn ein Dokument mit über 200 AFPs entwickelt werden, die Restfehler waren also sehr gering. Dieser Projektverlauf zeigt, wie wichtig es ist, die frühen Phasen so gründlich wie möglich zu erledigen. Gruppe sep_102 konnte durch intensives Testen des Codes viele Fehler aufdecken und eliminieren und somit die in der Modulspezifikation verlorenen AFPs im Code wieder einbringen.

## 4.4 Erfahrungen

In den letzten Abschnitten wurden die wichtigsten SesamScore-Diagramme vorgestellt, und beschrieben, welche Fehlentscheidungen des Spielers der Tutor daraus erkennen kann. Im Folgenden werden die Erfahrungen, die bei den Testspielen in der Lehrveranstaltung „Systementwicklungsprozess“ gesammelt werden konnten, zusammengefasst. Dabei handelt es sich um typische Probleme, die Studierende bei der Durchführung des ersten Spiels haben. Es wird dabei jedoch nur auf die Schwierigkeiten im Projektmanagement und nicht auf die Probleme mit der Benutzerschnittstelle eingegangen.

### 4.4.1 Unproduktive Zeit

Viele Spieler hatten enorme Probleme mit der Auslastung ihrer Mitarbeiter. Dies führte einerseits zu höheren Kosten, andererseits konnte beobachtet werden, dass im Gegensatz zu den meisten realen Projekten erstaunlich viel Aufwand für die Ausarbeitung des Handbuchs investiert wurde. Der Grund dafür liegt darin, dass die Spieler die Mitarbeiter, für die keine Tätigkeit gefunden werden konnte, zur Erstellung des Handbuchs eingesetzt haben.

### 4.4.2 Erreichen der Zielvorgaben

Das Erreichen der Zielvorgaben ist nicht einfach. Nur ein Team erreichte bereits beim ersten Spiel alle Zielvorgaben. Warum die Zielvorgaben nicht erreicht wurden, hatte mehrere Gründe. Die Vollständigkeit und Korrektheit vieler Zwischendokumente war bei vielen Spielverläufen zu gering, um die Zielvorgaben erfüllen zu können. Bereits das Ausgangsdokument, die Spezifikation, enthielt in den meisten Fällen zu viele Fehler, die sich dann im Laufe des Projekts noch fortpflanzten. Das Modulspezifikationsdokument schnitt in Vollständigkeit und Korrektheit am Schlechtesten ab, da im Allgemeinen zu früh mit der Codierung begonnen wurde, anstatt die Entwurfsdokumente genauer zu prüfen und zu verbessern.

Auffallend war aber, dass ein Qualitätsabfall vom Entwurf zur Modulspezifikation und dann wieder die Steigerung der Qualität hin zum Code beobachtet werden konnte. Der Qualitätsabfall tritt auf, weil die Spieler die Zwischendokumente nicht rechtzeitig nachführen. Dadurch werden viele entdeckte und im Vorgabedokument bereits verbesserte Fehler in das Nachfolgedokument übernommen. Nur wenige Spieler haben eine Korrektur der Nachfolgedokumente vorgenommen. Der Anstieg der Qualität des Codes kann auf den höheren Testaufwand zurückgeführt werden, der durch das herannahen der

Deadline zu einer stärkeren Konzentration auf das auszuliefernde System geführt hat.

### 4.4.3 Teilnahme des Kunden

Ein weiteres Problem bei der Erreichung der Zielvorgaben war die Funktionalität, die nicht in dem Ausmaß erreicht werden konnte, wie es vom Kunden gewünscht war. In vielen Spielverläufen war der Kunde nicht ausreichend am Prozess beteiligt. Dies konnte vor allem bei den Handbuchreviews festgestellt werden. Nur drei der zehn Gruppen haben den Kunden dazu eingeladen.
In der Spezifikationsphase haben immerhin neun Gruppen den Kunden an Reviews teilnehmen lassen. Zwei Gruppen davon haben den Kunden sogar zu zwei Reviews eingeladen. Nur eine Gruppe (sep_103) hat ihr Projekt ohne Rücksprache mit dem Kunden durchgeführt.

### 4.4.4 Parallelität

Häufig wurde mit der Erstellung der Folgedokumente bereits begonnen, noch ehe das Vorgabedokument geprüft und korrigiert war. Dadurch wurden viele Fehler in das Nachfolgedokument übertragen, obwohl man sie bereits im Vorgabedokument entdeckt hatte.
Der Grund dafür dürfte bei den meisten Spielern ein fehlender Überblick über den Prozess gewesen sein. Es konnte häufig festgestellt werden, dass wenn Vorgabedokumente später korrigiert wurden, darauf vergessen wurde, auch die bereits erstellten Nachfolger zu verbessern.

### 4.4.5 Abbrechen von Tätigkeiten

Manche Spieler waren gezwungen ihren Mitarbeitern Tätigkeiten zu entziehen, um ihnen andere dringendere Aufgaben zuteilen zu können. Häufig wurde darauf vergessen die begonnenen Tätigkeiten von jemand anderem fertig stellen zu lassen. Eine Gruppe hat zum Beispiel vergessen, die abgebrochene Korrektur eines Reviews von einem anderen Mitarbeiter beenden zu lassen und hatte dadurch keine Möglichkeit mehr, das Dokument einer weiteren Prüfung zu unterziehen.

Viele dieser Probleme sind typisch für das erste SESAM-Spiel und können bei erfahrenen Spielern nicht mehr in diesem Ausmaß aufgezeigt werden. Damit diese Schwierigkeiten nicht auftreten, muss der Tutor eine genaue Auswertung des ersten Spiels vornehmen und die Spieler auf diese Probleme hinweisen. Wenn die Spieler sich an diese Tipps halten und das Erlernte umsetzen,

dann werden sie feststellen, dass sie im nächsten Spiel bereits wesentlich besser abschneiden als im ersten.
Generell konnten dieselben Erfahrungen gesammelt werden, die auch schon bei Testspielen auf der Universität Stuttgart [MANDL-STRIEGNITZc] gemacht wurden. Alle Studenten haben angegeben, dass sie etwas über Projektmanagementfunktionen und über die Schwierigkeiten bei der Durchführung eines Software-Entwicklungsprojekts gelernt haben. Es wurde jedoch festgestellt, dass dieser Lernerfolg nur dann erzielt wird, wenn die Spieler eine genaue Analyse ihrer Spielverläufe erhalten. Dann können die Studierenden ihre falschen Entscheidungen erkennen und beim nächsten Spiel vermeiden. Außerdem wollte ein Großteil der Gruppe ein weiteres Spiel durchführen, um eine andere Projektmanagement-Strategie auszuprobieren.

Die „interessanten“ Aspekte, die in diesem Kapitel aufgezeigt wurden, werden in Kapitel 5 in Form von Bewertungskriterien zusammengefasst, die von der generische Erklärungskomponente benötigt werden, um eine Auswertung von Projektverläufen vornehmen zu können.

*Kapitel 5*

# IDENTIFIZIERTE BEWERTUNGSKRITERIEN FÜR DAS QS-MODELL

In diesem Abschnitt werden die im Kapitel 4 identifizierten und für die spätere Auswertung als wichtig erachteten Bewertungskriterien für das QS-Modell aufgelistet. Die empirische Datenbasis [JONES], die dem QS-Modell zugrunde liegt sowie die Effekte, die im QS-Modell realisiert wurden, werden in der Dissertation von Anke Drappa [DRAPPAc] genauer ausgeführt. Da sich viele Effekte überlagern (die Komplexität wird anhand eines Graphen in Anhang A gezeigt), werden die Bewertungskriterien zu folgenden Bereichen zusammengefasst, um eine Gruppierung zu ermöglichen.

| Gruppe | Bewertungskriterien |
|---|---|
| Zielvorgaben (Z): | Z1: Dauer des Projekts<br>Z2: Höhe der Projektkosten<br>Z3: Realisierte AFPs in Prozent im Code<br>Z4: Enthaltene Fehler pro KLOC<br>Z5: Realisierte AFPs in Prozent im Handbuch<br>Z6: Enthaltene Fehler pro Seite im Handbuch |
| Mitarbeiter (M): | M1: Anzahl Mitarbeiter pro Phase (gleichzeitig)<br>M2: Anzahl Mitarbeiter pro Phase (nicht gleichzeitig)<br>M3: Unproduktive Zeit pro Mitarbeiter<br>M4: Qualifikation der Mitarbeiter |
| Dokumente (D): | D1: Vollständigkeit der Dokumente<br>D2: Restfehler der Dokumente<br>D3: Autoren der Dokumente<br>D4: Kosten für das Dokument |

| | |
|---|---|
| Reviews (R): | R1: Eingesetztes Reviewteam<br>R2: Anzahl der beteiligten Gutachter<br>R3: Erfolgreiche Durchführung der Korrekturphase<br>R4: Korrektur durch den Autor<br>R5: Dauer von Reviewphasen<br>R6: Effizienz von Reviews<br>R7: Effektivität von Reviews<br>R8: Anzahl erfolgreicher/nicht erfolgreicher Reviews<br>R9: Verluste durch Reviews |
| Tests (T): | T1: Tester<br>T2: Korrektur von Tests<br>T3: Verluste durch Tests<br>T4: Effizienz von Tests<br>T5: Effektivität von Tests |
| Phasen (P): | P1: Parallelitäten<br>P2: Vorgabedokument zu 50 % erstellt<br>P3: Beginn der Nachfolgephase erst nach Korrektur der Vorgängerphase<br>P4: Modultest nach Codereview<br>P5: Integrationstest nach Modulspezifikation-Review<br>P6: Aufwandsverteilung |
| Kunde (K): | K1: Teilnahme des Kunden an Spezifikationsreview<br>K2: Teilnahme des Kunden an Handbuchreview<br>K3: Handbuchreview zu frühem Zeitpunkt |
| Projektplanung (PP): | PP1: Anzahl Inspektionen (Tätigkeiten)<br>PP2: Anzahl Inspektionen (verbrauchte Ressourcen) |

*Tab. 5.1:* Identifizierte Bewertungskriterien für das QS-Modell

Es muss jedoch an dieser Stelle darauf hingewiesen werden, dass einige Bewertungskriterien Einschränkungen enthalten, die nur deshalb in das QS-Modell eingefügt wurden, um die Komplexität des Modells nicht ausarten zu lassen. Diese Bewertungskriterien dürfen demnach nicht so streng bewertet werden wie andere, da ihre Einschränkungen in realen Projekten nicht im selben Ausmaß gelten. Als Beispiel dafür wäre das Bewertungskriterium P2 zu nennen, welches angibt, dass die Vorgabedokumente zu 50 % fertig gestellt werden müssen, bevor mit der Nachfolgephase begonnen werden kann. Diese Einschränkung hat zwar positive Auswirkungen auf das Projekt, da es sinnlos

ist zwei Phasen parallel durchzuführen, die vorgegebene Richtlinie von 50 % kann aber nicht direkt in reale Projekte übernommen werden.

## 5.1 Zielvorgaben (Z)

Die Zielvorgaben sind vor allem für den Spieler ein wichtiges Bewertungskriterium. Anhand dieser Daten kann er sofort nach Beendigung des Spiels prüfen, ob er das Projekt erfolgreich zu einem Ende gebracht hat. Auf den positiven Abschluss des Projekts weisen im QS-Modell sechs Zielvorgaben hin.

### 5.1.1 Z1: Dauer des Projekts

Die Dauer gibt an, ob der Spieler die vorgegebene Zeit von neun Monaten überschritten hat, oder ob er das Projekt vor dem geplanten Ende fertig stellen konnte.

### 5.1.2 Z2: Höhe der Projektkosten

Die Kosten zeigen, ob der Spieler das verfügbare Budget von 450 000 DM eingehalten oder überschritten hat.

### 5.1.3 Z3: Realisierte AFPs in Prozent im Code

Eine weitere Zielvorgabe bilden die AFPs für den Code. Der Kunde verlangt, dass 95 Prozent der vereinbarten Adjusted Function Points realisiert werden müssen. Es wird anhand dieser Zielvorgaben geprüft, ob der Spieler diese Grenze erreicht hat.

### 5.1.4 Z4: Enthaltene Fehler pro KLOC

Die Restfehler im Code werden ebenfalls geprüft. Es wird dabei ermittelt, ob das Dokument des Spielers die Grenze von maximal 12 Fehlern pro KLOC überschritten hat.

### 5.1.5 Z5: Realisierte AFPs in Prozent im Handbuch

Dieser Wert zeigt an, ob der Spieler die geforderten 95 Prozent der AFPs im Handbuch realisiert hat.

### 5.1.6 Z6: Enthaltene Fehler pro Seite im Handbuch

Die letzte Zielvorgabe prüft die Fehler pro Seite im Handbuch des Spielers. Der Kunde fordert, dass das Handbuch pro Seite nicht mehr als 0,5 Fehler enthält.

## 5.2 Bewertungskriterien für Mitarbeiter (M)

### 5.2.1 M1: Anzahl der Mitarbeiter pro Phase (gleichzeitig)

Ein interessanter Wert ist die Anzahl der Mitarbeiter, die gleichzeitig an einem Dokument gearbeitet haben. Aufgrund dieses Werts, kann man über den Kommunikationsaufwand Aussagen treffen. Je mehr Mitarbeiter in einer Phase beschäftigt sind, desto höher ist der Kommunikationsaufwand, da die Mitarbeiter ihre Teile miteinander absprechen müssen. Mit steigendem Kommunikationsaufwand kommt es außerdem zu höheren Kosten, da jeder Mitarbeiter für seinen Teil mehr Zeit beansprucht.

### 5.2.2 M2: Anzahl der Mitarbeiter pro Phase (nicht gleichzeitig)

Ebenso interessant ist es zu untersuchen, ob alle Mitarbeiter gleichzeitig zum Beispiel an der Erstellung eines Dokuments beteiligt waren, oder ob der Spieler zu einem späteren Zeitpunkt noch andere Mitarbeiter zu dieser Tätigkeit hinzugezogen hat. Dieses Bewertungskriterium weist auf Probleme im Personaleinsatz hin.

### 5.2.3 M3: Unproduktive Zeit pro Mitarbeiter

Die Aufgabeneinteilung der Mitarbeiter ist eine der schwierigsten Aufgaben im Projektmanagement. Wenn dem Spieler die Anzahl der unproduktiven Tage jedes einzelnen Mitarbeiters und die dadurch aufgetretenen Kosten präsentiert werden, dann kann der Spieler leichter darauf aufmerksam gemacht werden, welche Auswirkungen Lücken im Personaleinsatz haben können. Es ist sinnvoll, zusätzlich die Phasen anzugeben, in denen der Mitarbeiter nicht produktiv war. Möglicherweise gibt es einen Grund dafür, warum er zu einem bestimmten Zeitpunkt im Projekt untätig war. Ein möglicher Grund wäre, dass der Mitarbeiter nicht mit seiner Tätigkeit beginnen konnte, weil die Vorgabedokumente noch nicht ausreichend entwickelt waren. Ein lückenloser Einsatz des Personals während des gesamten Projekts ist nahezu unmöglich. Es kann durchaus vorkommen, dass ein Mitarbeiter ein paar Tage unproduktiv ist, weil der Spieler keine andere Wahl hat. Der Spieler soll darum an

dieser Stelle nur auf größere Lücken im Personaleinsatz und deren Auswirkungen aufmerksam gemacht werden.

### 5.2.4 M4: Qualifikation der Mitarbeiter

Dieses Bewertungskriterium soll prüfen, ob der Spieler die Mitarbeiter ihren Fähigkeiten entsprechend eingesetzt hat. Wird eine Person mit falschen Fähigkeiten für eine Tätigkeit eingesetzt, dann spiegelt sich dieser Fehler in den Dokumenten wider. Bei der Erstellung von Dokumenten führt der Einsatz eines wenig qualifizierten Mitarbeiters zu einer geringeren Vollständigkeit und Korrektheit des Dokuments. Bei Prüfungen und Test hingegen, finden minder qualifizierte Mitarbeiter weniger Fehler im Prüfobjekt. Allerdings muss an dieser Stelle auch erwähnt werden, dass Mitarbeiter während des Projekts auch Erfahrungen sammeln. Wenn der Spieler die gleichen Mitarbeiter mehrmals als Gutachter zu Reviews einlädt, wissen sie bereits, was sie zu tun haben. Dadurch können Prüfungen effektiver durchgeführt werden. Setzt der Spieler bei der anschließenden Korrektur der Dokumente unqualifizierte Personen ein, dann können diese nur wenige der gefundenen Fehler verbessern und fügen zusätzlich auch neue Fehler in das Dokument ein. Die hier angeführten Auswirkungen werden in den einzelnen Bereichen noch genauer beschrieben.

## 5.3 Bewertungskriterien für Dokumente (D)

### 5.3.1 D1: Vollständigkeit der Dokumente

Die Vollständigkeit eines Dokuments gibt an, wie viele AFPs das Dokument enthält. Damit kann festgestellt werden, ob der Spieler das Dokument ausreichend geprüft hat. Stellt man die Vollständigkeit der in den einzelnen Phasen erstellten Dokumente gegenüber, kann man erkennen, ob der Spieler die Dokumente der frühen Phasen (Spezifikation und Entwurf) ausreichend geprüft hat, und ob er die Vollständigkeit bzw. die Korrektheit der Dokumente in den weiteren Phasen beibehalten konnte.

### 5.3.2 D2: Restfehler der Dokumente

Dieser Wert gibt an, wie viele Restfehler ein Dokument nach Abschluss des Projekts noch enthält. Aussagekräftiger wird dieser Wert, wenn man die Restfehler der Dokumente der einzelnen Phasen gegenüberstellt.

Jedes Dokument kann noch genauer untersucht werden, indem man die Restfehler nach ihrer Fehlerart in Analyse-, Grobentwurfs-, Feinentwurfs-, Implementierungs- und Handbuchfehler aufsplittet. Damit kann festgestellt werden, ob sich die Fehler über die Phasen hinweg fortgepflanzt haben, oder ob der Spieler versucht hat, Fehler in den Vorgabedokumenten auch in den Nachfolgedokumenten zu korrigieren.

### *5.3.3 D3: Autoren der Dokumente*

Hierbei muss Rücksicht auf die Anzahl und die Qualifikation der Autoren genommen werden. In M4 wurde bereits festgehalten, dass die Mitarbeiter ihren Fähigkeiten entsprechend eingesetzt werden müssen. Wird ein Dokument von einem Mitarbeiter erstellt, der nicht qualifiziert dafür ist, dann leidet die Korrektheit und die Vollständigkeit darunter, da mehr Fehler in das Dokument eingefügt und einige Anforderungen nicht verwirklicht werden.
Auch die Anzahl der Autoren eines Dokuments kann sich negativ auf dieses auswirken. Da es im QS-Modell nur eine bzw. maximal zwei Personen gibt, die qualifiziert genug sind, um ein Dokument zu erstellen, fügen weitere Mitarbeiter durch ihre Unwissenheit zusätzliche Fehler in das Dokument ein.

### *5.3.4 D4: Kosten für das Dokument (ohne Review/Korrektur)*

Die Kosten für ein Dokument ergeben sich aus der Summe der Kosten für jeden einzelnen Mitarbeiter, der an der Erstellung des Dokuments beteiligt war. Dadurch kann der Spieler ein Gefühl dafür bekommen, wie sich der Einsatz von mehreren Entwicklern auf den Kommunikationsaufwand und weiter auf die Kosten auswirkt.

## *5.4 Bewertungskriterien für Reviews (R)*

Im QS-Modell werden Reviews durchgeführt, um Fehler in Dokumenten aufdecken zu können. Zu einem Review können mehrere Gutachter eingeladen werden, die das Prüfobjekt prüfen müssen. In einer anschließenden Reviewsitzung werden die von den Gutachtern gefundenen Mängel in einem Reviewbericht festgehalten, der später für die Korrektur des Dokuments verwendet wird. Durch einen Review können sowohl fehlende Aspekte aufgedeckt als auch fehlende Funktionalität gefunden werden. Außerdem kann auch ein Teil der im Referenzdokument enthaltenen Fehler erkannt werden.

Im QS-Modell gibt es die Möglichkeit zwei oder drei Gutachter zu einem Review einzuladen. Mitarbeiter können neben ihrer normalen Tätigkeit auch als Gutachter zu Reviews zugeteilt werden. Allerdings geht der Durchführung einer Review-Sitzung eine Einarbeitungszeit der Gutachter voraus. Diese Vorbereitungszeit kostet dem Mitarbeiter ca. 20 % seiner Arbeitszeit. Dadurch reduziert sich der Anteil der verbleibenden Arbeitszeit für die parallel ausgeführte Aktivität auf 80 %.

### *5.4.1 R1: Eingesetztes Reviewteam*

Um eine Reviewsitzung bewerten zu können, müssen mehrere Faktoren geprüft werden. Einer davon ist das Reviewteam, das das Dokument prüft. Wenn das Reviewteam schon mehrfach eingesetzt wurde, wirkt sich das vorteilhaft auf die Effektivität und Effizienz des Reviews aus, denn je häufiger ein Gutachter an einem Review teilnimmt, desto effektiver kann er die Prüfung des Prüfobjekts vornehmen, und desto effizienter kann die Reviewsitzung gestaltet werden.

Der Autor ist an jedem Review seines Dokuments beteilig, er sollte allerdings immer eine passive Haltung während der Reviewsitzung einnehmen. Nimmt ein Autor als Gutachter an einem Review teil, dann kann sich dies negativ auf das Prüfobjekt auswirken, da es eher unwahrscheinlich ist, dass der Autor bei der Reviewsitzung auf Fehler in seinem eigenen Dokument hinweist, sofern er sie selbst entdeckt, denn er wird kaum seine eigenen Fehler zugeben wollen. Im QS-Modell hat man versucht diesen Effekt zu realisieren. Setzt der Spieler den Autor als Gutachter für sein Dokument ein, dann hat dies zur Folge, dass der Autor keine (oder kaum) Fehler findet. Dadurch werden im Review insgesamt weniger Fehler gefunden und die Korrektheit bzw. Vollständigkeit des Prüfobjekts leidet darunter.

### *5.4.2 R2: Anzahl der beteiligten Gutachter*

Dieses Bewertungskriterium prüft, wie viele Gutachter für einen bestimmten Review eingesetzt wurden. Im QS-Modell ist es möglich zwei oder drei Gutachter zu einem Review einzuladen. Wie wir bereits im Kapitel 4 gesehen haben, finden drei Gutachter nur unwesentlich mehr Fehler als zwei Gutachter. Das liegt daran, dass es viele Fehler gibt, die alle Gutachter entdecken und jeder Gutachter nur ein paar neue Mängel im Dokument finden kann. Jeder zusätzliche Gutachter verursacht allerdings zusätzlichen Aufwand und damit auch höhere Kosten.

Je größer die Zeitspanne zwischen dem Einfügen eines Fehlers und dem Entfernen ist, desto höher sind die Fehlerkosten. Dadurch kann ein höherer Reviewaufwand in den frühen Phasen gerechtfertigt sein.

### *5.4.3 R3: Erfolgreiche Durchführung der Korrekturphase*

Wurde ein Review durchgeführt, so muss überprüft werden, ob die gefundenen Fehler auch im Anschluss verbessert wurden. In den Testspielen, die im SS 2002 durchgeführt wurden, konnte festgestellt werden, dass viele Spieler auf die Korrekturphase nach einem Review vergessen, oder diese Phase aus Kostengründen nicht mehr durchführen.

### *5.4.4 R4: Korrektur durch den Autor*

Wichtig für die Korrektur eines Dokuments ist, dass die Fehler von einem qualifizierten Mitarbeiter verbessert werden. Der geeignetste Mitarbeiter dafür ist verständlicherweise der Autor des Dokuments. Er kennt sich in seinem Dokument am besten aus und kann dadurch die Fehlerstellen schnell entdecken und die Fehler rasch korrigieren. Außerdem kann angenommen werden, dass er bei der Korrektur weniger neue Fehler in das Dokument einfügt.

### *5.4.5 R5: Dauer von Reviewphasen*

Zieht sich die Reviewphase über mehrere Monate, dann muss genauer untersucht werden, wie viele Reviews es gegeben hat, und ob die Reviews gleich hintereinander durchgeführt wurden.

Hat es mehr als drei Reviews gegeben, so muss weiter untersucht werden, wie viele Fehler in den einzelnen Reviews gefunden wurden. Es ist eher unwahrscheinlich, dass sich der Aufwand, bei gegebener Entwicklerqualität, ab dem dritten Review noch lohnt, denn es werden in jedem Review weniger Fehler gefunden. Der Aufwand für die Reviews und die damit verbundenen Kosten stehen somit in keiner Relation zu den gefundenen Fehlern.

### *5.4.6 R6: Effizienz von Reviews*

Um die Effizienz eines Reviews zu messen, wird der Aufwand für den Review den gefundenen Fehlern gegenübergestellt. Schlechte bzw. gute Ergebnisse können auf die Anzahl der Gutachter zurückgeführt werden. Werden zu viele Gutachter eingesetzt, dann steigt der Aufwand und somit auch die Kosten.

### 5.4.7 R7: Effektivität von Reviews

Um die Effektivität eines Reviews zu messen, werden die gefundenen Fehler den im Dokument enthaltenen Fehlern gegenübergestellt. Schlechte bzw. gute Ergebnisse können auf die Qualifikation der Gutachter zurückgeführt werden.

### 5.4.8 R8: Anzahl erfolgreicher/nicht erfolgreicher Reviews

Durch diese absoluten Werte kann festgestellt werden, ob der Spieler viele Reviews durchgeführt hat. Hat es viele nicht beendete Reviews gegeben, dann kann der Tutor durch genaueres Untersuchen der Projektverlaufsdiagramme („Spezifikation und Entwurf“, „Modulspezifikation und Codierung“ und „Handbuch und Integration“) feststellen warum, und dem Spieler Tipps für die nächsten Spiele geben.

Zum Abbruch eines Reviews kommt es zum einen, wenn der Spieler selbst den Review abbricht, zum anderen wenn Vorgabedokumente noch nicht zu 50 % fertig gestellt wurden, oder im dritten Fall, wenn die Korrektur eines früheren Reviews dieses Dokuments noch nicht abgeschlossen wurde. Im zweiten Fall gibt es nichts was von den Gutachtern überprüft werden könnte, das bedeutet, die vom Spieler eingestellten Gutachter beenden den Review relativ schnell, ohne das sie Fehler gefunden haben. Dies führt häufig dazu, dass der Spieler ein neues Review durchführt, welches wieder nicht erfolgreich ist. Er versucht es dann solange bis das Referenzdokument zu 50 % erstellt wurde und der Review durchgeführt werden kann. Dies führt zu hohen Kosten, die völlig unnötig entstanden sind.

### 5.4.9 R9: Verluste durch Reviews (AFPs)

Die Verluste in Reviews werden in AFPs gemessen. Dieser Wert sagt aus, wie viele AFPs während eines Reviews verloren gingen. Der Verlust kann häufig auf ein schlecht gewähltes Reviewteam zurückgeführt werden.

## 5.5 Bewertungskriterien für Tests (T)

### 5.5.1 T1: Tester

Um eine Bewertung der Effizienz eines Tests durchführen zu können, ist es wichtig zu wissen wer das Testdokument verfasst hat, und wer den Test für dieses Dokument geschrieben hat. Bei den Tests gelten nicht dieselben Einschränkungen wie bei Reviews. Der Test darf auch vom Autor durchgeführt

werden. Allerdings muss auch hier geprüft werden, ob der Tester die nötigen Fähigkeiten für diese Tätigkeit mitbringt. Je öfter ein Mitarbeiter zum Testen eingesetzt wird, desto effizienter und effektiver kann er den Test vorbereiten und durchführen.

### *5.5.2 T2: Korrektur von Tests*

Auch bei der Korrektur aufgrund eines Tests ist darauf zu achten, dass der Autor sein Dokument selbst korrigiert. Es hat sich gezeigt, dass dies eine positive Wirkung auf den Aufwand sowie auf die effiziente Durchführung der Korrektur hat. Der Autor fühlt sich verantwortlich für sein Dokument, er kennt sich am besten in seinem Dokument aus und es ist anzunehmen, dass der Autor bei der Korrektur weniger zusätzliche Fehler in das Dokument einfügt.

Der Korrekturaufwand wird durch die Qualifikation des Entwicklers bestimmt. Der Mitarbeiter, der den Code geschrieben hat, kennt ihn auch am besten und hat dadurch keinen Einarbeitungsaufwand.

### *5.5.3 T3: Verluste durch Tests (AFPs)*

Die Verluste in Tests werden in AFPs gemessen. Dieser Wert sagt aus, wie viele AFPs während eines Tests verloren gingen. Diese Verluste weisen auf schlecht gewählte Tester hin.

### *5.5.4 T4: Effizienz von Tests*

Um die Effizienz eines Tests zu messen, wird der Aufwand für den Test den gefundenen Fehlern gegenübergestellt. Schlechte bzw. gute Ergebnisse können auf die Anzahl der Tester zurückgeführt werden. Werden zu viele Tester eingesetzt, dann steigt der Aufwand und somit auch die Kosten.

### *5.5.5 T5: Effektivität von Tests*

Um die Effektivität eines Tests zu messen, werden die gefundenen Fehler den im Dokument enthaltenen Fehlern gegenübergestellt. Schlechte bzw. gute Ergebnisse können auf die Qualifikation der Tester zurückgeführt werden.

## 5.6 Bewertungskriterien für Phasen (P)

### 5.6.1 P1: Parallelitäten

Es sollte überprüft werden, ob bestimmte Phasen parallel durchgeführt wurden, da zum Beispiel ein paralleles Ausführen von Spezifikationsreview und Entwurf dazu führt, das die Fehler, die in dem Review entdeckt werden, noch im Entwurf enthalten sind. In diesem Fall sollte die Korrektur der Mängel auch in den Nachfolgedokumenten nachgezogen werden. Es konnte festgestellt werden, dass diese Änderungen im Nachfolgedokument zu einem höheren Aufwand bei Reviews bzw. Korrektur führen und deshalb natürlich auch zu höheren Kosten. Parallelitäten von Phasen führen häufig zu höheren Restfehlern in den Dokumenten, da die Spieler die Korrektur der Dokumente nicht nachziehen.

### 5.6.2 P2: Vorgabedokument bereits zu 50 Prozent erstellt

Es ist möglich Aktivitäten parallel durchzuführen, allerdings muss auf die Vollständigkeit des Referenzdokuments geachtet werden. Erst wenn das Vorgabedokument zu 50 % fertig gestellt wurde kann mit der nächsten Phase begonnen werden. Sollte der Spieler seinem Mitarbeiter bereits davor die nächste Aktivität zuweisen, so wird der Mitarbeiter diesen Auftrag ignorieren. Bei diesem Bewertungskriterium handelt es sich um eine reine Modelleigenschaft, die realisiert wurde, um die Komplexität des Modells in den Griff zu bekommen.

### 5.6.3 P3: Beginn der Nachfolgephase erst nach Korrektur der Vorgängerphase

Die Korrektur der Vorgabedokumente sollte erfolgreich abgeschlossen werden, bevor das Dokument für die nächst Phase verwendet wird. Andernfalls werden viele Fehler übernommen, die im Review des Vorgabedokuments bereits gefunden, aber noch nicht verbessert wurden. Sollte darauf nicht geachtet werden, dann muss geprüft werden, ob der Spieler die Folgedokumente nachgezogen hat, um die Fehler im Nachfolgedokument zu entfernen und die fehlenden Anforderungen zu ergänzen.

### 5.6.4 P4: Modultest erst nach Codereview

Im Modultest werden einerseits Fehler identifiziert, die während der Implementierung eingefügt wurden, andererseits wird geprüft, ob die Vorgaben

aus der Modulspezifikation im Code erfüllt wurden. Fehler in der Modulspezifikation werden nicht durch Modultests gefunden, allerdings werden einige Mängel durch die Beschäftigung mit dem Dokument entdeckt.

### *5.6.5 P5: Integrationstest erst nach Modulspezifikation-Review*

Im Integrationstest werden Fehler gefunden, die während des Feinentwurfs entstanden sind. Es wird überprüft, ob die Subsysteme ihre Funktion erfüllen, und ob die Module fehlerfrei zusammenspielen.

### *5.6.6 P6: Aufwandsverteilung*

Die Aufwandsverteilung zeigt, wie viel Aufwand in die einzelnen Phasen investiert wurde. Dabei muss vor allem untersucht werden, ob der Aufwand für die Spezifikation und für den Entwurf nicht zu gering ausgefallen ist bzw. der Aufwand für die Testphase im Vergleich zu den anderen Phasen zu groß war. Dadurch kann festgestellt werden, ob der Spieler bereits einen höheren Aufwand in die frühen Phasen (Spezifikation und Entwurf) investiert hat und dadurch nur geringer Testaufwand notwendig war, oder ob er die frühen Dokumente nicht sorgfältig genug geprüft hat und somit durch intensives Testen die Fehler zu einem späteren Zeitpunkt in den Dokumenten entdecken und entfernen musste.

## *5.7 Bewertungskriterien für den Kundeneinsatz (K)*

Der Kontakt zum Kunden ist für den Erfolg von Softwareprojekten entscheidend. Eine intensive und regelmäßige Kommunikation mit dem Auftraggeber und den Benutzern hat in der Regel positive Auswirkungen auf den Projektverlauf [Mandl-Striegnitzb]. Da Projektleiter realer Projekte Defizite auf diesem Gebiet aufweisen, wurde dieser Effekt auch im QS-Modell realisiert. Die Anwesenheit des Kunden an Reviews führt im QS-Modell dazu, dass eine Vielzahl zusätzlicher Mängel gefunden wird, die von den internen Gutachtern nicht entdeckt werden können.

### *5.7.1 K1: Teilnahme des Kunden an erfolgreichen Spezikationsreviews*

Bei der Prüfung der Spezifikation kann der Kunde fehlende Funktionalität aufdecken, die zu einem frühen Zeitpunkt noch recht günstig im Projekt realisiert werden kann. Der Kunde findet zusätzlich Analysefehler, die auf-

grund fehlerhafter Analysenotizen in das Spezifikationsdokument übernommen wurden.

### 5.7.2 K2: Teilnahme des Kunden an erfolgreichen Handbuchreviews

Prüft der Kunde auch das Benutzerhandbuch, das recht einfach die Funktionalität des Systems beschreibt, kann er dabei Mängel finden, die die Spezifikation betreffen. Deshalb sollte auch das Handbuch zu einem frühen Zeitpunkt geprüft werden, damit eventuelle Fehler und fehlende Funktionalität noch in der Spezifikation korrigiert bzw. hinzugefügt werden können.

### 5.7.3 K3: Erfolgreicher Handbuchreview zu einem frühen Zeitpunkt

Da vom Kunden durch eine Prüfung des Handbuchs Analysefehler gefunden werden können, sollte diese Prüfung zu einem frühen Zeitpunkt im Projekt durchgeführt werden, denn je weiter das Projekt fortgeschritten ist, desto teurer wird die Beseitigung der Fehler.

## 5.8 Bewertungskriterien für die Projektplanung (PP)

### 5.8.1 PP1: Anzahl der Inspektionen der Tätigkeiten der Entwickler

Dieser absolute Wert gibt an, wie oft der Spieler sich nach dem Zustand der einzelnen Dokumente erkundigt hat. Nehmen wir an, ein Spieler hat häufig Inspektionen durchgeführt, dann könnte dies möglicherweise auf Parallelitäten in seinem Spielverlauf zurückzuführen sein. Das bedeutet der Spieler hat sich erkundigt, ob das Vorgabedokument bereits zu 50 % fertig gestellt wurde, bevor er seine Mitarbeiter für die Erstellung des Dokuments der nächsten Phase einsetzt.

### 5.8.2 PP2: Anzahl der Inspektionen der verbrauchten Ressourcen

Dieser absolute Wert gibt an, wie oft sich der Spieler nach seinem verbrauchten Budget erkundigt hat. Auch aufgrund dieses Wertes können Schlüsse über geplanten bzw. chaotischen Projektverlauf gezogen werden.

Die Bewertungskriterien für die Projektplanung können allerdings nur Anhaltspunkte liefern. Um genauere Aussagen über die Projektplanung treffen zu können, muss der Projektplan analysiert werden. Da das Erstellen und Führen eines Projektplans für die SESAM-Spiele nicht verpflichtend ist, können Probleme der Studenten in diesem Bereich nicht aufgedeckt und behoben

werden. Verpflichten sich die Studierenden hingegen dazu einen Projektplan zu führen und diesen nach dem Spiel dem Lehrveranstaltungsleiter auszuhändigen, nimmt die Anzahl der scheiternden Projekte ab. Auch [NOTTER] weist darauf hin, dass das Erstellen und Führen eines Projektplans verpflichtend sein sollte, da viele Projekte nur aufgrund einer chaotischen Projektplanung scheiterten.

Wir haben nun gesehen, anhand welcher Kriterien eine Auswertung von Spielverläufen vorgenommen werden kann. Der Tutor hatte bis jetzt die Aufgabe die einzelnen Spielverläufe aufgrund dieser Kriterien genauer zu analysieren, um den Spielern möglichst viele Fehler im Projektmanagement aufzeigen zu können. Diese zeitraubende Aufgabe soll nun von einer generischen Erklärungskomponente übernommen werden. Welche Architekturüberlegungen bzw. Designentscheidungen bei der Entwicklung dieser Erklärungskomponente angestellt wurden, wird im nächsten Kapitel beschrieben.

*Kapitel 6*

# ENTWICKLUNG EINER GENERISCHEN ERKLÄRUNGSKOMPONENTE

Das bereits entwickelte Auswertungswerkzeug „SesamScore“, welches in Kapitel 3 kurz besprochen wurden, wird in diesem Kapitel etwas genauer untersucht, um zu zeigen, warum die Entwicklung eines neuen Werkzeugs notwendig war. Die neue Erklärungskomponente hat die Aufgabe, Spielverläufe, anhand der in Kapitel 5 identifizierten Bewertungskriterien, zu analysieren, um den Tutor zu entlasten und den Einsatz des AMEISE-System für ein Selbststudium zu ermöglichen. In diesem Kapitel werden deshalb Architekturüberlegungen und Designentscheidungen vorgestellt, die bei der Entwicklung der generischen Erklärungskomponente vorgenommen wurden. Außerdem werden mögliche Ansätze zur Realisierung einer solchen Komponente diskutiert.

## *6.1 Simulationsdateien*

Damit die Auswertung eines Projektverlaufs möglich wird, werden Informationen über das Spiel benötigt. Diese Informationen sind jedoch in verschiedenen Simulationsdateien versteckt, die während der Simulation erzeugt werden. Zu diesen Simulationsdateien zählen:

- Kommandodatei
- Situationsdatei
- Protokolldatei

Relevant sind allerdings nur die Kommando- und die Situationsdatei, da sie alle Informationen enthalten, die für die Analyse von Spielverläufen benötigt werden.
In den folgenden Abschnitten werden die einzelnen Simulationsdateien genauer betrachtet.

### 6.1.1 Die Kommandodatei

Die Kommandodatei mit der Erweiterung „prot“ hat eine ungefähre Größe von 80 KB und enthält alle Kommandos, die der Spieler während der Simulation abgesetzt hat sowie das Datum zu dem das Kommando erfolgt ist. Jedes natürlichsprachliche Kommando, das vom Spieler abgesetzt wird, wird in dieser Datei gespeichert. Sie ist allerdings für die anschließende Auswertung des Projektverlaufs nicht von Bedeutung und wird hier nur der Vollständigkeit halber angeführt.

### 6.1.2 Die Situationsdatei

Die zweite Datei, welche vom Simulator exportiert wird, trägt die Erweiterung „sit“ und entspricht der Situationsdatei. Sie weist eine ungefähre Größe von 60 KB auf. Die Situationsdatei enthält den gesamten Zustand des Projekts zum aktuellen Zeitpunkt, inklusive aller Relationen, Entitäten und deren Attributwerten. Ein Auszug daraus ist in der Abbildung 6.1 zu sehen.

Nach jedem „Proceed“ des Spielers wird die Situationsdatei überschrieben, da bei diesem Befehl der Zustand des Projekts aktualisiert wird. Führt der Spieler ein „Proceed“ aus, dann werden alle Kommandos, die zwischen dem aktuellen und dem letzten „Proceed“ abgesetzt wurden, verarbeitet. Die Simulation schaltet einen Tag weiter und das Projekt befindet sich in einem neuen Zustand, da sich bei der Verarbeitung der Kommandos der aktuelle Zustand der simulierten Dokumente ändert, Beziehungen neu erzeugt werden bzw. Beziehungen aufhören zu existieren.

Die Abbildung 6.1 zeigt einen Auszug aus einer Situationsdatei. Den ersten beiden Zeilen kann entnommen werden, dass es sich um ein Spiel mit dem QS-Modell handelt und dass diese Datei den Zustand des Projekts vom 18.04.2000 (Simulationsdatum) widerspiegelt. In dieser Datei sind sämtliche Dokumente und ihre Attributwerte zum aktuellen Zeitpunkt enthalten.
Außerdem zeigt der Ausschnitt in Abbildung 6.1 die Projektdaten zum aktuellen Zeitpunkt. Anhand des Attributs „KOSTEN“ kann festgestellt werden, dass der Spieler bereits 415 560 DM des Budgets verbraucht hat. Weiter unten in der Attributliste findet man das Datum des Projektbeginns sowie des vorläufigen Projektendes. Dem Attribut „PROJEKT_IST_BEENDET = false“ kann entnommen werden, dass das Projekt noch nicht beendet wurde. Der Eintrag „SYSTEM_AUSGELIEFERT = true“ weist darauf hin, dass der Spieler das System bereits an den Kunden übergeben hat und dass dieser im Moment mit der Prüfung des ausgelieferten Systems beschäftigt ist.

```
model QS_SAVE is
begin at 2000/04/18/08:00 using "model.dib" seed "48979510, 43603650, 94833359, 47416679"
...
         create entity SPEZIFIKATIONSREVIEWBERICHT62:SPEZIFIKATIONSREVIEWBERICHT
                aka "Specificationreviewreport" with
                 INHALT := BAG_RECORD_IRRRRRRRR_TYPE'();
                 NICHT_KORRIGIERTE_IDS := BAG_I_TYPE'();
                 FEHLER := 73.5136;
                 VERLUSTE := 5.4048;
                 IST_LEER := true;
                 ANZ_ANF := 0.0;
                 ANZ_NICHT_KORR_ANF := 0.0;
                 NUMMER := 2;
                 ANZ_FEHLER_PRO_PRF := ARRAY(73.5136,0.0,0.0,0.0,0.0);
                 ANZ_KORR_FEHLER_PRO_PRF := ARRAY(67.6326,0.0,0.0,0.0,0.0,0.0);
         end create;
         create entity PROJEKTZUSTAND58: PROJEKTZUSTAND
                aka "Projectstatus"' with
                 KOSTEN := 415560.0;
                 AUFWAND_SPEZIFIKATIONSPHASE := 179.478;
                 AUFWAND_ENTWURFSPHASE := 488.428;
                 AUFWAND_CODIERUNGSPHASE := 855.606;
                 AUFWAND_TESTPHASE := 584.0;
                 AUFWAND_HANDBUCHERSTELLUNG := 183.987;
                 AUFWAND_REVIEWS := 77.1168;
                 AUFWAND_TESTS := 288.0;
                 AUFWAND_KORREKTUR_REVIEWS := 80.0;
                 AUFWAND_KORREKTUR_TESTS := 296.0;
                 AUFWAND_FUER_KUNDE := 735.567;
                 BEHOBENE_FEHLER := 0.0;
                 PROJEKTBEGINN := 1999/08/02/08:00;
                 PROJEKTENDE := 2000/04/17/08:00;
                 PROJEKT_IST_BEENDET := false;
                 SYSTEM_AUSGELIEFERT := true;
                 ANZAHL_ABNAHMETESTS := 2;
                 GESAMTAUFWAND := 17.3598;
                 KOSTEN_PRO_MM := 23938.0;
                 AUFWANDSVERTEILUNG_PHASEN := ARRAY_5_R_TYPE'
                 (7.83235,21.3148,37.3383,25.4855,8.0291);
                 MITTELWERT_VOLLSTAENDIGKEIT := 95.1275;
                 MITTELWERT_VOLLSTAENDIGKEIT_ZV := 98.5687;
                 MITTELWERT_KORREKTHEIT_ZV := 90.0147;
                 PROZENT_DAUER_ZV := 104.074;
                 PROZENT_KOSTEN_ZV := 107.653;
         end create;
...
```

*Abb. 6.1:* Auszug aus einer Situationsdatei

### 6.1.3 Die Protokolldatei

Die dritte Datei, die während der Simulation durch den Simulator erzeugt wird, ist die Datei mit der Erweiterung „sit.prot“ (Abbildung 6.2). Sie hat eine ungefähre Größe von 20 KB. In ihr werden alle Aktionen des Benutzers abgespeichert. Diese Datei ermöglicht eine Rekonstruktion des gesamten Spielverlaufs. Sie enthält nicht nur den Namen des Modells, welches gespielt wurde, sondern auch alle Kommandos, die der Spieler abgesetzt hat. Die Kommandos werden hier aber nicht in natürlichsprachlicher Form, wie sie der Dolmetscher[1] erhält, und wie sie in der Kommandodatei festgehalten werden, mitprotokolliert, sondern in der Form, die das Modell versteht.

Abbildung 6.2 stellt einen Auszug aus einer Protokolldatei dar. Die erste Zeile enthält Informationen über das Modell, das geladen bzw. gespielt wurde. Im Beispiel in Abbildung 6.2 handelt es sich um das QS-Modell. Danach folgt eine Liste von Kommandos, die der Spieler abgesetzt hat. Die Befehle werden hier nicht in natürlichsprachlicher Form angegeben, sondern bereits in der Art und Weise, wie sie das Modell versteht. Das bedeutet, der Spieler setzt das Kommando „hire Richard“ ab und der Dolmetscher übersetzt es in den Befehl „TEILE_DEM_PROJEKT_ZU(Richard), damit dieser vom Modell verarbeitet werden kann.

Nach genauerer Betrachtung der Inhalte der einzelnen Dateien kann man erkennen, dass die Situationsdatei (sit) und die Protokolldatei (sit.prot) eng zusammenhängen. Die erste enthält den Zustand, die zweite beschreibt, wie man zu dem Zustand gelangt.

## 6.2 Genauere Betrachtung verschiedener Ansätze

Im Kapitel 4 wurden die Auswertungsdiagramme mit Hilfe von SesamScore untersucht. Auf den ersten Blick würde man meinen, dass SesamScore schon

[1] Der Dolmetscher ermöglicht die Kommunikation zwischen Spieler und Simulator. Er verwendet dazu ein Wörterbuch, in dem alle Benutzerkommandos und Nachrichten definiert sind. Mit Hilfe dieses Wörterbuchs können alle Befehle, die der Spieler natürlichsprachlich eingibt, vom Dolmetscher übersetzt und in Kommandos des Simulators transformiert werden. Der Simulator verarbeitet das Kommando und erzeugt eine Nachricht, die anschließend vom Dolmetscher unter Zuhilfenahme des Wörterbuchs in einen natürlichsprachlichen Text übersetzt wird, bevor sie der Spieler erhält.
Der Dolmetscher wurde im Rahmen der Diplomarbeit von Andre Spiegel [Spielgel] entwickelt und in das damalige SESAM-1-System integriert. Die spätere Anpassung des Dolmetschers an das SESAM-2-System wurde im Dokument „Feinentwurf Teilsystem Dolmetscher“ [Rohrbach] festgehalten.

```
load_test_model/home/projects/susi/Spiele/qs_exp_200_pms_info_V2.b2
proceed  1 0 0 0
execute_command "TEILE_DEM_PROJEKT_ZU(Richard)" 0 0 0
execute_command "LASSE_MIT_KUNDEN_SPRECHEN(Richard)" 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
execute_command "LASSE_SPEZIFIZIEREN(Richard)" 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
execute_command "TEILE_DEM_PROJEKT_ZU(Bernd)" 0 0 0
execute_command "LASSE_REVIEWS_STATTFINDEN(Analysis, Bernd, Richard)" 0 0 0
proceed  1 0 0 0
execute_command "TEILE_DEM_PROJEKT_ZU(Christine)" 0 0 0
execute_command "LASSE_SPEZREVIEWS_MIT_KUNDE_STATTFINDEN(Bernd, Christine)" 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
execute_command "LASSE_SPEZIFIKATION_NACH_REVIEW_KORRIGIEREN(Richard)" 0 0 0
...
```

*Abb. 6.2:* Auszug aus der Protokolldatei

alles kann, was eine automatische Auswertung können muss, denn es kann ausgewählte Daten anschaulich als Diagramm darstellen. Für eine Erklärungskomponente würde es genügen mittels SesamScore Diagramme zu erzeugen und diese mit Erklärungstexten zu verknüpfen, die dem Spieler erklären, was das Diagramm zeigt bzw. welche Fehler der Spieler während des Spiels gemacht hat, und wie diese Fehler im Diagramm sichtbar werden. Bei genauerer Betrachtung, und die soll im folgenden Abschnitt vorgenommen werden, werden viele Probleme sichtbar, die die Entwicklung eines neuen Auswertungswerkzeugs verlangen. Im folgenden Abschnitt werden mögliche Ansätze beschrieben und gegenübergestellt.

### *6.2.1 Verwendung von SesamScore*

Nehmen wir an, wir haben ein SESAM-Spiel beendet. Uns stehen also alle drei Dateien (Kommandodatei, Situationsdatei und Protokolldatei) zur

```
1999/08/02/08:00
PRINT_DATE (1999/08/02/08:00)
AUFGABENSTELLUNG_BESCHREIBEN (200.88,2000/04/28/08:00,450000.0)
MINDESTENS_PROZENT_AFP (Code,95.0)
MAX_FEHLER_IM_CODE_PRO_KLOC (12.0)
MINDESTENS_PROZENT_AFP (Manuals,95.0)
MAX_FEHLER_PRO_SEITE (Manuals,0.5)

1999/08/03/08:00
TEILE_DEM_PROJEKT_ZU(Richard)
ENTWICKLER_WURDE_EINGESTELLT (Richard)

1999/08/03/08:00
LASSE_MIT_KUNDEN_SPRECHEN(Richard)

1999/08/03/08:00
PRINT_DATE (1999/08/03/08:00)
BEGONNEN_MIT_KUNDE_ZU_SPRECHEN (Richard)

1999/08/04/08:00
PRINT_DATE (1999/08/04/08:00)

AUFGEHOERT_MIT_KUNDE_ZU_SPRECHEN (Richard)

1999/08/05/08:00
LASSE_SPEZIFIZIEREN(Richard)

1999/08/05/08:00
PRINT_DATE (1999/08/05/08:00)
BEGONNEN_ZU_SPEZIFIZIEREN (Richard)
...
```

*Abb. 6.3:* Auszug aus der Analysedatei (Teil 1)

Verfügung. Nun wollen wir den Spielverlauf unseres Spiels mit SesamScore genauer untersuchen. Dann müssen wir leider feststellen, dass das Auswertungswerkzeug SesamScore eine bestimmte Analysedatei mit der Erweiterung „ist“ benötigt, um ausgewählte Daten als Diagramm anzeigen zu können.

Auszüge aus der Analysedatei zeigen die Abbildungen 6.3 und 6.4. Im ersten Block in Abbildung 6.3 werden die geforderten Zielvorgaben angegeben. Das Projekt umfasst demnach 200 AFPs, die bis zum 28.04.2000 realisiert werden müssen. Dabei darf der Spieler das zur Verfügung gestellte Budget von 450 000 DM nicht überschreiten. Weitere Zielvorgaben für den Code und das Handbuch folgen. Danach enthält die Analysedatei nur noch die Einträge, die angeben wann der Spieler welche Kommandos abgesetzt hat. „Proceeds“ werden nicht mehr angeführt, da sie für die Auswertung mittels

```
...
2000/03/20/07:00
GIB_RESULTATE_SPEZIFIKATION
ZUSTAND_VON_SPEZIFIKATION (79.3,1999/08/05/08:00,1999/08/19/08:00,0)
ANZAHL_AFP_KUNDENWUNSCH (200.88)
ANZAHL_AFP_ANALYSE (200.88)
ZUSTAND_VON_SPEZ_DOK (198.03,55,55)
ZUSTAND_FEHLER_PRO_AFP_SPEZ_DOK (0.28,0.28,0.63)
EINHALTUNG_ZV_DOK (Specification,98.58,-1.0,-1.0)

2000/03/20/07:00
GIB_RESULTATE_ENTWURF
ZUSTAND_VON_ENTWURF (186.6,1999/09/13/08:00,1999/09/29/08:00,0)
ANZAHL_AFP_KUNDENWUNSCH (200.88)
ANZAHL_AFP_ANALYSE (200.88)
ANZAHL_AFP_SPEZIFIKATION (198.03)
ZUSTAND_VON_ENTW_DOK (195.62,47,26,73)
ZUSTAND_FEHLER_PRO_AFP_ENTW_DOK (195.62,0.13,0.37,0.85)
EINHALTUNG_ZV_DOK (Systemdesign,97.38,-1.0,-1.0)

...

2000/03/20/07:00
GIB_RESULTATE_SREVIEW
ZUSTAND_VON_SREVIEW (41.4,1999/08/24/08:00,1999/09/08/08:00,0)
GEPRUEFTE_AFP_SREVIEW (0.0)
BEFUNDE_IN_PRUEFBERICHT (Specificationreviewreport,66,5.5)
ZUSTAND_PRUEFUNGEN (Specificationreviewreport,0,1,65.99,0.0,0.0, 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)

2000/03/20/07:00
GIB_RESULTATE_SREVIEW_K
ZUSTAND_VON_SREVIEW_K (8.4,1999/08/27/08:00,1999/09/09/08:00,0)
ZUSTAND_VON_SPEZ_DOK (198.03,55,55)
ZUSTAND_KORREKTUR (Specificationreviewreport,true,0,60.71,0.0,0.0, 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)
...
```

*Abb. 6.4:* Auszug aus der Analysedatei (Teil 2)

SesamScore nicht mehr relevant sind. Am Ende der Analysedatei (Abbildung 6.4) werden die aktuellen Zustände der einzelnen Dokumente (Spezifikation, Entwurf, Spezifikation-Reviewbericht, Spezifikation-Reviewkorrektur, usw.) aufgelistet. In dieser Datei sind nun alle Daten in einer Art und Weise aufbereitet, dass sie von SesamScore ausgewertet werden können.

Für die Erklärungskomponente würde das bedeuten, dass sie zuerst eine Analysedatei erzeugen muss, bevor sie mit Hilfe des Werkzeuges „SesamScore“ eine Auswertung des Spielverlaufs vornehmen kann.
Die Analysedatei kann durch Nachspielen (der Vorgang wird im Anschluss genauer beschrieben) des gesamten Spiels erstellt werden, allerdings sind da-

für zusätzliche Eingriffe in die Protokolldatei notwendig, auf die im Weiteren kurz eingegangen wird.

Wie wir bereits festgestellt haben, enthält die Protokolldatei alle Informationen, die für eine Rekonstruktion des gesamten Spiels notwendig sind.
Der Unterschied zwischen der Protokolldatei und der Analysedatei liegt nur darin, dass die Protokolldatei die Aktionen des Spielers speichert, während die Analysedatei die Kommunikation zwischen Spieler und Simulator mitschneidet.

Um eine Auswertung zu ermöglichen, müssen zusätzliche Kommandos in die Protokolldatei (vor dem Befehl „ÜBERGIB_SYSTEM_AN_KUNDEN“) eingefügt werden. Die zusätzlichen Kommandos werden in Abbildung 6.5 aufgelistet.
Betrachtet man zum Beispiel die dritte Zeile genauer, so kann man erkennen, dass der Befehl „GIB_RESULTAT_SPEZIFIKATION“ zur strukturierten Auflistung aller Daten des Spezifikationsdokuments in der Analysedatei führt, wie dem ersten Block der Abbildung 6.4 entnommen werden kann. Auf diese Weise wird der genaue Zustand aller Dokumente ermittelt.
Die Analysedatei wird schließlich durch das Kommando „create_test_output“, wie in der letzten Zeile der Abbildung 6.5 ersichtlich, erzeugt.

Wenn die Protokolldatei um die zusätzlichen Befehle erweitert wurde, muss lediglich noch die erste Zeile editiert werden, die den Pfad zum Modell enthält (zu sehen in Abbildung 6.2), dass geladen bzw. gespielt werden soll, denn für das Nachspielen darf nicht das klassische QS-Modell verwendet werden, da es die neu hinzugefügten Kommandos nicht versteht. Stattdessen muss ein spezielles QS-Modell geladen werden, das wir im Weiteren als QS-Expertenmodell bezeichnen wollen, dass die neue Kommandos (GIB_RESULTAT_ANALYSE, GIB_RESULTAT_SPEZIFIKATION, usw.) problemlos verarbeiten kann.

Anschließend kann das gesamte Spiel anhand der erweiterten Protokolldatei nachgespielt werden. Das Nachspielen muss nicht von Hand vorgenommen werden, sondern wird von dem Programm „Basetest“ durchgeführt, das nichts anderes tut, als die erweiterte Protokolldatei Schritt für Schritt zu durchlaufen und jedes Kommando nochmals am Dolmetscher vorbei am Simulator ausführt. Das Programm benötigt für das Nachspielen des gesamten Spiels nur mehr einen Bruchteil der Zeit, die der Spieler dafür benötigt hat.

Nach dem Nachspielen des Projekts, erhält man eine Analysedatei, deren Daten mit SesamScore analysiert werden können. Für die Analyse der Spiel-

```
# Informationen zu den verschiedenen Produktionsphasen und Dokumente
execute_command "GIB_RESULTATE_ANALYSE" 0 0 0
execute_command "GIB_RESULTATE_SPEZIFIKATION" 0 0 0
execute_command "GIB_RESULTATE_ENTWURF" 0 0 0
execute_command "GIB_RESULTATE_MODSPEZ" 0 0 0
execute_command "GIB_RESULTATE_CODE" 0 0 0
execute_command "GIB_RESULTATE_HANDBUCH" 0 0 0
execute_command "GIB_RESULTATE_INTEGRATION" 0 0 0

# Informationen zu den verschiedenen Test- und Korrekturaktivitaeten
execute_command "GIB_RESULTATE_MTEST" 0 0 0
execute_command "GIB_RESULTATE_MTEST_K" 0 0 0
execute_command "GIB_RESULTATE_ITEST" 0 0 0
execute_command "GIB_RESULTATE_ITEST_K" 0 0 0
execute_command "GIB_RESULTATE_STEST" 0 0 0
execute_command "GIB_RESULTATE_STEST_K" 0 0 0
execute_command "GIB_RESULTATE_ATEST" 0 0 0
execute_command "GIB_RESULTATE_ATEST_K" 0 0 0

# Informationen zu den verschiedenen Review- und Korrekturtaetigkeiten
execute_command "GIB_RESULTATE_SREVIEW" 0 0 0
execute_command "GIB_RESULTATE_SREVIEW_K" 0 0 0
execute_command "GIB_RESULTATE_EREVIEW" 0 0 0
execute_command "GIB_RESULTATE_EREVIEW_K" 0 0 0
execute_command "GIB_RESULTATE_MREVIEW" 0 0 0
execute_command "GIB_RESULTATE_MREVIEW_K" 0 0 0
execute_command "GIB_RESULTATE_CREVIEW" 0 0 0
execute_command "GIB_RESULTATE_CREVIEW_K" 0 0 0
execute_command "GIB_RESULTATE_HREVIEW" 0 0 0
execute_command "GIB_RESULTATE_HREVIEW_K" 0 0 0

# Informationen zur Konsistenz der Dokumente
execute_command "GIB_KONSISTENZ_DOKUMENTE" 0 0 0

# Zusammenfassende Information zum Zustand aller Dokumente
execute_command "GIB_ZUSTAND_ALLER_DOKUMENTE" 0 0 0
execute_command "UEBERGIB_SYSTEM_AN_KUNDEN" 0 0 0
proceed  1 0 0 0
proceed  1 0 0 0
create_test_output sep_105.ist
```

*Abb. 6.5:* Erweiterungen in der Protokolldatei

verläufe stehen in SesamScore eine Reihe von Schemata zur Verfügung, die im Kapitel 4 schon genau untersucht wurden.

### *6.2.2 Parsen und Filtern der Daten aus den Dateien*

Die Problematik des „Nachspielens“ der Spielverläufe, um eine Auswertung ermöglichen zu können, hat schließlich dazu geführt, dass eine andere Möglichkeit gesucht wurde, um eine Auswertung eines Spiels vornehmen zu können. Unter genauerer Analyse der Situationsdatei konnte festgestellt werden, dass beinahe alle notwendigen Daten dort enthalten sind und eine Analyse des Spiels auch anhand der Situationsdatei durchgeführt werden kann. Allerdings bedeutete das, dass man sich die fehlenden Informationen und dabei handelt es sich um Informationen, die für die Auswertung des Personaleinsatzes notwendig sind, auf einem anderen Weg beschaffen muss.

Ein geringer Teil der Informationen über den Personaleinsatz kann der Situationsdatei entnommen werden. Es ist zum Beispiel möglich den Autor bzw. die Autoren eines Dokuments aus einer Liste von Records, wie sie in Abbildung 6.6 zu sehen ist, zu filtern. Dabei geht man wie folgt vor:
Jeder Mitarbeiter wird in der Situationsdatei durch eine eindeutige Id gekennzeichnet (siehe dazu Abbildung 6.7), die auch in dem Record des jeweiligen Dokuments enthalten ist, sofern er an der Erstellung des Dokuments beteiligt war.

Eine genauere Untersuchung der Hochsprachendateien ergab, dass es sich bei den Records um Anforderungen handelt, die in das Dokument übertragen wurden. Der Auszug der Hochsprachendatei „Schema“, in Abbildung 6.8 zeigt, dass eine Anforderung als Record definiert ist. Der dritte Eintrag des Records repräsentiert den Autor, der die Anforderung in das Dokument überführt hat. Außerdem handelt es sich bei den Anforderungen, die in das Dokument übertragen wurden um einen Bag von einzelnen Anforderungen, die bereits in dem Dokument realisiert wurden.
Laut der Spezifikation des Hochsprachenübersetzers bleibt die Reihenfolge innerhalb eines Records erhalten. Damit sieht in einer Situationsdatei zum Beispiel eine Anforderung in der Spezifikation wie folgt aus:

RECORD_IRIRRRRR_TYPE(8,13.95,5,8.12,0.0,0.0,0.0,0.0)

Damit handelt es sich um die Anforderung mit der Id 8, die einem Umfang von 13.95 AFPs aufweist, vom Entwickler mit der Id 5 realisiert wurde und

```
...
     create entity SPEZIFIKATION61: SPEZIFIKATION
           aka "Specification"' with
                 INHALT := BAG_RECORD_IRIRRRRR_TYPE'(
                    RECORD_IRIRRRRR_TYPE'(30,3.72,5,0.931947,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(10,4.65,5,1.16493,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(9,13.95,5,3.4948,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(18,3.72,5,0.931947,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(6,9.3,5,2.32987,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(7,13.95,5,3.4948,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(27,5.6939,5,1.96997,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(15,9.3,5,2.32987,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(19,2.75484,5,1.28617,0.0,0.0,0.0,0.0),
                    RECORD_IRIRRRRR_TYPE'(4,9.3,5,2.20858,0.0,0.0,0.0,0.0),);
                 UNBEARBEITETE_IDS := BAG_I_TYPE'();
                 ANZ_AFP := 197.827;
                 ANZ_FEHLER_GES := 51.0;
                 FEHLER_PRO_AFP := 0.2578;
                 PROZENT_AFP_100 := 98.4804;
                 PROZENT_AFP_ZV := -1.0;
                 SEITEN := 87;
                 UNBEARBEITETE_HB_IDS := BAG_I_TYPE'();
                 IN_ARBEIT_FUER_ENTW_IDS := BAG_I_TYPE'();
                 IN_ARBEIT_FUER_MSPEZ_IDS := BAG_I_TYPE'();
                 IN_ARBEIT_FUER_CODE_IDS := BAG_I_TYPE'();
                 IN_ARBEIT_FUER_HB_IDS := BAG_I_TYPE'();
                 ANZ_AF := 51.0;
                 AF_PRO_AFP := 0.2578;
                 FEHLER_PRO_SEITE := 0.586207;
                 KONS_SC_AF := 0.0;
                 PROZENT_FEHLER_PRO_SEITE_ZV := -1.0;
    end create;
...
```

*Abb. 6.6:* Personaleinsatz (Auszug aus der Situationsdatei)

8.12 Analysefehler enthält.

Über die Id kann der Autor des Dokuments identifiziert werden. Abbildung 6.6 zeigt, dass der Entwickler mit der Id 5 Autor des Spezifikationsdokuments ist, da er alle Anforderungen in der Anforderungsliste im Dokument realisiert hat. Sucht man nun in der Situationsdatei nach dem Entwickler, der zu dieser Id gehört (siehe Abbildung 6.7), findet man heraus, dass es sich bei diesem Entwickler um Richard handelt.

Leider können auf diese Weise nur die Autoren der Dokumente rekonstruieren werden. Interessant für die Auswertung eines Spielverlaufs wäre aber zum Beispiel auch, wie bei den Bewertungskriterien in Kapitel 5 gezeigt wur-

```
model QS_SAVE is
begin at 2000/04/18/08:00 using "model.dib"' seed "48979510, 43603650,
94833359, 47416679"'
...
         create entity ENTWICKLER31: ENTWICKLER aka "Richard"' with
                 KOSTEN := 100.0;
                 IST_EINGESTELLT := false;
                 VERFUEGBAR_AB := 2000/01/25/07:00;
                 AUTOR_ID := 5;
                 ERFAHRUNG := RECORD_IIIII_TYPE'(3,4,4,3,2);
                 NOTATIONSERFAHRUNG := SET_RECORD_SI_TYPE'(
                    RECORD_SI_TYPE'("German"',3),
                    RECORD_SI_TYPE'("German_in_Analysis"',3),
                    RECORD_SI_TYPE'("German_in_Spec"',4),
                    RECORD_SI_TYPE'("German_in_System_design"',4),
                    RECORD_SI_TYPE'("German_in_Module_design"',3),
                    RECORD_SI_TYPE'("German_in_Manuals"',3),
                    RECORD_SI_TYPE'("English"',3),
                    RECORD_SI_TYPE'("Ada95"',3),
                    RECORD_SI_TYPE'("C++"',3),
                    RECORD_SI_TYPE'("SA"',3));
         end create;
         create entity ENTWICKLER33: ENTWICKLER aka "Thomas"' with
                 KOSTEN := 90.0;
                 IST_EINGESTELLT := false;
                 VERFUEGBAR_AB := 2000/01/27/07:00;
                 AUTOR_ID := 7;
                 ERFAHRUNG := RECORD_IIIII_TYPE'(4,3,3,3,3);
                 NOTATIONSERFAHRUNG := SET_RECORD_SI_TYPE'(
                    RECORD_SI_TYPE'("German"',3),
                    RECORD_SI_TYPE'("German_in_Analysis"',4),
                    RECORD_SI_TYPE'("German_in_Spec"',3),
                    RECORD_SI_TYPE'("German_in_System_design"',3),
                    RECORD_SI_TYPE'("German_in_Module_design"',3),
                    RECORD_SI_TYPE'("German_in_Manuals"',3),
                    RECORD_SI_TYPE'("English"',3),
                    RECORD_SI_TYPE'("Ada95"',3),
                    RECORD_SI_TYPE'("C++"',3),
                    RECORD_SI_TYPE'("SA"',3));
         end create;
...
```

*Abb. 6.7:* Entwickler (Auszug aus der Situationsdatei)

```
type  Anforderung is record
   ID  : integer;  --zur Unterscheidung der Anforderungen
   AFP  : real;  --Komplexität in AFP
   Autor_ID  : integer;
   Analysefehler : real;    -- Anzahl der ...
   Grobentwurfsfehler : real; -- entsprechenden ....
   Feinentwurfsfehler : real; -- Fehler, die in ...
   Implementierungsfehler  : real; -- der Anforderung ...
   Handbuchfehler : real; -- enthalten sind.
end;

type Anforderungen is bag of Anforderung;
```

*Abb. 6.8:* Definition Anforderung (Auszug aus „Schema.hs“)

de, welches Reviewteam für die Prüfung von Dokumenten eingesetzt wurde, um die Effektivität eines Reviews untersuchen zu können. Unter genauerer Betrachtung der vorhandenen Dateien konnte herausgefunden werden, dass diese Informationen aus der Protokolldatei extrahieren werden können, da es dort ein Kommando LASSE_REVIEW_STATTFINDEN(Diana, Christine, Spezifikation) geben muss. Dieses Kommando sagt aus, dass Diana und Christine als Gutachter für die Prüfung der Spezifikation eingesetzt wurden. Es müsste in diesem Fall jedoch genau festgehalten werden, um den wievielten Review der Spezifikation es sich handelt. Denn der Spieler kann mehrere Reviews für ein Dokument anordnen und dazu auch verschiedene Gutachter einladen.
Anhand genauer Untersuchungen aller Dateien, die während des Spiels erzeugt werden, konnte ermittelt werden, dass alle fehlenden Informationen für den Personaleinsatz, die für eine Auswertung notwendig sind, auch auf anderen Wegen beschafft werden können. Das Nachspielen des Projekts kann durch diesen Ansatz vermieden werden. Die Daten können jedoch nur sehr aufwändig zum einen aus der unstrukturierten Situationsdatei, und zum anderen aus der Protokolldatei extrahiert werden. Der neue Ansatz weist kaum Vorteile gegenüber dem ersten Ansatz auf, da sich das Parsen der Dateien und das Filtern und Strukturieren der Daten als aufwändig erwiesen hat. Ein alternativer Ansatz musste gefunden werden.

### *6.2.3 Verwendung einer Datenbank*

Parallel zur Entwicklung einer generischen Erklärungskomponente wurden andere Hilfsmittel für das SESAM-Spiel entwickelt. Zu diesen Hilfsmitteln

zählen „Ratgeber“ und „väterlicher Freund“, die in der Diplomarbeit von Markus Nusser [NUSSER] näher beschrieben werden. Die beiden Hilfsmittel sollen den Spieler während des Spiels beim Treffen von Entscheidungen unterstützen. Es wurde rasch festgestellt, dass auch sie Daten aus der Situationsdatei benötigten. Eine genauere Untersuchung der relevanten Daten für den Ratgeber bzw. den „väterlichen Freund“ ergab, dass viele dieser Daten auch für die Erklärungskomponente interessant sind. Aufgrund dieser Entwicklungen konnte ein neuer Ansatz überlegt werden.

Da alle Hilfsmittel die gleichen Daten benötigen, ist es von Vorteil, diese Daten für alle Hilfsmittel leicht zugänglich zu machen. Wenn man die Daten, die bei der Verarbeitung im SESAM-Simulator entstehen, in einer Datenbank sichert, dann können alle Hilfsmittel, egal ob Erklärungskomponente, Ratgeber oder „väterlicher Freund“, recht einfach auf diese Daten zugreifen und sie für ihre Zwecke nutzen.

Bei der Erstellung des Datenmodells konnte festgestellt werden, dass man die Hilfsmittel so entwickeln kann, dass sie für unterschiedliche Modelle eingesetzt werden können. Außerdem fand man heraus, dass alle Hilfsmittel gleich aufgebaut sind, und deshalb alle durch dieselbe Architektur realisiert werden können. Die verwendete Architektur der Hilfsmittelkomponenten wird in Kapitel 7 beschrieben. Aufgrund dieser enormen Vorteile wurde dieser Ansatz im AMEISE Projekt verwirklicht.

Auf die Generizität, die durch die Datenbank ermöglicht wird, wird in Abschnitt 6.3.4 noch genauer eingegangen. Damit der Leser den dortigen Ausführungen besser folgen kann, muss zuvor noch beschrieben werden, wie die Daten, die bei der Simulation erzeugt werden, in die Datenbank übernommen werden. Dazu mussten die so genannten ZARMS-Daten genauer untersucht werden.

## *6.3 ZARMS-Daten*

Die Daten, der internen Datenstruktur im Simulator, werden ZARMS-Daten (ZARMS steht für Zustandsanalyse und Regel - Monitor für SESAM) genannt und spiegeln als Ganzes somit die aktuelle Situation des Projekts wider. Mit anderen Worten, die ZARMS-Daten beinhalten aller Entitäten und Relationen eines Zustands sowie die dazugehörigen Attribute und ihre Wertausprägungen zum aktuellen Zeitpunkt der Simulation. Das bedeutet, dass alle Daten, die in der Situationsdatei festgehalten werden, bereits in

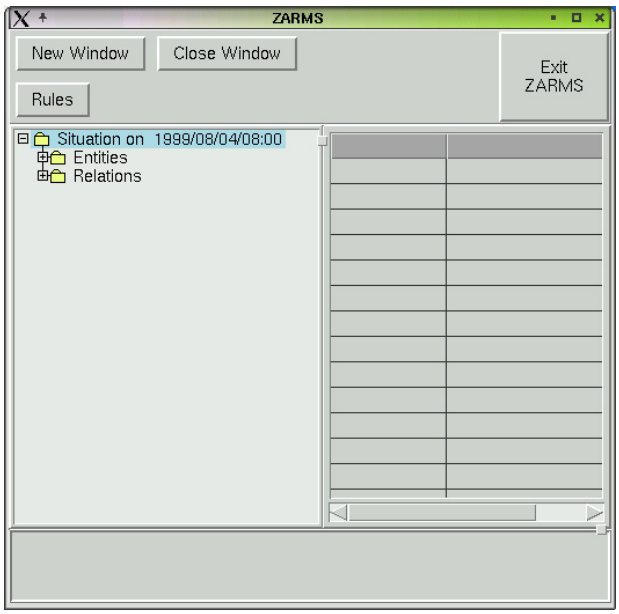



*Abb. 6.9:* ZARMS Struktur

strukturierter Form in einem ZARMS-Datenstrom vorliegen.

### *6.3.1 Aufbau der ZARMS-Daten*

Die ZARMS-Daten können als Baum repräsentiert werden. Die Abbildung 6.9 veranschaulicht die Struktur, die den ZARMS-Daten zugrunde liegt. Auf erster Ebene werden die Entitäten von den Relationen getrennt. Sowohl die Entitäten als auch die Relationen sind gleich aufgebaut. Abbildung 6.10 zeigt den genauen Aufbau (im Weiteren auch ZARMS-Pfad genannt) am Beispiel der Entitäten.

#### *Entitäten*

Die erste Ebene bei den Entitäten (wie auch bei den Relationen) wird als ZARMS-Typ bezeichnet. Im obigen Beispiel 6.10 wird der ZARMS-Typ „PROJEKTZUSTAND“ genauer betrachtet. Die nächste Ebene wird als ZARMS-Entität bezeichnet. In Abbildung 6.10 wurde die ZARMS-Entität „Projektstatus“ ausgewählt.

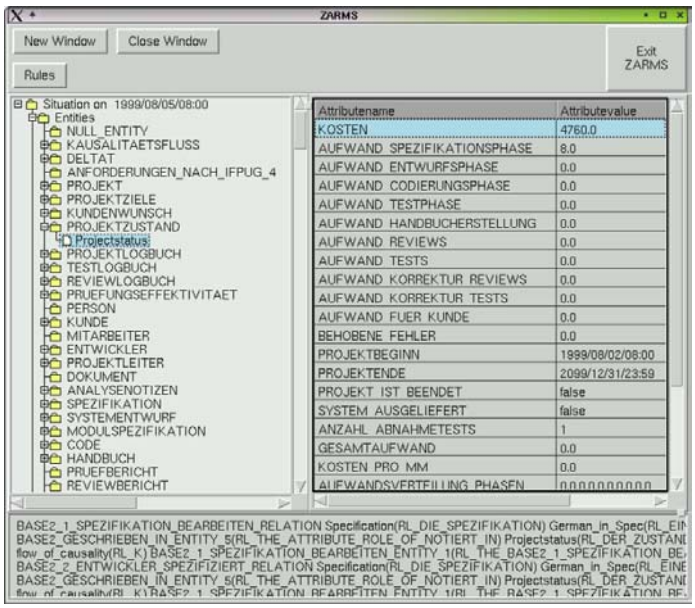



*Abb. 6.10:* ZARMS-Entität Projektstatus

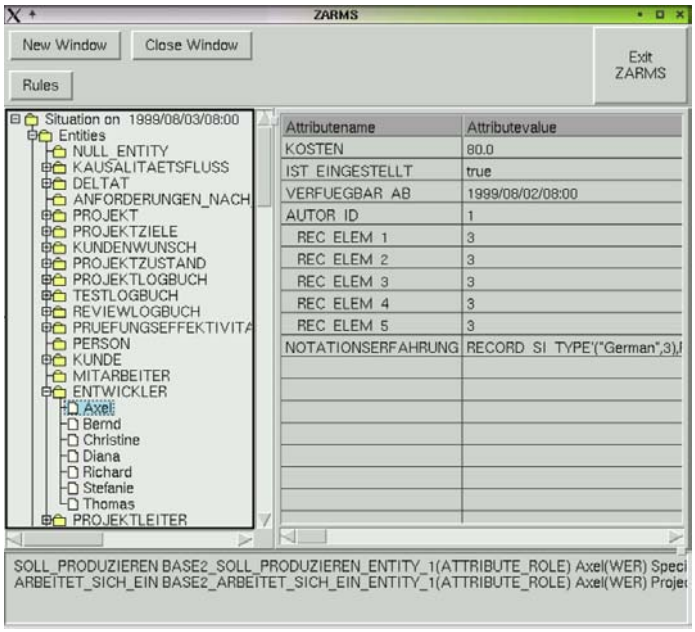



*Abb. 6.11:* ZARMS-Typ Entwickler

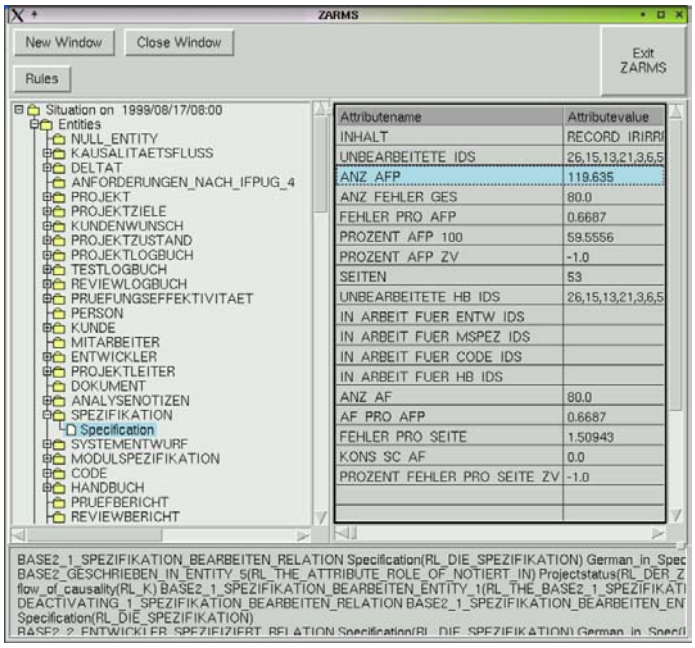



*Abb. 6.12:* ZARMS-Entität Specification

Es kann auch mehrere ZARMS-Entitäten geben, wie die Abbildung 6.11 zeigt. Im QS-Modell gibt es sieben Entwickler, von denen jeder als ZARMS-Entität dargestellt wird, da jede ZARMS-Entität grundsätzlich verschiedene Attribute aufweisen kann. Im Fall der Entwickler haben zwar alle Entitäten die gleichen Attribute, allerdings gibt es unterschiedliche Attributwerte, da zum Beispiel Richard 100 DM pro Stunde kostet, während es bei Axel nur 80 DM sind.

Die letzte Ebene bilden die ZARMS-Attribute, die in den ZARMS-Abbildungen jeweils auf der rechten Seite zu sehen sind. Im Beispiel in Abbildung 6.10 gibt es eine Vielzahl von ZARMS-Attributen, die zu der ZARMS-Entität „Projektstatus“ gehören. Es wird sowohl die Bezeichnung als auch der aktuelle Wert des ZARMS-Attributs angezeigt. Wichtige Attribute für die Auswertung sind unter anderem:

- KOSTEN: bisher verbrauchtes Budget
- PROJEKTBEGINN bzw. PROJEKTENDE: Dauer des Projekts
- AUFWANDSVERTEILUNG: Verteilung des Aufwands über alle Phasen

Im Beispiel in Abbildung 6.12 hingegen handelt es sich um die ZARMS-Entität „Specification“. Unter den zugehörigen Attributen können auch hier für die Auswertung wichtige Attribute identifiziert werden:

- ANZ_AFP: Anzahl der realisierten AFPs im Dokument
- ANZ_FEHLER_GES: Gesamtfehler im Dokument
- SEITEN: Umfang des Dokuments in Seiten
- ANZ_AF: Anzahl der Analysefehler im Dokument
- FEHLER_PRO_SEITE: Fehler pro Seite im Dokument

Diese Attribute sind auch bei allen anderen Dokumenten zu finden. Allerdings hat jedes Dokument noch zusätzliche Attribute, die spezielle Eigenschaften des Dokuments beschreiben. Eine genaue Auflistung der für die Auswertung interessanten Attribute findet der Leser in Anhang D.

*Relationen*

Bei den Relationen gibt es ebenfalls mehrere Ebenen. Der Aufbau ist der gleiche wie bei den Entitäten.

Abbildung 6.13 zeigt eine Relation (ZARMS-Typ), die den Namen „PRODUZIERT“ trät. Die zweite Ebene (ZARMS-Entität) weist den gleichen Namen auf, allerdings werden die ZARMS-Entitäten[2] durchnummeriert (beginnend bei „#1“, „#2“, usw.), um eine Identifikation zu ermöglichen. Die Attribute (ZARMS-Attribute) befinden sich wieder auf der dritten Ebene.

Die Relationen „PRODUZIERT“, „BEGUTACHTET“, „KORRIGIERT“, „TESTET_SYSTEM“ usw. enthalten wichtige Informationen, die für den Personaleinsatz relevant sind. Betrachtet man den ZARMS-Typ „PRODUZIERT“ in Abbildung 6.13, kann man Informationen darüber erhalten, welcher Mitarbeiter gerade mit welcher Tätigkeit beschäftigt ist. Nach genauerer Betrachtung der zwei „PRODUZIERT“-Relationen kann ermittelt werden, dass im ersten Fall (Abbildung 6.13) Mitarbeiter Bernd gerade mit dem Handbuch

[2] Der Begriff der ZARMS-Entität ist in diesem Zusammenhang irreführend, da es sich eigentlich um eine Relation handelt. Da aber sowohl der Entitäten- als auch der Relationenstrom gleich aufgebaut sind, kann diese Unterscheidung auf dieser Ebene nicht mehr getroffen werden. Es kann lediglich anhand des Datenstroms, der gerade bearbeitet, wird festgestellt werden, ob es sich um Entitäten oder Relationen handelt.

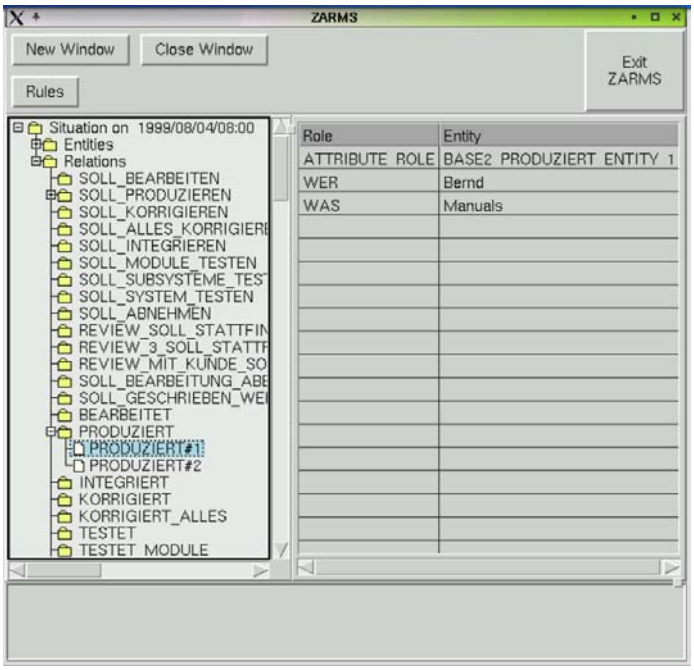



*Abb. 6.13:* ZARMS-Relation „PRODUZIERT#1“

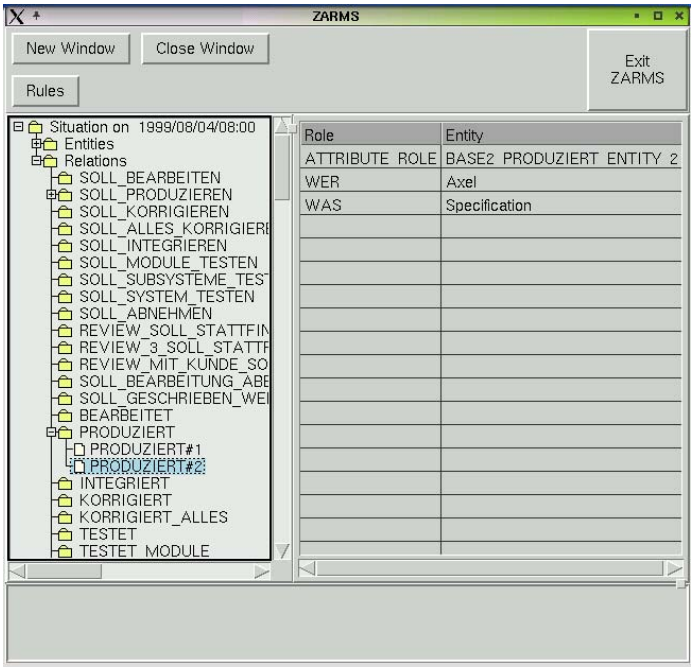



*Abb. 6.14:* ZARMS-Relation „PRODUZIERT#2“

(Manuals) beschäftigt ist, während im zweiten Fall (Abbildung 6.14) Mitarbeiter Axel an der Spezifikation arbeitet.

Diese Darstellung der ZARMS-Daten steht ursprünglich in Form einer Zeichenkette zur Verfügung, der in der Urversion des SESAM-Simulators aus Visualisierungsgründen als Baumstruktur aufbereitet wurde. Für die Realisierung unseres Ansatzes benötigen wir jedoch die ursprüngliche Zeichenkette, um die Daten in die Datenbank überführen zu können.

### *6.3.2 Parsen des ZARMS-Datenstroms*

Der ZARMS-Datenstrom, der die aktuelle Situation des Projekts enthält muss durchlaufen werden. Dabei müssen alle relevanten Daten aus dem Datenstrom gefiltert und an die vorgeschriebene Stelle in der Datenbank gespeichert werden.

Der ZARMS-Datenstrom besteht aus einem Entitäten- und einem Relationen-Datenstrom. Beide sind wie folgt aufgebaut:

```
Datenstrom  ::= ZARMS-type ('§' ZARMS-type)*
ZARMS-type   ::= <type> '*' ZARMS-entity
ZARMS-entity ::= <entity> ZARMS-attribute+
ZARMS-attribute ::= '^' <attrib> '*' <attribval>
```

In den folgenden Abschnitten werden die beiden Datenströme genauer untersucht, um zeigen zu können, wie das Filtern der relevanten ZARMS-Attribute möglich ist.

#### *Der Entitäten-Datenstrom*

Das folgende Beispiel zeigt, wie der Entitäten-Datenstrom aufgebaut ist:

```
...§ PROJEKTZUSTAND * Projektstatus ^ KOSTEN * 450000 ^
PROJEKTBEGINN * 1999/08/02 ^ PROJEKTENDE * 2000/04/17 ^
PROJEKT_IST_BEENDET * true §...
```

Jeder ZARMS-Typ wird durch ein „§“-Zeichen vom nächsten ZARMS-Typ getrennt. Die erste Ebene ist demnach im obigen Beispiel „PROJEKTZUSTAND“. Die erste Ebene (ZARMS-Typ) wird von der zweiten Ebene (ZARMS-Entität) durch ein „∗“-Zeichen getrennt. Im obigen Beispiel heißt demnach die zweite Ebene „Projektstatus“. Nach der ZARMS-Entität folgt ein „^“-Zeichen

und eine Liste von Attributen. Die Attribute werden als Attribut-Wert-Tupel angegeben, die jeweils durch ein ‘“∗“-Zeichen voneinander und durch ein „^“-Zeichen vom nächsten Tupel getrennt sind.

Beim Durchlaufen des Datenstroms wird zuerst überprüft, ob der ZARMS-Typ und die ZARMS-Entität zu den zu filternden Entitäten gehören. Ist dies nicht der Fall, dann wird die anschließende Liste von Attribut-Wert-Tupeln übersprungen und die Überprüfung beim nächsten ZARMS-Typ fortgesetzt. Handelt es sich jedoch um einen ZARMS-Typ bzw. um eine ZARMS-Entität, die gefiltert werden soll, dann wird die anschließende Liste der Attribut-Wert-Tupel durchlaufen. Schritt für Schritt wird kontrolliert ob das Attribut zu den zu filternden Attributen zählt. Ist dies der Fall, dann wird es in der Datenbank unter demselben ZARMS-Pfad (ZARMS-Typ, ZARMS-Entität, ZARMS-Attribut) gespeichert. Im obrigen Beispiel setzt sich der ZARMS-Pfad wie folgt zusammen: ZARMS-Typ = “PROJEKTZUSTAND”, ZARMS-Entität = „Projektstatus“ und ZARMS-Attributbeschreibung = KOSTEN und ZARMS-Attributwert = 450000. Sollte es sich bei dem Attribut nicht um ein relevantes Attribut handeln, dann wird es übersprungen und das nächste Attribut wird überprüft.

*Der Relationen-Datenstrom*

Der Relationen-Datenstrom ist grundsätzlich gleich aufgebaut wie der Entitäten-Datenstrom. Allerdings gibt es bei den Relationen zusätzliche Aspekte, die anhand des folgenden Auszugs aus dem Datenstrom gezeigt werden sollen:

```
...§ PRODUZIERT * PRODUZIERT#1 ^ WER * Axel ^ WAS * Code §...
```

Auch im Relationen-Datenstrom wird ein Glied der Zeichenkette durch das „§“-Zeichen von den anderen getrennt. Dann folgen wie bei den Entitäten zuerst der ZARMS-Typ und dann die ZARMS-Entität. Nach der Angabe des ZARMS-Typs und der ZARMS-Entität folgt wieder eine Liste von Attribut-Wert-Tupeln, die für diese Relation gelten.

Das Vorgehen beim Parsen des Relationen-Datenstroms weist im Gegensatz zu dem bei den Entitäten nur einen Unterschied auf. Im Fall der Relationen wird auf die zweite Ebene (ZARMS-Entität) keine Rücksicht genommen. Das würde für das obige Beispiel bedeuten, dass nur geprüft wird, ob der ZARMS-Typ „PRODUZIERT“ gefiltert werden muss oder nicht. Handelt es sich dabei um keinen relevanten ZARMS-Typ, dann wird die Verarbeitung beim nächsten ZARMS-Typ fortgesetzt. Muss der ZARMS-Typ gefiltert werden, dann

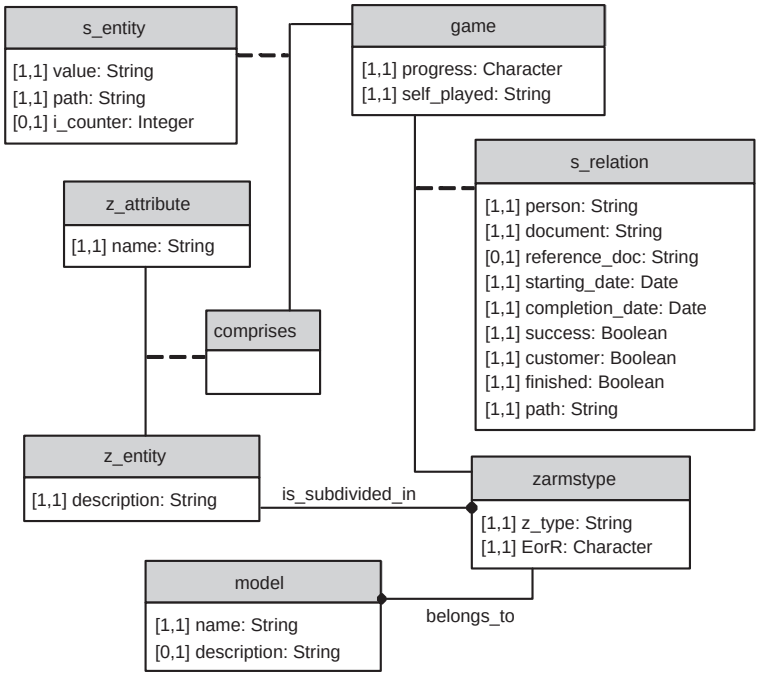



*Abb. 6.15:* Realisierung des ZARMS-Pfads in der Datenbank (Auszug aus dem Datenmodell)

werden alle Attribute der Attributliste in die Datenbank übertragen. Bei Relationen sind alle Attribute von Bedeutung, deshalb ist ein Filtern der einzelnen Attribute nicht notwendig.

### *6.3.3 Realisierung des ZARMS-Pfades in der Datenbank*

Wie bereits beschrieben wurde, setzt sich der ZARMS-Pfad sowohl bei Entitäten als auch bei Relationen aus ZARMS-Typ, ZARMS-Entität und ZARMS-Attribut zusammen. Diese Struktur wurde auch in die Datenbank übernommen, wie der Ausschnitt in Abbildung 6.15 zeigt. Das vollständige Datenmodell findet der Leser in Anhang B.

#### *Speichern von Entitäten in der Datenbank*

Die Entitäten können auf diese Weise recht einfach in die Datenbank übertragen werden. Der ZARMS-Typ wird in der Datenbank durch die Tabelle „zarmstype“ repräsentiert. Jedem ZARMS-Typ können mehrere ZARMS-

Entitäten (Datenbank-Tabelle „z_entity“) zugeordnet sein, wie wir am Beispiel der Entwickler gesehen haben. Die letzte Ebene bilden die Attribute, die in der Tabelle „z_attribute“ gespeichert werden. Auch hier gilt, dass einer ZARMS-Entität mehrere Attribute zugeordnet sein können. Außerdem kann ein Attributname mehreren Entitäten zugeteilt werden. Die aktuelle Wertausprägung, die ein Attribut in einem bestimmten Spiel (Abhängigkeit zur Tabelle „game“) zu einem bestimmten Zeitpunkt (wird durch den Eintrag „path“ festgehalten) aufweist, wird in der Tabelle „s_entity“ gespeichert.

*Speichern von Relationen in der Datenbank*

Relationen hingegen können nicht direkt, entsprechend ihrem ZARMS-Pfad, in die Datenbank übernommen werden, sondern müssen den Attributen in der Datenbanktabelle „s_relation“ zugeordnet werden. Das bedeutet, dass der ZARMS-Pfad bei Relationen nur durch den ZARMS-Typ (Datenbank-Tabelle „zarmstype“) repräsentiert wird.

Wird beim Parsen des Relationen-Datenstroms ein relevanter ZARMS-Typ gefunden, dann werden alle Attribute in der Attributliste dieses ZARMS-Typs den Attributen der Datenbanktabelle „s_relation“ zugewiesen. Betrachtet man noch einmal das Beispiel des Relationen-Datenstroms, dann würde das bedeuten, dass der Wert des Attributes „WER“, also Axel, in der „s_relation“-Tabelle im Attribut „person“ gespeichert werden muss. Der Wert des Attributes „WAS“, also die Spezifikation, wird im Attribut „document“ festgehalten.

*Zusätzliche Flags für Relationen*

Relationen bereiten aber auch Schwierigkeiten. Bei Entitäten gibt es zwei Einträge, die den Beginn und das Ende einer Phase festhalten. Das Ende einer Phase wird bei jedem „Proceed“ um eine Zeiteinheit erhöht, solange der Mitarbeiter seine Tätigkeit noch nicht vollendet hat. Beendet er seine Aktivität, dann wird das Ende der Phase eingefroren und bleibt somit erhalten. Bei Relationen ist dies nicht so einfach. Sie tauchen plötzlich auf und verschwinden nach Beendigung der Aktivität wieder. Es gibt keine Attribute, die den Beginn und das Ende der Aktivität festhalten. Um den Beginn einer Tätigkeit fixieren zu können, wird beim ersten Erscheinen der Relation ein Attribut „starting_date“ in der Datenbanktabelle „s_relation“ mit den aktuellen Datum gefüllt. Dadurch kann das Beginndatum eingefroren werden. Hat der Mitarbeiter seine Tätigkeit beendet, dann verschwindet die Relation. Das bedeutet, dass der Mitarbeiter am Tag zuvor mit seiner Tätigkeit fertig ge-

worden ist. Deshalb wird ein zusätzliches Attribut benötigt, das das Ende der Aktivität festhalten kann. Sobald eine Relation das erste mal erscheint, wird darum nicht nur das „starting_date“, sondern auch das „completion_date“ auf das aktuelle Datum gesetzt. An jedem folgenden Tag, an dem die Relation noch existiert, wird das „completion_date“ aktualisiert, das heißt auf das aktuelle Datum gesetzt. Dadurch kann garantiert werden, dass das Endedatum in der „s_relation“ mit dem tatsächlichen Ende der Aktivität übereinstimmt.

Ein weiterer wichtiger Punkt, der für die Auswertung von Bedeutung ist, ist die Tatsache, ob eine Tätigkeit erfolgreich durchgeführt werden konnte, oder ob der Mitarbeiter die Aktivität nicht erledigen konnte, weil zum Beispiel im QS-Modell zum aktuellen Zeitpunkt das Vorgabedokument nicht zu 50 % fertig gestellt war, oder weil der Mitarbeiter noch mit einer anderen Tätigkeit beschäftigt war. Aus den Attributen der Entitäten geht jedoch genau hervor, ob eine Tätigkeit erfolgreich war oder nicht. Diese Attribute müssen überprüft werden bevor die Werte in der Tabelle „s_relation“ gespeichert werden. Das „success“-Flag kann dementsprechend auf true oder false gesetzt werden.

Ein weiteres Flag ist das „customer“-Flag, das im Falle eines Spezifikations- oder Handbuchreviews angibt, ob der Kunde an diesem Review teilgenommen hat. Die Teilnahme des Kunden geht aus einer weiteren Relation hervor, deren Existenz vor dem setzten des Flags überprüft werden muss.

### *6.3.4 Generizität des gewählten Ansatzes*

In diesem Abschnitt wird auf zwei Teile der Datenbank (ZARMS-Daten und Hilfsmittelkomponenten) eingegangen, mit deren Hilfe die Generizität des gewählten Ansatzes realisiert werden konnte.

#### *ZARMS-Daten*

Im Datenmodell in Abbildung 6.15 wurde durch die Beziehung zwischen „model“ und „zarmstype“ gezeigt, dass die ZARMS-Daten jeweils für ein bestimmtes Modell gültig sind. Es muss an dieser Stelle jedoch darauf hingewiesen werden, dass es mehrere Instanzen eines Modells geben kann. Im Abschnitt 3.3 wurden bereits verschiedene Erweiterungen (Modellinstanzen) des QS-Modells vorgestellt wie zum Beispiel das QSVA-Modell. Jede Instanz eines Modells basiert grundsätzlich auf denselben ZARMS-Daten, das heißt dem QSVA-Modell liegen dieselben ZARMS-Daten zugrunde wie dem klassischen QS-Modell. Allerdings gibt es in jeder Modellinstanz auch zusätzliche

ZARMS-Attribute, die nur für diese Instanz gelten. Das QSVA-Modell zum Beispiel weist spezielle ZARMS-Attribute auf, die für die genauere Betrachtung der Reviews benötigt werden.

Da der Aufbau der ZARMS-Daten (ZARMS-type, ZARMS-entity, ZARMS-attribute) in allen Modellen der gleiche ist, und sich nur die Attributbezeichnungen und die Werte ändern, kann die Datenbank verschiedene Modelle, die auf unterschiedlichen ZARMS-Daten beruhen in dieser Struktur verwalten.

*Hilfsmittelkomponente*

Nun muss noch gezeigt werden, dass auch die Auswertung der Bewertungskriterien generisch realisiert werden kann.

Um Spiele anhand der in Kapitel 5 vorgestellten Bewertungskriterien analysieren zu können, müssen aktuelle Werte verschiedener ZARMS-Attribute eines Spielers betrachtet und mit Richtwerten verglichen werden. Betrachtet man die einzelnen Bewertungskriterien genauer, so kann man erkennen, dass gewisse Fragestellungen immer wieder gelöst werden müssen, und dass diese Fragestellungen alle ähnlich sind. Beispiele dafür wären:

- KOSTEN <= 450 000 DM =» „Budget nicht überschritten!“
- AFPs >= 95 % =» „Geforderte AFPs erreicht!“
- Dauer > 270 Tage =» „Projektdauer überschritten!“
- Autor Code == „Diana“ =» „Qualifizierteste Mitarbeiterin eingesetzt!“

Zur Lösung dieses Problems bietet sich das Design Pattern „Interpreter“ [Vlissides] an, welches folgendes vorschlägt: „Definiere aus einer gegebenen Sprache eine Repräsentation der Grammatik sowie einen Interpreter, der die Repräsentation nutzen kann, um Sätze in dieser Sprache interpretieren zu können.“

Betrachten wir für die weiteren Ausführungen ein Beispiel genauer. Für die Auswertung des Bewertungskriteriums „KOSTEN“ muss der aktuelle Wert des ZARMS-Attributs „KOSTEN“ mit dem Richtwert 450 000 verglichen werden, um feststellen zu können, ob der Spieler die Zielvorgabe erfüllt hat oder nicht. Das bedeutet, dass die Erklärungskomponente für die Analyse dieses Bewertungskriteriums überprüfen muss, welcher der folgenden zwei Fälle zum aktuellen Zeitpunkt zutrifft:

- **Attribut-Wert „KOSTEN“ <= 450 000**
  =» „Die Budgetgrenze von 450 000 DM wurde nicht überschritten!“
- **Attribut-Wert „KOSTEN“ > 450 000**
  =» „Die Budgetgrenze von 450 000 DM wurde überschritten!“

Anhand einer genaueren Untersuchung der einzelnen Bewertungskriterien, konnte die Grammatik festgelegt werden. Ein Satz besteht demnach aus genau einem Attributwert gefolgt von genau einer Regel, die sich aus Operator und Richtwert zusammensetzt. Aus mehreren Sätzen lassen sich Satzketten bilden, die mit dem logischen „UND“-Operator miteinander verkettet werden. Zum Beispiel:

- **AFPs > 199 „UND“ Fehler < 30**
  =» „Es wurde ein recht vollständiges und korrektes Dokument erstellt!“
- **Autor == „Diana“ „UND“ qualifiziert == „Diana“**
  =» „Das Dokument wurde von einem geeigneten Mitarbeiter erstellt!“

Dadurch können auch komplexere Bewertungskriterien analysiert werden. Wie diese Verkettung konkret eingesetzt werden kann, wird in Kapitel 7 erklärt.
Für jede Satzkette, die aus ein oder mehreren Sätzen bestehen kann, gibt es genau einen Erklärungstext, der allerdings nur ausgegeben wird, wenn alle Sätze der Satzkette zutreffen.
Die Repräsentation dieser Grammatik wurde durch die spezielle Hilfsmittelkomponente realisiert.

Im folgenden Abschnitt wird etwas vorgegriffen, um die Generizität des Ansatzes zeigen zu können. Die Architektur der speziellen Hilfsmittelkomponente wird in Kapitel 7 genauer beschrieben.

Das Bewertungskriterium „KOSTEN“ kann demnach durch zwei Sätze analysiert werden. Jede dieser Satzketten wird im AMEISE-System jeweils durch eine spezielle Hilfsmittelkomponente repräsentiert. Die spezielle Hilfsmittelkomponente setzt sich aus einer Abfrage, einer Regel und einem Erklärungstext zusammen. Die Abfrage enthält das Attribut, dessen aktueller Wert aus der Datenbank geladen werden muss. Im oberen Beispiel wäre dies das ZARMS-Attribut „KOSTEN“. Die Regel hingegen beschreibt, wie, also mit welchem Operator, und womit, das heißt mit welchem Richtwert, dieser Attribut-Wert verglichen werden muss. In unserem Beispiel wäre der Richtwert zwar in beiden Fällen der gleiche, der Operator hingegen einmal „<=“ und das andere Mal „>“.

Der Erklärungstext gilt jeweils nur für eine spezielle Hilfsmittelkomponente und wird dem Spieler als Ergebnis der Auswertung angezeigt. Im oberen Beispiel gibt es demnach zwei unterschiedliche Erklärungstexte, einen, falls der Spieler die Zielvorgabe erfüllt hat und einen anderen, falls er die Kostengrenze überschritten hat.
Auf eine Einschränkung muss jedoch an dieser Stelle hingewiesen werden. Die Architektur der speziellen Hilfsmittelkomponente erlaubt nur paarweise Vergleiche. Da jedoch bei den Bewertungskriterien nur diese Vergleichsart aufgetreten ist, wurde die Realisierung von Mehrfachvergleichen bei der Erstellung der Architektur der Hilfsmittelkomponenten nicht berücksichtigt.

Werfen wir nun einen kurzen Blick nach vorne auf die Abbildung 7.9, die den Ausschnitt des Datenmodells darstellt, der sich mit der speziellen Hilfsmittelkomponente beschäftigt. Man kann erkennen, dass jede spezielle Hilfsmittelkomponente („specific_aid“) an einem Modell („model“) hängt. Das bedeutet, dass die Hilfsmittelkomponenten modellabhängig sind, da sie auf den ZARMS-Daten des Modells beruhen. Die speziellen Hilfsmittelkomponenten eines Modells können aber sowohl für die Auswertung bestehender Instanzen als auch für neue Instanzen dieses Modells herangezogen werden, da die zugrunde liegenden ZARMS-Daten dieselben sind. Dies geschieht durch eine einfache Zuweisung der bestehenden Hilfsmittelkomponenten zu der neuen Modellinstanz. Es muss aber darauf hingewiesen werden, dass einige Hilfsmittelkomponenten vielleicht durch die Erweiterung des Modells ihre Gültigkeit verloren haben. Diese dürfen nicht von dieser Instanz übernommen werden. Außerdem müssen zusätzliche Hilfsmittelkomponenten entwickelt und in der Datenbank abgelegt werden, damit auch die neuen Effekte dieser Modellinstanz ausgewertet werden können.
Wird ein neues Modell verwendet, dass auf anderen ZARMS-Daten beruht, dann müssen zum einen neue Bewertungskriterien identifiziert, und zum anderen völlig neue Hilfsmittelkomponenten erstellt werden, die diese Kriterien auswerten können.

Um nun die Auswertung eines Bewertungskriteriums vornehmen zu können, muss die generische Erklärungskomponente alle speziellen Hilfsmittelkomponenten, die dieses Bewertungskriterium repräsentieren, interpretieren. Die Erklärungskomponente fungiert dabei im Sinne des beschriebenen Design Patterns nur als Regelinterpreter. Die einzelnen Hilfsmittelkomponenten werden von der Erklärungskomponente interpretieren und anschließend verarbeitet. Trifft die Hilfsmittelkomponente zum aktuellen Zeitpunkt zu, dann kann die Erklärungskomponente als Ergebnis den Erklärungstext dieser Hilfsmittelkomponente zurückgeben.

Die Generizität dieses Ansatzes ermöglicht es, dass auch bei einem Modellwechsel die Architektur der speziellen Hilfsmittelkomponente verwendet werden kann. Es müssen lediglich die modellspezifischen Daten für Abfrage und Regel sowie entsprechende Erklärungstexte zu neuen speziellen Hilfsmittelkomponenten zusammengesetzt werden. Aufgrund der Generizität der Erklärungskomponente müssen an ihr keine Änderungen vorgenommen werden, solange sich nicht die Grammatik der Sprache ändert.

In diesem Kapitel wurden die Architekturüberlegungen und die Designentscheidungen beschrieben, die bei der Entwicklung der generischen Erklärungskomponente vorgenommen wurden. Im Weiteren wird nun genauer auf die Überführung der Bewertungskriterien in spezielle Hilfsmittelkomponenten eingegangen. Außerdem wird beschrieben, wie die Erklärungskomponente die Auswertung eines Bewertungskriteriums vornimmt.

*Kapitel 7*

# HILFSMITTELKOMPONENTEN

Wir haben in Kapitel 5 eine Vielzahl von Bewertungskriterien gefunden, die für die Analyse eines Spielverlaufs von der generischen Erklärungskomponente genauer untersucht werden können. Es wurde bereits gezeigt, dass alle Daten, die für die Auswertung benötigt werden, in einer Datenbank zur Verfügung stehen, da die ZARMS-Daten nach jedem „Proceed“ aus dem Simulator geholt und in die Datenbank übertragen werden. Mit einfachen SQL-Abfragen können die Hilfsmittel (Ratgeber, „väterlicher Freund“ und Erklärungskomponente) auf diese Daten zugreifen und diese für ihre Zwecke nutzen.

Die Erklärungskomponente hat nun die Aufgabe, sich bestimmte Daten aus der Datenbank zu holen und diese zu analysieren. Um Aussagen über die ausgelesenen Attributwerte treffen zu können, müssen diese mit Richtwerten verglichen werden. Wie dabei vorgegangen wird, wird im folgenden Abschnitt genauer betrachtet.

## *7.1 Architektur der Hilfsmittelkomponenten*

Nehmen wir an, die Erklärungskomponente möchte das Bewertungskriterium „Kosten“ auswerten. Dieses Bewertungskriterium sagt aus, dass der Spieler für die Durchführung des Projekts nicht mehr als 450 000 DM verbrauchen darf. Um den Spielverlauf aufgrund dieses Bewertungskriteriums analysieren zu können, müssen wir zuerst den aktuellen Wert des ZARMS-Attributs „KOSTEN“ aus der Datenbank laden. Um jedoch eine Aussage darüber treffen zu können, ob der Spieler die Budgetgrenze überschritten hat oder nicht, muss der Attributwert zusätzlich mit dem Richtwert „450 000“ verglichen werden. Nun gibt es folgenden Möglichkeiten:

- Attributwert „KOSTEN“ $>$ 450 000
- Attributwert „KOSTEN“ $<=$ 450 000

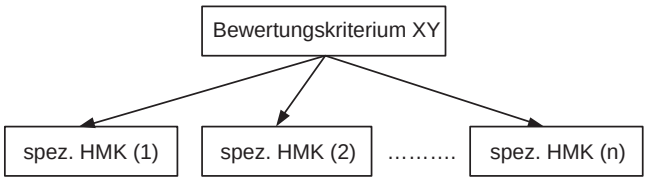



*Abb. 7.1:* Aufbau eines Bewertungskriteriums

Im AMEISE-System werden Bewertungskriterien durch spezielle Hilfsmittelkomponenten realisiert (siehe Abbildung 7.1). Um gewährleisten zu können, dass zu jedem Zeitpunkt im Spiel eine Aussage über ein Bewertungskriterium getroffen werden kann, müssen alle Werte, die das Bewertungskriterium annehmen kann, von speziellen Hilfsmittelkomponenten abgedeckt werden. In unserem Beispiel gibt es demnach zwei spezielle Hilfsmittelkomponenten (Abbildung 7.2), die die beiden Aussagen überprüfen, die wir aufgestellt haben.

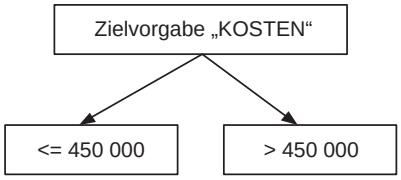



*Abb. 7.2:* Hilfsmittelkomponenten für das Bewertungskriterium „Kosten“

Bei unserem Beispiel handelt es sich um einen einfachen Fall. Zwei spezielle Hilfsmittelkomponenten genügen, um erkennen zu können, ob der Spieler die Zielvorgabe erfüllt hat oder nicht. Es kann aber auch mehrere Wertebereiche geben, wie sie die Abbildung 7.3 zeigt, in denen der aktuelle Wert liegen kann. In diesem Fall sind vier spezielle Hilfsmittelkomponenten notwendig, um feststellen zu können, in welche Klasse der aktuelle Attributwert fällt.

Betrachtet man zum Beispiel die Aussage „KOSTEN“ $>$ 450 000 genauer, kann man erkennen, dass sie aus zwei Teilen besteht. Zum einen aus dem Attribut, das aus der Datenbank geladen werden muss, und zum anderen aus der Regel, die angibt, wie und womit der Wert verglichen werden muss.

Die Abbildung 7.4 zeigt anhand dieses Beispiels, wie eine spezielle Hilfsmittelkomponente aufgebaut ist. Die Abfrage enthält zum einen den Namen des

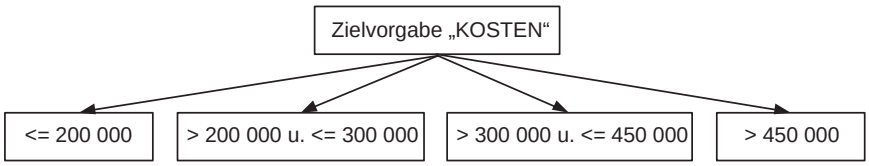



*Abb. 7.3:* Wertebereiche eines Bewertungskriteriums

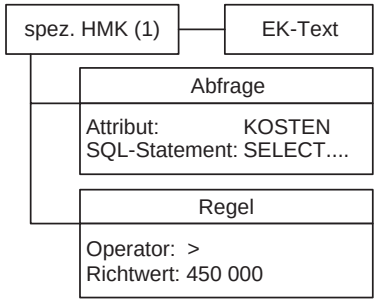



*Abb. 7.4:* Aufbau einer speziellen Hilfsmittelkomponente

Attributs, das aus der Datenbank geladen werden muss, und zum anderen das vollständige SQL-Statement mit dessen Hilfe das Attribut aus der Datenbank geholt werden kann. Die Regel hingegen enthält den Operator und den Richtwert. Somit sind alle Informationen vorhanden, die für die Ausführung der speziellen Hilfsmittelkomponente benötigt werden. Es fehlt nur noch das Ergebnis, das der Spieler erhält, wenn die Aussage „KOSTEN“ $>$ 450 000 zutrifft. Darum besitzt jede spezielle Hilfsmittelkomponente genau einen Erklärungstext, der dem Spieler angezeigt wird, wenn die Hilfsmittelkomponente zum aktuellen Zeitpunkt zutrifft. Der Text wurde deshalb mit der Hilfsmittelkomponente verknüpft, um für jeden Wertebereich, in dem der aktuelle Attributwert eines Bewertungskriteriums liegen kann, einen entsprechenden Erklärungstext angegeben zu können.

Die Erklärungskomponente benötigt demnach für das Analysieren von einfachen Bewertungskriterien wie das Einhalten der Projektkostengrenze oder das Erreichen der geforderten AFPs für den Code, nur eine SQL-Abfrage, einen Operator und einen Richtwert.
Mit Hilfe der SQL-Abfrage wird das ZARMS-Attribut aus der Datenbank geladen, der im Anschluss mit dem Richtwert aufgrund des Operators verglichen werden muss. Je nachdem, ob der Vergleich zutrifft oder nicht, kann eine Aussage über den Wert gemacht werden.

Die meisten Bewertungskriterien sind jedoch komplexer, das bedeutet, dass die Erklärungskomponente mehrere Werte untersuchen muss, um eine Aussage über ein Bewertungskriterium treffen zu können. In diesem Fall werden mehrere SQL-Abfragen, Richtwerte und Vergleichsoperatoren benötigt. Um auch diese Fälle abdecken zu können, musste eine Verkettung mehrer Hilfsmittelkomponenten-Teile, im Weiteren Instanzen genannt, ermöglicht werden. Eine Instanz steht für einen Basisfall einer speziellen Hilfsmittelkomponente, das heißt sie besteht aus einer Abfrage und aus einer Regel, wie schon in Abbildung 7.4 gezeigt wurde. Es muss jedoch an dieser Stelle darauf hingewiesen werden, dass die Reihenfolge, in der die einzelnen Instanzen abgearbeitet werden sollen, festgelegt werden muss. Warum das notwendig ist, werden wir bei der genaueren Betrachtung der Regelarten in Abschnitt 7.1.1 sehen. Durch die Bildung einer Sequenz von Instanzen, können auch komplexere Hilfsmittelkomponenten ausgewertet werden.
Betrachten wir zur Veranschaulichung noch einmal das Beispiel aus Abbildung 7.3, das eine Abstufung der Kosten in vier Klassen darstellt. Um das Bewertungskriterium auswerten zu können, muss festgestellt werden, in welche Klasse der aktuelle Attributwert fällt. Dazu wird jede dieser Klassen durch jeweils eine spezielle Hilfsmittelkomponente realisiert, die in Abbildung 7.5 dargestellt werden.

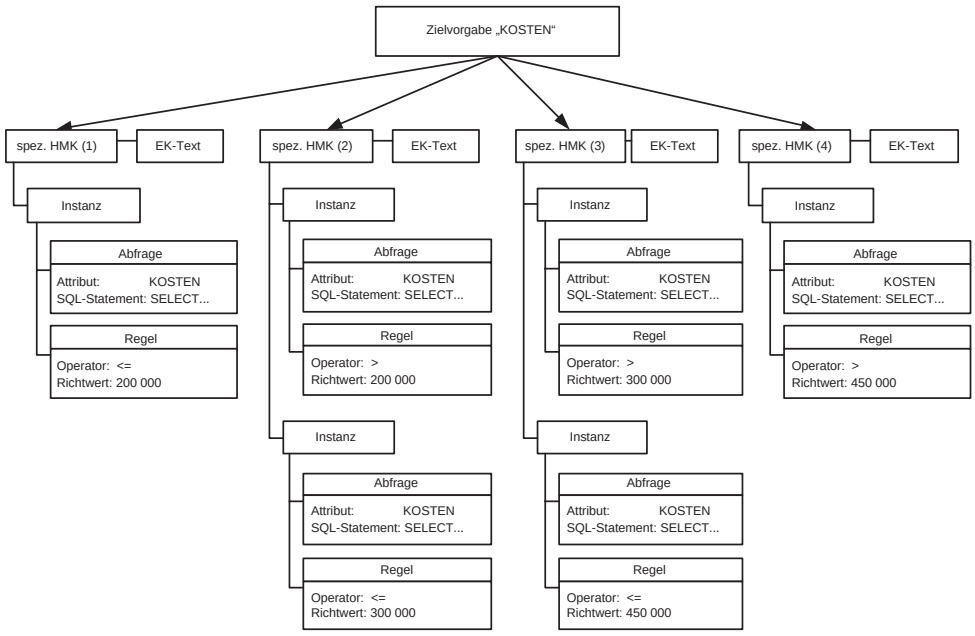



*Abb. 7.5:* Hilfsmittelkomponenten für das Bewertungskriterium „Kosten“

Die spez. HMK (1) zeigt einen Basisfall, das heißt eine Instanz mit zugehöriger Abfrage und Regel, während die spez. HMK (2) aus einer Sequenz von zwei Instanzen besteht, die prüfen, ob die Kosten zwischen der Untergrenze von 200 000 DM und der Obergrenze von 300 000 DM liegen.

Bei der Erstellung der speziellen Hilfsmittelkomponenten für die Auswertung der identifizierten Bewertungskriterien hat man erkannt, dass es mehrere Arten von Regeln gibt, die ebenfalls durch diese Architektur realisiert werden müssen. Diese Regelarten werden im folgenden Abschnitt genauer beschrieben.

### *7.1.1 Regelarten*

Bei der Auswertung von speziellen Hilfsmittelkomponenten können zum derzeitigen Entwicklungszeitpunkt drei Arten von Regeln unterschieden werden, die in diesem Abschnitt anhand von Beispielen verdeutlicht werden sollen.

#### *Instanz mit Regel, die Operator und Richtwert enthält*

Dabei handelt es sich um den einfachsten Fall. Der Wert, der mit Hilfe des SQL-Statements aus der Datenbank geholt wird, wird mit dem Operator und dem Richtwert zu einem Statement zusammengesetzt. Betrachten wir dazu ein Beispiel:

Abbildung 7.6 zeigt die speziellen Hilfsmittelkomponenten, die für die Auswertung des Bewertungskriteriums „Budget“ benötigt werden. Betrachten wir die erste Hilfsmittelkomponente (spez. HMK (1)) genauer. Der Erklärungstext, der dem Spieler angezeigt wird, sofern die spezielle Hilfsmittelkomponente zum aktuellen Zeitpunkt zutrifft, weist den Spieler darauf hin, dass dieser die Kosten überschritten hat. Die Instanzensequenz dieser speziellen Hilfsmittelkomponente besteht nur aus einer Instanz. Die Abfrage beinhaltet ein SQL-Statement, welches das Attribut „KOSTEN“ aus der Datenbank lädt. Das SQL-Statement ist in der Abbildung 7.6 aus Platzgründen nicht vollständig angegeben. Die Regel besitzt den Operator „>“ und den Richtwert 450 000.

Bei der Verarbeitung der speziellen Hilfsmittelkomponente wird wie folgt vorgegangen. Zuerst muss mit Hilfe des SQL-Statements aus der Abfrage das Attribut aus der Datenbank geladen werden. Im nächsten Schritt wird der erhaltene Wert, hier die aktuellen Kosten, mit dem Operator und dem

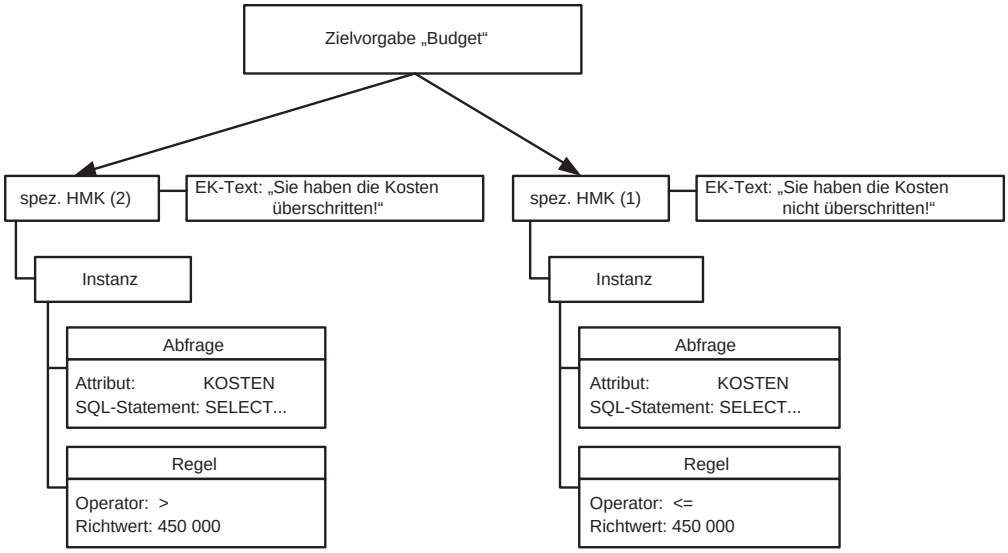



*Abb. 7.6:* Bewertungskriterium „Budget“

Richtwert aus der Regel zu einem neuen Statement zusammengesetzt und ausgewertet. Das Statement würde somit „KOSTEN $>$ 450 000“ lauten. Dieses Statement kann entweder wahr oder falsch sein. Liegen die aktuellen Kosten über 450 000 DM, dann ist das Statement wahr und die Auswertung wird bei der nächsten Instanz fortgesetzt. In unserem Fall gibt es nur eine Instanz, die ausgewertet werden muss. Ist diese Instanz wahr, dann bedeutet dies, dass die Hilfsmittelkomponente zum aktuellen Zeitpunkt zutrifft. Deshalb darf nun der Erklärungstext dieser Hilfsmittelkomponente dem Spieler angezeigt werden. Nach unserem Beispiel würde die Erklärungskomponente den Erklärungstext „Sie haben die Kosten überschritten!“ zurückgeben.
Sollten jedoch die aktuellen Kosten unter 450 000 DM liegen, das heißt das Statement „KOSTEN $>$ 450 000“ ist falsch, dann wird die Auswertung dieser Hilfsmittelkomponente sofort abgebrochen und die nächste spezielle Hilfsmittelkomponente für das Bewertungskriterium „Budget“ gesucht und ausgewertet.

In Abbildung 7.6 wird auch die zweite spezielle Hilfsmittelkomponente dargestellt. Sie deckt den positiven Fall des Bewertungskriteriums „Budget“ ab. Nachdem der Attributwert mit dem SQL-Statement aus der Datenbank geladen wurde, wird wieder ein Statement aus Attributwert, Operator und Richtwert zusammengebaut und überprüft. Hat der Spieler die Kostengrenze noch nicht überschritten, dann ist das Statement wahr und es wird der

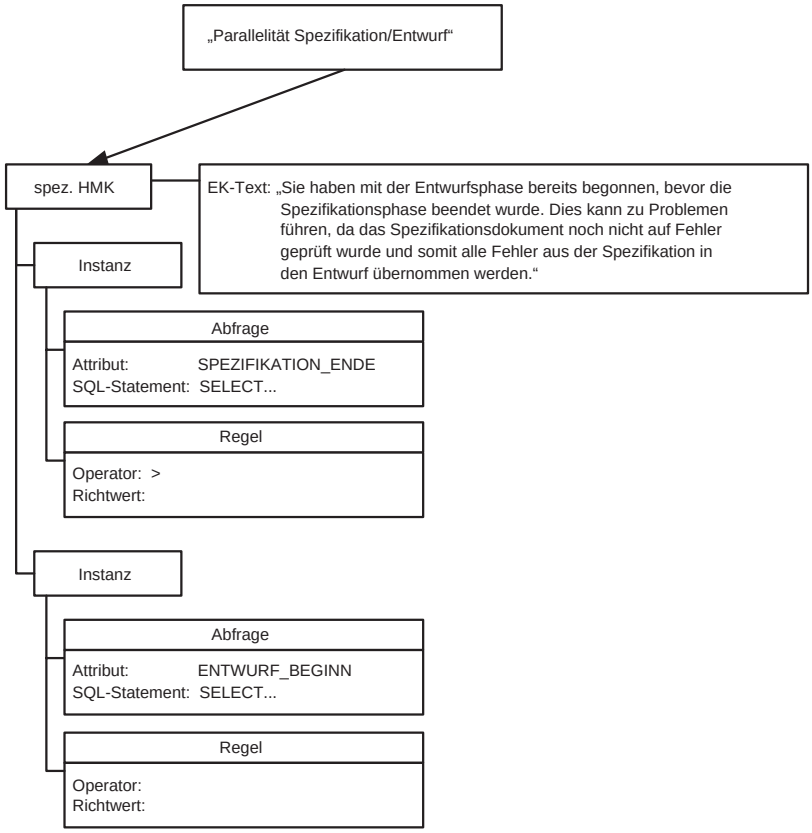



*Abb. 7.7:* Bewertungskriterium „Parallelität Spezifikation/Entwurf“

Erklärungstext „Kosten wurden noch nicht überschritten.“ ausgegeben.

*Instanz mit Regel, die nur einen Operator enthält*

Bei dieser Regelart wird der Attributwert der Instanz mit dem Operator der Regel und dem Attributwert aus der nächsten Instanz zu einem Statement zusammengesetzt.

Abbildung 7.7 zeigt eine spezielle Hilfsmittelkomponente, die dieser Regelart entspricht. Die Abarbeitung der Hilfsmittelkomponente erfolgt wie bereits beschrieben. Zuerst wird die erste Instanz verarbeitet. Dazu wird, mit Hilfe des dazugehörigen SQL-Statements, das Attribut „SPEZIFIKATION_ENDE“ aus der Datenbank geholt. Da die betreffende Regel nur aus einem Operator besteht, muss der Attributwert der ersten Instanz mit dem der zweiten Instanz verglichen werden. Anhand dieses Beispiels wird klar, wie wichtig

es ist, dass die Reihenfolge, in der die Instanzensequenz abgearbeitet wird, vorher festgelegt werden muss. Als Vergleichsoperator wird der Operator, der in der Regel der ersten Instanz enthalten ist, verwendet. Das zu überprüfende Statement lautet demnach: „SPEZIFIKATION_ENDE > ENTWURF_BEGINN“. Der Wert, den „SPEZIFIKATION_ENDE“ zum aktuellen Zeitpunkt aufweist sowie der Operator müssen in einem Vektor gespeichert werden. Im nächsten Schritt wird die zweite Instanz betrachtet. Zuerst wird das Attribut „ENTWURF_BEGINN“ aus der Datenbank geladen. Danach wird geprüft, ob es dazu eine Regel gibt. Da dies nicht der Fall ist, wird das Statement aus den Einträgen im Vektor („SPEZIFIKATION_ENDE“-Wert, „>“-Operator) und dem Wert des Attributs „ENTWURF_BEGINN“ zusammengesetzt. Die Erklärungskomponente überprüft anhand des Statements „SPEZIFIKATION_ENDE > ENTWURF_BEGINN“, ob eine Überlappung der Phasen Spezifikation und Entwurf vorliegt.

Natürlich sind auch Vermischungen der ersten beiden Regelarten denkbar. Da jede Instanz mit dem logischen UND-Operator mit der nächsten Instanz verbunden wird, müssen alle Instanzen wahr sein, damit die spezielle Hilfsmittelkomponente wahr ist. Es muss allerdings darauf geachtet werden, dass alle Fälle, die auftreten können, durch eine spezielle Hilfsmittelkomponente abgedeckt werden. Sonst wird die Bedingung verletzt, dass es immer eine Hilfsmittelkomponente gibt, die zu einem bestimmten Zeitpunkt für ein bestimmtes Bewertungskriterium zutrifft. Die Länge der Instanzketten ist unbegrenzt, allerdings muss an dieser Stelle darauf hingewiesen werden, dass mit der Anzahl der Instanzen einer speziellen Hilfsmittelkomponente auch die Anzahl der möglichen Permutationen steigt und die Auswertung eines Bewertungskriteriums dadurch durch eine Vielzahl von Hilfsmittelkomponenten realisiert werden muss, was zu einem erhöhten Verarbeitungsaufwand bei der Auswertung führt.

*Instanz ohne Regel*

Der Wert der Instanz wird bei dieser Regelart entweder mit dem Wert und dem Operator aus der vorherigen Instanz zu einem Statement zusammengesetzt oder es handelt sich um ein Diagramm. Der erste Fall wurde bereits im Abschnitt „Instanz mit Regel, die nur einen Operator enthält“ beschrieben. Bei Diagrammen hingegen werden die Werte, die für die Erstellung eines Diagramms notwendig sind, aus der Datenbank ausgelesen, dabei muss keine Regel angewandt werden.

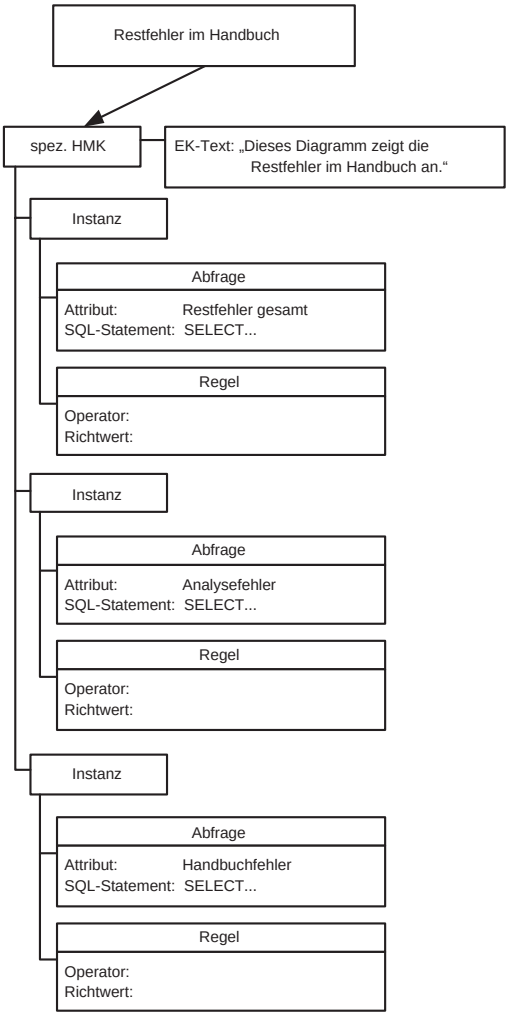



*Abb. 7.8:* Bewertungskriterium „Restfehler im Handbuch“

Auch das Zusammenfassen von Werten, die für die Erstellung eines Diagramms benötigt werden, wird durch die Architektur der speziellen Hilfsmittelkomponenten abgedeckt. Abbildung 7.8 zeigt die Hilfsmittelkomponente, die die Daten für das Diagramm „Restfehler im Handbuch“ aus der Datenbank lädt. Jede Instanz steht dabei für einen Balken im Diagramm. Das bedeutet, dass das Diagramm drei Balken (Restfehler gesamt, Analysefehler und Handbuchfehler) aufweisen wird. In diesem Fall benötigen die Instanzen keine Regeln sondern nur Abfragen. Die Werte für die einzelnen Balken werden zusammen mit der Attributbezeichnung in einem Vektor gespeichert.

Dieser Vektor wird später für die Erstellung des Diagramms benötigt. Auf die Bedeutung des Vektors und das Vorgehen bei der Erstellung der Diagramme wird in Kapitel 8 in Abschnitt 8.2 genauer eingegangen.

### *7.1.2 Ablauf*

In diesem Abschnitt wird der Verarbeitungsablauf der speziellen Hilfsmittelkomponenten beschrieben. Um ein bestimmtes Bewertungskriterium auswerten zu können, geht die Erklärungskomponente wie folgt vor.
Im ersten Schritt werden alle Hilfsmittelkomponenten, die zu diesem Bewertungskriterium gehören, gesucht und in einer Liste gespeichert.
Nehmen wir an, wir wollen das Bewertungskriterium „Anzahl AFPs im Code in Prozent“ auswerten. Dann findet das Auswertungstool zwei Hilfsmittelkomponenten („AFPs Code in Prozent“ - geforderte Grenze wurde nicht erreicht bzw. „AFPs Code in Prozent“ - geforderte Grenze wurde erreicht), die für die Erklärungskomponente dieses Kriteriums notwendig sind und speichert diese in einer Liste ab.
Diese Liste wird nun durchlaufen, wobei die Erklärungskomponente versucht, jede einzelne spezielle Hilfsmittelkomponente auszuwerten. In unserem Beispiel wird also die erste spezielle Hilfsmittelkomponente aus der Liste verarbeitet. Dazu werden die Daten, die für die Verarbeitung dieser Hilfsmittelkomponente notwendig sind, aus der Datenbank geladen. Zu diesen Daten zählen alle Instanzen und die dazugehörigen Regeln und Abfragen.
Die Erklärungskomponente sucht nun aus der Instanzenmenge die erste Instanz sowie deren Abfrage und Regel heraus und beginnt das SQL-Statement aus der Abfrage auszuwerten. Im Anschluss wird, wie bereits beschrieben, das Statement aus SQL-Wert, Operator und Richtwert zusammengesetzt und geprüft. In unserem Beispiel wird untersucht, ob das Statement „SQL-Wert $<$ 95“ zutrifft, das heißt es wird kontrolliert, ob die Grenze von 95 % der geforderten AFPs bereits erreicht wurde.
Ist das Statement wahr, dann wird die Nachfolgeinstanz gesucht und die Auswertung dort fortgesetzt. Das geht solange weiter, bis entweder ein Statement falsch ist, dann wird die Auswertung beendet, oder alle Instanzen geprüft wurden. Sind alle Instanzen wahr, dann kann die Erklärungskomponente den Erklärungstext dieser Hilfsmittelkomponente zurückgeben.
Wurde die Auswertung abgebrochen, da eine Instanz falsch war, dann wählt die Erklärungskomponente die nächste spezielle Hilfsmittelkomponente aus der Liste aus und beginnt mit der Auswertung von vorne. In unserem Beispiel würde nun der positive Fall „SQL-Wert $>=$ 95“ überprüft werden. Da es zu jedem Zeitpunkt nur genau eine zutreffende spezielle Hilfsmittelkomponente

geben darf, kann die Auswertung sofort beendet werden, wenn diese gefunden wurde.

### 7.1.3 Realisierung der Hilfsmittelkomponenten in der Datenbank

In der Datenbank wird die Architektur der speziellen Hilfsmittelkomponenten in vier Tabellen realisiert. Es muss an dieser Stelle angemerkt werden, dass die Hilfsmittelkomponenten nur für Instanzen eines Modells gelten, da sie auf den ZARMS-Daten basieren, die in jedem Modell anders aufgebaut sein können.

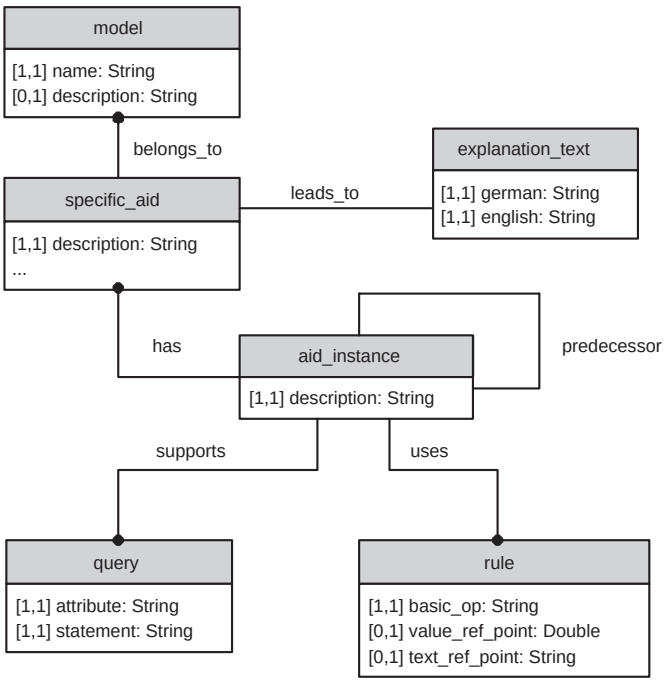



*Abb. 7.9:* Realisierung der Hilfsmittelkomponente in der Datenbank (Auszug aus dem Datenmodell)

Die Abbildung 7.9 stellt den Ausschnitt aus dem Datenmodell dar, der die spezielle Hilfsmittelkomponente enthält. Die eigentliche Hilfsmittelkomponente wird durch die Tabelle „specific_aid“ repräsentiert. Sie enthält neben einer eindeutigen Id auch die Beschreibung, die angibt, um welche Hilfsmittelkomponente es sich handelt. Jeder Hilfsmittelkomponente wird ein Erklärungstext (explanation_text) zugeteilt, der von der Erklärungskomponente

zurückgegeben wird, sofern die Hilfsmittelkomponente zum aktuellen Zeitpunkt zutrifft. Aus Abbildung 7.9 geht außerdem hervor, dass auch daran gedacht wurde, den Erklärungstext in mehreren Sprachen zu speichern. Diese Überlegung hängt mit der AMEISE-Spiel zusammen, das sowohl in Deutsch als auch in Englisch gespielt werden kann. Sollte der Spieler das Spiel in englischer Sprache durchführen, ist es sinnvoll, auch die Erklärungstexte der Auswertung in dieser Sprache anzuzeigen. Die Erweiterung um zusätzliche Sprachen kann durch das Hinzufügen zusätzlicher Attribute in der Tabelle „explanation_text“ einfach realisiert werden. Das Festhalten der Erklärungstexte in HTML stellt eine zusätzliche Erweiterungsmöglichkeit dar, die zum einen das Erscheinungsbild des Textes erheblich verbessert, und zum anderen das Einfügen von Images bzw. das Verweisen auf andere Bewertungskriterien über Links ermöglicht. Vor- bzw. Nachteile dieser Erweiterung werden im Moment diskutiert, eine Entscheidung liegt jenseits der Diplomarbeit.

Zu jeder Hilfsmittelkomponente gehören eine oder mehrere Instanzen (aid_instance), die durch eine Vorgängerliste zu einer Sequenz zusammengefasst werden. Die Reihenfolge, in der die Instanzen abgearbeitet werden müssen, wird durch diesen Vorgängereintrag festgelegt. Die erste Instanz ist die einzige, die keinen Vorgänger enthält. Bei dieser Instanz beginnt die Verarbeitung der speziellen Hilfsmittelkomponente. Jeder Instanz wird eine Abfrage (query) und gegebenenfalls auch eine Regel (rule) zugeteilt.
Die Abfrage enthält den Eintrag „attribute“, der das ZARMS-Attribut enthält, das mit dem SQL-Statement im Attribut „statement“ aus der Datenbank geladen wird. Die Regel hingegen hat Einträge für Operator und Richtwert, wobei es sich beim Richtwert um einen Text (text_ref_point) oder um einen Wert (value_ref_point) handeln kann.

Ein Text wird immer dann als Richtwert verwendet, wenn der Personaleinsatz des Spielers analysiert werden soll. Der Spieler hat während der Simulation des Projekts die Möglichkeit verschiedenen Entwicklern Aufgaben zuzuteilen. Er muss jedoch darauf achten, dass die Entwickler ihren Fähigkeiten entsprechend eingesetzt werden. Laut QS-Modell ist Richard der geeignetste Mitarbeiter für die Spezifikation. Das bedeutet, dass die Erklärungskomponente überprüfen kann, ob der Spieler Richard für die Erstellung des Spezifikationsdokuments eingesetzt hat. In diesem Fall muss die Hilfsmittelkomponente den/die Autor(en) der Spezifikation aus der Datenbank auslesen und mit dem Richtwert „Richard“ vergleichen. Sollte Richard einer der Autoren des Dokuments sein, dann hat der Spieler den Entwickler an der richtigen Stelle im Projekt eingesetzt. Hat er jedoch anderen Mitarbeitern diese Tätigkeit zugeteilt, dann kann die Erklärungskomponente dies feststellen und den Spieler

auf seinen Fehler hinweisen.
Der Richtwert kann sich auch aus einer Liste von mehreren Mitarbeitern zusammensetzen. Für das Design zum Beispiel ist sowohl Richard als auch Christine qualifiziert, der Richtwert entspricht dann der folgenden Liste von Richtwerten: „Richard, Christine“. Die Autoren des Entwurfs können somit mit dieser Liste verglichen werden, um den Personaleinsatz des Spielers analysieren zu können.

Für textuelle Richtwerte wurden im Moment folgende Operatoren realisiert:

- **„IN“:** vergleicht, ob der Richtwert bzw. ein Wert aus der Liste der Richtwerte in der Liste der ZARMS-Attribute enthalten ist, die aus der Datenbank geladen wurden
- **„NOT IN“:** prüft, ob keines der Element in der Liste der Richtwerte in der ZARMS-Attribut-Liste vorkommt
- **„ALL IN“:** überprüft, ob alle Elemente in der Liste der Richtwerte auch in der ZARMS-Attribut-Liste vorkommen
- **„==“:** vergleicht, ob beide Listen die gleichen Elemente enthalten
- **„!=“:** ermittelt, ob die beiden Listen unterschiedliche Elemente beinhalten

Sollte es sich bei dem Richtwert um einen numerischen Wert handeln, können folgende Operatoren verwendet werden:

- **„<=“:** prüft ob der ZARMS-Attributwert kleiner als der Richtwert oder gleich dem Richtwert ist
- **„<“:** untersucht, ob der ZARMS-Attributwert kleiner dem Richtwert ist
- **„>=“:** ermittelt, ob der ZARMS-Attributwert größer als der Richtwert oder gleich dem Richtwert ist
- **„>“:** kontrolliert, ob der ZARMS-Attributwert größer als der Richtwert ist
- **„==“:** überprüft, ob der ZARMS-Attributwert dem Richtwert entspricht
- **„!=“:** prüft, ob sich der ZARMS-Attributwert vom Richtwert unterscheidet

Eine Erweiterung der Operatorenliste kann jederzeit und mit minimalem Aufwand vorgenommen werden.

Jede spezielle Hilfsmittelkomponente gilt nur für die Instanzen eines Modells, da die ZARMS-Daten nur für ein bestimmtes Modell gelten. Darum findet sich auch die Beziehung zwischen den Tabellen „specific_aid“ und „model“ im Datenmodell wieder. Durch diese Beziehung kann gewährleistet werden, dass eine spezielle Hilfsmittelkomponente für ein bestimmtes Modell gültig ist.

Fassen wir an dieser Stelle die Vorteile dieses Ansatzes nochmals zusammen:

- **die speziellen Hilfsmittelkomponenten sind generisch**
  Die Architektur der speziellen Hilfsmittelkomponente kann für alle bestehenden sowie für alle zukünftigen Modelle verwendet werden. Die bestehenden Modelle stellen alle Erweiterungen (Instanzen) des QS-Modells dar und verwenden deshalb, abgesehen von einigen Erweiterungen, dieselben ZARMS-Daten. Deshalb können für diese Modellinstanzen die bisher entwickelten speziellen Hilfsmittelkomponenten übernommen werden. Dies geschieht durch eine einfache Zuweisung der bestehenden Hilfsmittelkomponenten zu der neuen Modellinstanz. Möglicherweise müssen einige Hilfsmittelkomponenten um zusätzliche Instanzen erweitert werden, um die neuen Effekte auswerten zu können. Ist dies der Fall, dann müssen neue Hilfsmittelkomponenten entwickelt werden.
  Wird ein neues Modell verwendet, das auf anderen ZARMS-Daten beruht, dann müssen für die Bewertungskriterien dieses Modells auch neue Hilfsmittelkomponenten erstellt werden.

- **Abfragen und Regeln sind mehrfach verwendbar**
  Da es zu jedem Bewertungskriterium mehrere spezielle Hilfsmittelkomponenten geben muss, müssen häufig die gleichen Abfragen an die Datenbank gestellt werden. Abfragen stellen jedoch eigenständige Teile der speziellen Hilfsmittelkomponente dar und können daher für verschiedene Hilfsmittelkomponenten eingesetzt werden. Auch Regeln können, sofern sie auch für eine andere Hilfsmittelkomponente zutreffen, mehrfach verwendet werden.

- **die speziellen Hilfsmittelkomponenten sind leicht erweiterbar**
  Damit ein neues Bewertungskriterium ausgewertet werden kann, müssen neue Hilfsmittelkomponenten erstellt werden. Diese Erweiterungen können rasch vorgenommen werden, sofern die Instanzenreihenfolge

und die Regeln bzw. die für die Regel notwendigen Attribute durchdacht sind. Da die Hilfsmittelkomponenten aus unabhängigen Teilen bestehen, kann eine Erweiterung auch schrittweise durchgeführt werden. Das Erstellen von speziellen Hilfsmittelkomponenten wird im Anhang C genauer beschrieben.
Auch das Hinzufügen von neuen Operatoren für Regeln kann mit minimalem Aufwand vorgenommen werden, da diese Erweiterungen keine Auswirkungen auf die bereits bestehenden speziellen Hilfsmittelkomponenten haben.

- **Diagramme**
  Nicht nur die Ausgabe von Erklärungstexten, sondern auch die Verwendung der Daten für die Erstellung von Diagrammen, wird durch diesen Ansatz ermöglicht.

In diesem Kapitel wurde gezeigt, wie die Bewertungskriterien mit Hilfe von speziellen Hilfsmittelkomponenten durch die generische Erklärungskomponente ausgewertet werden können. Im nächsten Kapitel soll gezeigt werden, wie die Auswertungsergebnisse dem Spieler präsentiert werden, und wie die Daten zuvor aufbereitet werden müssen. Außerdem werden die bereits realisierten Bewertungskriterien vorgestellt und mögliche Erweiterungen diskutiert.

*Kapitel 8*

# AUSWERTUNG VON PROJEKTVERLÄUFEN IN AMEISE

In diesem Kapitel wird gezeigt, wie die in Kapitel 5 identifizierten Bewertungskriterien mit Hilfe der Hilfsmittelkomponenten, die in Kapitel 7 beschrieben wurden, umgesetzt werden, um dem Spieler eine möglichst umfangreiche Auswertung des Projektverlaufs zu geben.

## *8.1 Funktionalität der Auswertungsbenutzeroberfläche*

In diesem Abschnitt wird beschrieben, wie die Auswertungsergebnisse der generischen Erklärungskomponente am Client angezeigt werden, damit der Spieler möglichst schnell alle relevanten Informationen über den Projektverlauf finden kann.

Die Erklärungskomponente hat die Aufgabe am Ende eines Spiels eine Analyse des Spielverlaufs vorzunehmen. Sie wurde jedoch so entwickelt, dass sie auch während des Spiels eingesetzt werden kann. Da Aussagen über Fehlentscheidungen schon während des Spiels möglich sind, stellt sich die Frage, ob es auch sinnvoll ist eine Zwischenauswertung durchzuführen. Die Antwort liegt im Auge des Tutors.
Die Auswertung einiger Bewertungskriterien wie zum Beispiel der Vollständigkeit oder der Korrektheit der bereits erstellten Dokumente, könnten hilfreich für den Spieler sein, da er die Hinweise aus den Erklärungstexten bereits im weiteren Spiel umsetzten kann. Untersucht der Spieler zum Beispiel bei einer Zwischenauswertung die Effektivität seiner bisherigen Reviews, so kann er feststellen, ob er geeignete Gutachter eingesetzt hat und gegebenenfalls die Zusammensetzung der Reviewteams ändern.
Die Analyse anderer Bewertungskriterien hingegen macht nur am Ende des Spiels Sinn, wie es zum Beispiel bei der „Aufwandsverteilung über alle Phasen“ der Fall ist. Der Tutor kennt aber die einzelnen Bewertungskriterien und kann die Spieler vor Beginn des Spiels darauf hinweisen, welche Zwischenauswertungen hilfreich sein könnten.

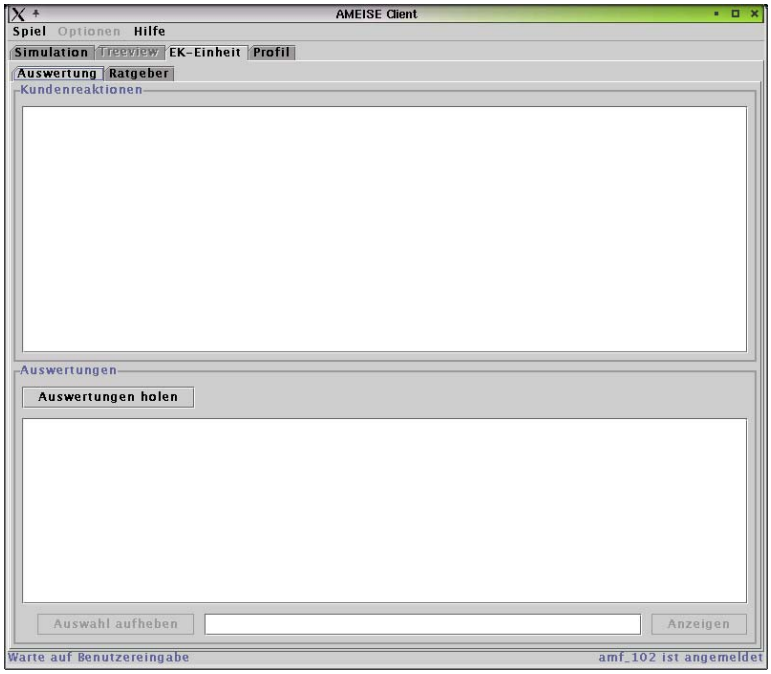



*Abb. 8.1:* Screenshot des Auswertungsfensters

Die Benutzeroberfläche der generischen Erklärungskomponente ist im Reiter EK-Einheit im Client eingebettet. Die Abbildung 8.1 zeigt das Auswertungsfenster hinter dem sich die generischen Erklärungskomponente verbirgt. Im oberen Feld werden nach einer Beendigung des Spiels die Kundenreaktionen angezeigt. Das Feld ist noch leer, da wir die Auswertung während der Simulation des Projekts durchführen, dass heißt, bevor das System an den Kunden übergeben wurde. Das untere Fenster ist im Moment noch leer. Durch Betätigen des Buttons „Auswertungen holen“ werden alle Bewertungskriterien geladen, die von der generischen Erklärungskomponente analysiert werden können.

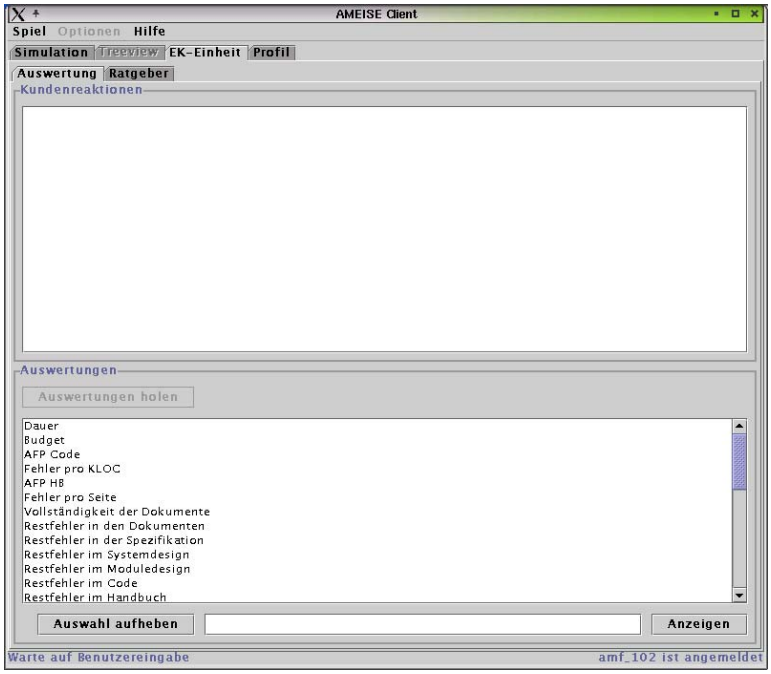



*Abb. 8.2:* Mögliche Auswertungen zum aktuellen Zeitpunkt

Abbildung 8.2 zeigt diese Situation. Der Spieler kann nun ein Bewertungskriterium auswählen. Dies ist in Abbildung 8.3 zu sehen. Durch die Betätigung des Buttons „Anzeigen“ wird die generische Erklärungskomponente gestartet.

Die generische Erklärungskomponente wertet nun das ausgewählte Bewertungskriterium aus. Dabei geht man, wie in Abschnitt 7.1.2 ausführlich beschrieben wurde, vor. Die Ergebnisse der Erklärungskomponente werden dem Spieler entweder nur in Form eines Erklärungstextes oder durch eine Kombination aus Diagramm und Erklärungstext präsentiert.

Abbildung 8.4 stellt eine Auswertung des Bewertungskriteriums „Budget“ dar, für das nur ein Erklärungstext zur Verfügung steht.

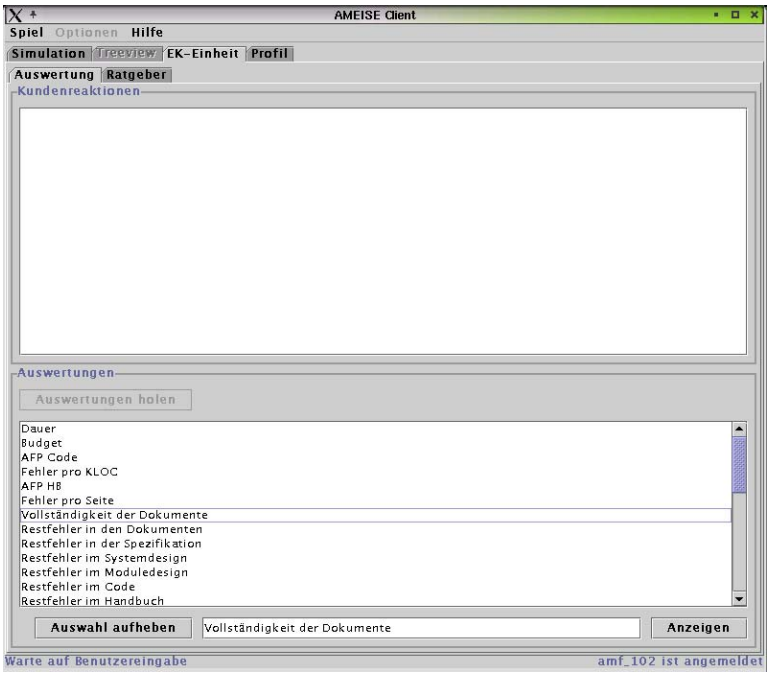



*Abb. 8.3:* Auswertung getroffen

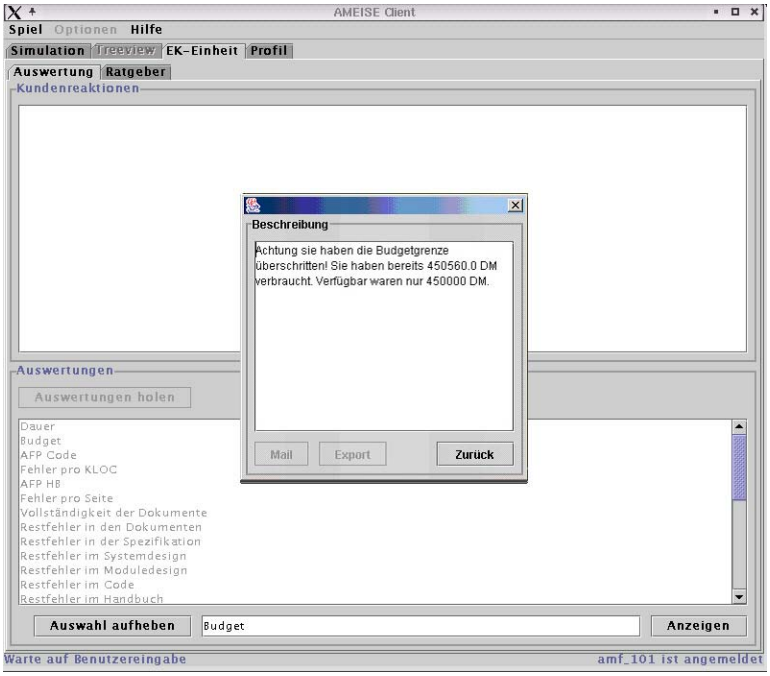



*Abb. 8.4:* Beispiel einer Auswertung ohne Diagramm

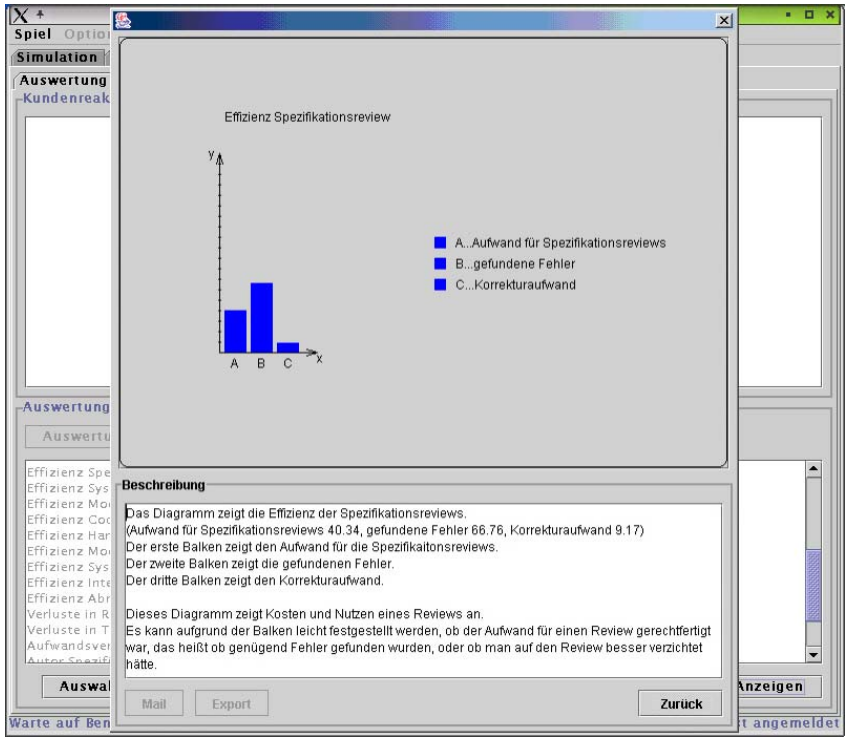



*Abb. 8.5:* Beispiel einer Auswertung mit Diagramm

Im Gegensatz dazu wird in Abbildung 8.5 ein Diagramm angezeigt, dass die Effizienz von Spezifikationsreviews darstellt. Der Erklärungstext im unteren Fenster beschreibt kurz, wie das Diagramm zu deuten ist und weist den Spieler darauf hin, welche Werte er genauer betrachten muss und worauf diese hinweisen können. Durch Betätigen des Buttons „Zurück“ kann der Spieler wieder zur Liste der Bewertungskriterien gelangen[1].

## 8.2 Aufarbeitung der Auswertungsergebnisse am Client

Nachdem der Spieler ein Bewertungskriterium aus der Liste ausgewählt hat, beginnt die generische Erklärungskomponente die dazugehörigen speziellen

[1] Der Button „Mail“ sowie der Button „Export“ wurden noch nicht realisiert und sind darum ausgegraut. Der „Mail“-Button soll die Ergebnisse als Mail verschicken, während der „Export“-Button dem Spieler ermöglichen soll, die Auswertungsergebnisse in andere Programme wie zum Beispiel Microsoft Excel, zu exportieren. Da aber im Moment an der Entwicklung eines Diagrammviewers gearbeitet wird, der sich genauer mit diesen Fragen beschäftigt, wurde vorerst auf die Realisierung dieser Punkte verzichtet.

Hilfsmittelkomponenten aus der Datenbank zu laden und auszuwerten. Während der Auswertung merkt sich die Erklärungskomponente die Bezeichnung sowie die Werte aller Attribute, die in der Instanzenkette ausgewertet werden, in einem Vektor, der zum Beispiel wie folgt aussehen könnte: [Restfehler Spezifikation, 45, Restfehler Entwurf, 66, Restfehler Code 89, Restfehler Handbuch, 56]. Sollte die Hilfsmittelkomponente nicht zutreffen, dann wird dieser Vektor wieder geleert. Damit kann gewährleistet werden, dass der Vektor nach Beendigung der Auswertung die Werte für die zutreffende Hilfsmittelkomponente enthält. Dieser Vektor wird zusammen mit dem Erklärungstext der zutreffenden speziellen Hilfsmittelkomponente an den Client übermittelt. Dort wird der Vektor zum einen zum Erstellen der Diagramme verwendet, und zum anderen für die Erweiterung des Erklärungstextes.
Für das Erzeugen des Diagramms wird der Vektor durchlaufen, wobei jedes Bezeichnung-Wert-Tupel durch einen Balken dargestellt wird. Eine ansprechendere Darstellung der Diagramme, dazu zählen zum einen unterschiedliche Farben für Werte aus verschiedenen Bereichen, zum anderen eine genaue Beschriftung der Achsen, soll durch die Entwicklung eines Diagrammviewers realisiert werden.
Um die aktuellen Werte in den Erklärungstext einfließen lassen zu können, durchläuft der Client den Erklärungstext und fügt, sofern dieser das Zeichen „*“ enthält, den Vektor an dieser Stelle ein, um dem Spieler die genauen Werte der Attribute an einer geeigneten Stelle im Erklärungstext anzeigen zu können. In Abbildung 8.5 taucht dieser Vektor durch die Erweiterung des Erklärungstextes in der zweiten Zeile auf.

## *8.3 Visualisierung der Auswertungsergebnisse am Client*

Dieser Abschnitt beschreibt die grafischen Auswertungen der Bewertungskriterien, die bereits realisiert wurden bzw. noch realisiert werden. Diese Diagramme visualisieren dem Spieler positive und negative Aspekte seines Projektverlaufs. Die Erklärungstexte können dem Spieler dabei helfen, aus den Diagrammen mögliche Auswirkungen von guten bzw. schlechten Entscheidungen zu erkennen.

## *8.4 Realisierte Bewertungskriterien*

Die realisierten Bewertungskriterien werden im Folgenden zu Bereichen zusammengefasst, die ähnliche Kriterien abdecken.

*Zielvorgaben*

Für dieses Bereich wurden bereits folgende Diagramme realisiert:

- Projektdauer
- Budget
- realisierte AFPs im Code in Prozent
- Fehler pro KLOC im Code
- realisierte AFPs im Handbuch in Prozent
- Fehler pro Seite im Handbuch

Bei den Zielvorgaben handelt es sich um die wichtigsten Bewertungskriterien, da den Spieler unmittelbar nach Beendigung des Projekts interessiert, ob er die vorgegebenen Grenzen (Dauer und Budget) eingehalten hat und trotzdem die geforderten AFPs für Code und Handbuch realisieren konnte. Die Erklärungskomponente stellt darum für jede Zielvorgabe eine textuelle Auswertung dar. Die Abbildung 8.6 zeigt die Auswertung der AFPs für den Code in Prozent.

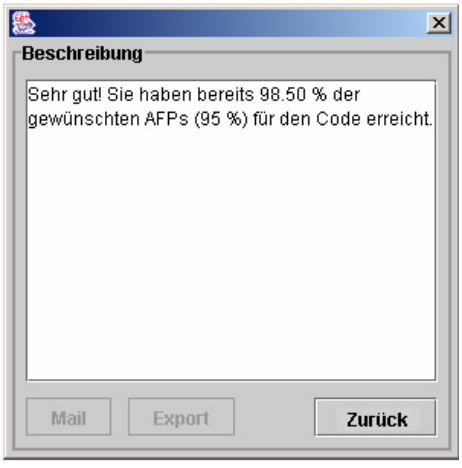



*Abb. 8.6:* Erreichte AFPs für den Code

*Vollständigkeit der Dokumente*

Dieses Bewertungskriterium gibt einen Überblick über die Vollständigkeit der Dokumente (Spezifikation, Systemdesign, Moduldesign, Code, Handbuch) und wird mit Hilfe eines Diagramms dem Spieler repräsentiert.

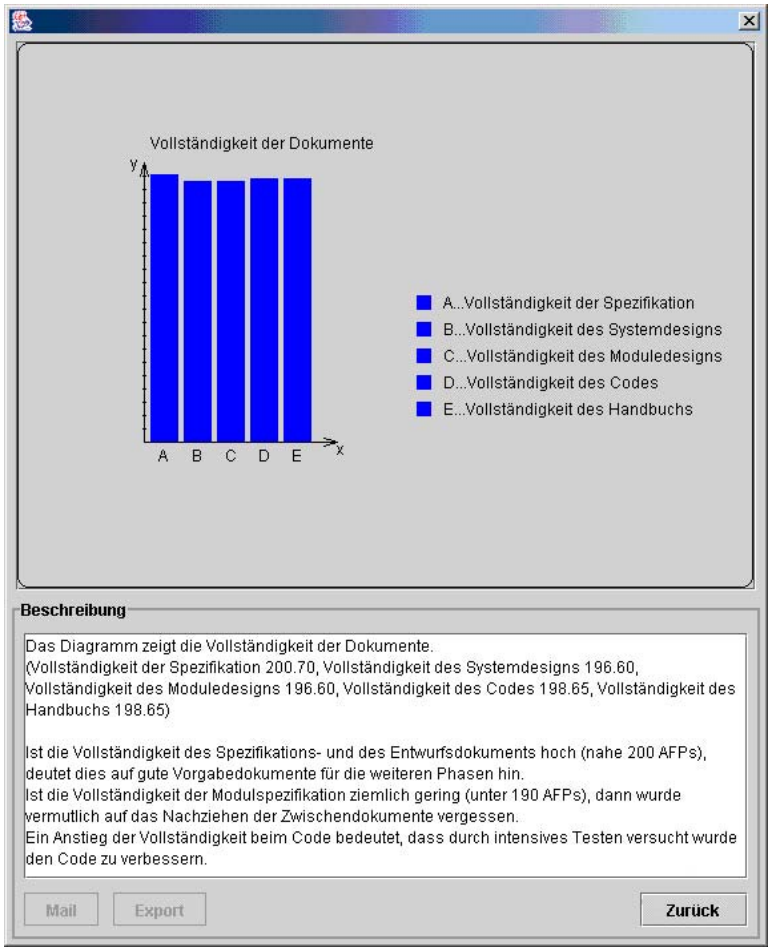



*Abb. 8.7:* Vollständigkeit der Dokumente

Das Auswertungsbeispiel in Abbildung 8.7 zeigt einen typischen Verlauf. Die Vollständigkeit der Dokumente nahm bis hin zum Moduldesign ständig ab, während sie beim Code wieder anstieg. Im entsprechenden Erklärungstext werden zusätzlich zu den genauen Werten der Balken im Diagramm mögliche Aspekte beschrieben, auf die das Diagramm hinweist. Ist die Vollständigkeit der Dokumente in den ersten Phasen sehr hoch, dann deutet dies auf korrek-

te Vorgabedokumente hin. Im Beispiel in Abbildung 8.7 ist dies nur bei der Spezifikation der Fall. Das bedeutet, dass das Spezifikationsdokument eine gute Vorgabe für die weiteren Phasen war. Der Anstieg der Vollständigkeit hin zum Code ist klassisch und weist darauf hin, dass der Spieler intensiv getestet hat, um die Zielvorgaben für den Code noch erfüllen zu können. Die Hinweise, die das Diagramm gibt, müssen jedoch vom Spieler mit den jeweiligen Bewertungskriterien überprüft werden. Auf weitere Bewertungskriterien kann bereits an dieser Stelle im Erklärungstext hingewiesen werden.

*Restfehler der Dokumente*

Ein weiteres wichtiges Bewertungskriterium gibt an, wie viele Fehler die Dokumente nach Beendigung des Projekts noch enthalten.

Folgende Bewertungskriterien wurden schon realisiert:

- Restfehler der Dokumente
- Restfehler in der Spezifikation
- Restfehler im Systemdesign
- Restfehler im Moduldesign
- Restfehler im Code
- Restfehler im Handbuch

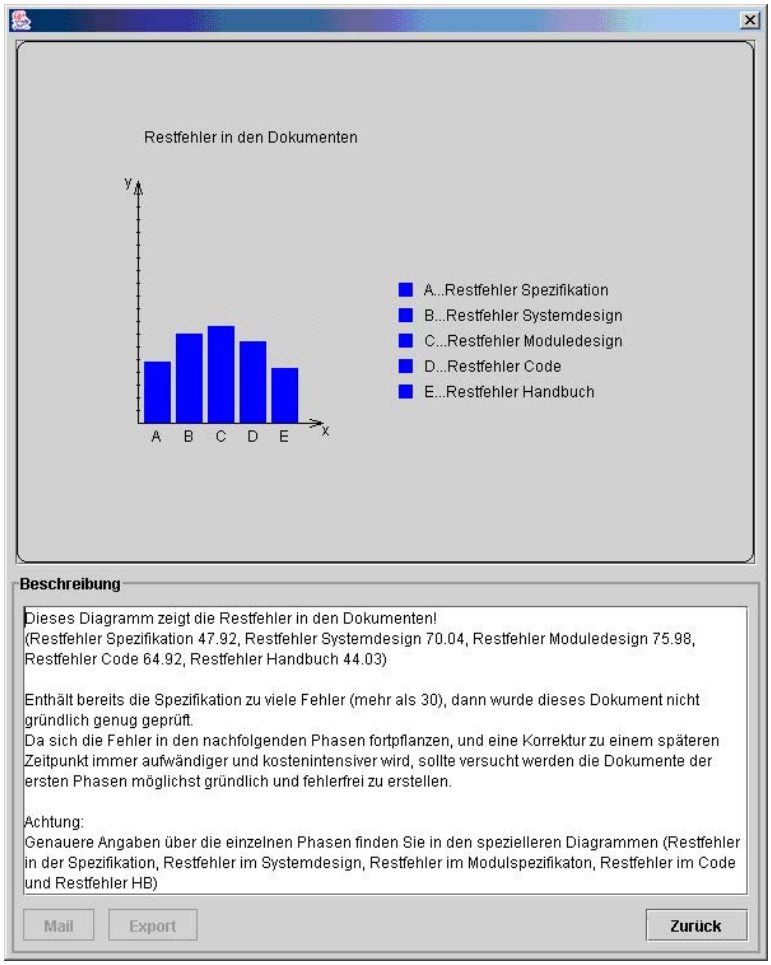



*Abb. 8.8:* Restfehler der Dokumente

In Abbildung 8.8 kann man erkennen, dass die Fehler mit jeder Phase zunahmen, da in jeder Phase neue Fehler hinzugefügt wurden. In der Codierungsphase wurde versucht, durch Testen möglichst viele der Fehler zu finden und zu korrigieren, die während der vorherigen Phasen eingefügt wurden. Im folgenden Erklärungstext wird darauf hingewiesen, dass das Spezifikationsdokument möglichst wenige Fehler enthalten soll, da sich diese Fehler während des Projekts fortpflanzen. Ein höherer Aufwand in den frühen Phasen kann somit zu korrekteren Dokumenten führen und erspart dem Spieler Kosten und Zeit, die beim Testen des Codes entstehen. Außerdem wird der Spieler im Erklärungstext darauf hingewiesen, dass es speziellere Diagramme zu den Restfehlern in den einzelnen Dokumenten gibt.

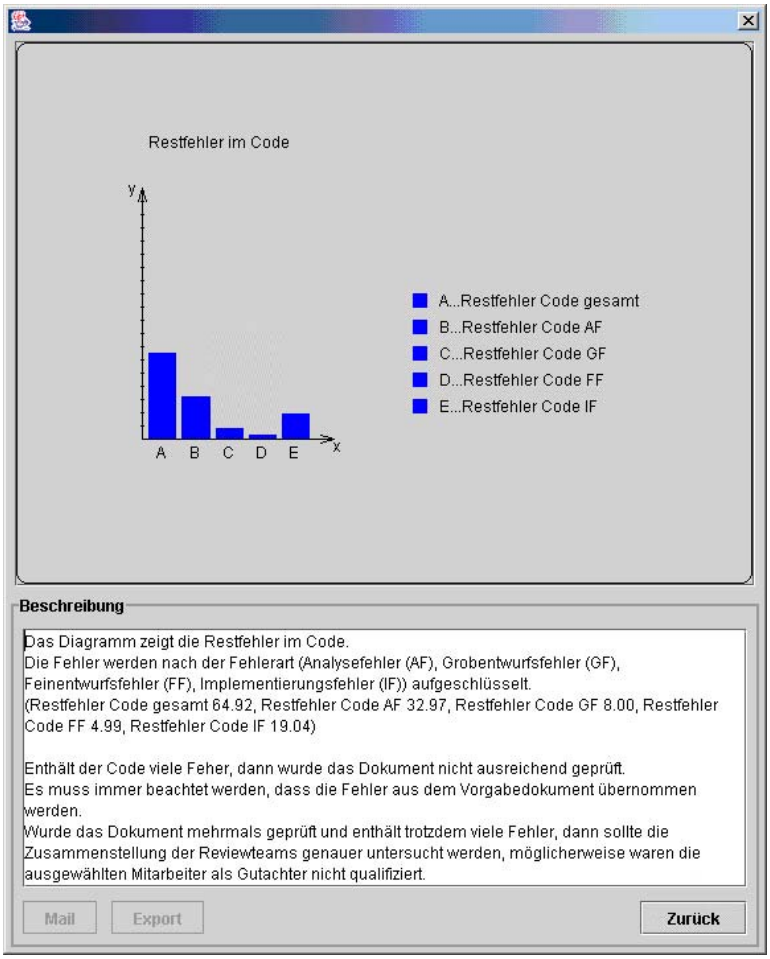



*Abb. 8.9:* Restfehler im Code

Als Beispiel für ein spezielleres Diagramm werden in der Abbildung 8.9 die Restfehler des Codes aufgeschlüsselt nach Fehlerart dargestellt. Man kann erkennen, dass der Code noch einige Analysefehler enthält, während nur wenige Designfehler übernommen wurden. Sollte man während des Testens Analysefehler entdecken, dann ist die Korrektur kostenintensiv und sehr aufwändig. Da der Code dieses Spielers in diesem Beispiel nur mehr 19 Implementierungsfehler enthält, hat er es ausreichend geprüft und dafür auch die richtigen Mitarbeiter als Gutachter eingesetzt.

*Effizienz von Reviews bzw. Tests*

Dieses Bewertungskriterium stellt den Aufwand für Reviews bzw. Tests, die gefundenen Fehler und den Aufwand für die Korrektur dieser Fehler gegenüber. Mit Hilfe dieses Diagramms kann auf einen Blick festgestellt werden, ob ein Review bzw. Test effizient durchgeführt werden konnte oder nicht.

Die Diagramme der folgenden Bewertungskriterien wurden bereits realisiert:

- Effizienz von Spezifikationsreviews
- Effizienz von Systemdesignreviews
- Effizienz von Moduldesignreviews
- Effizienz von Codereviews
- Effizienz von Handbuchreviews
- Effizienz von Modultests
- Effizienz von Systemtests
- Effizienz von Integrationstests
- Effizienz von Abnahmetests
  Zeigt jedoch nur, ob der Spieler die vom Kunden gefundenen Fehler effizient korrigiert hat. Der Aufwand des Abnahmetests und die dabei gefundenen Fehler können lediglich Auskunft darüber geben, ob der Kunde effizient gearbeitet hat.

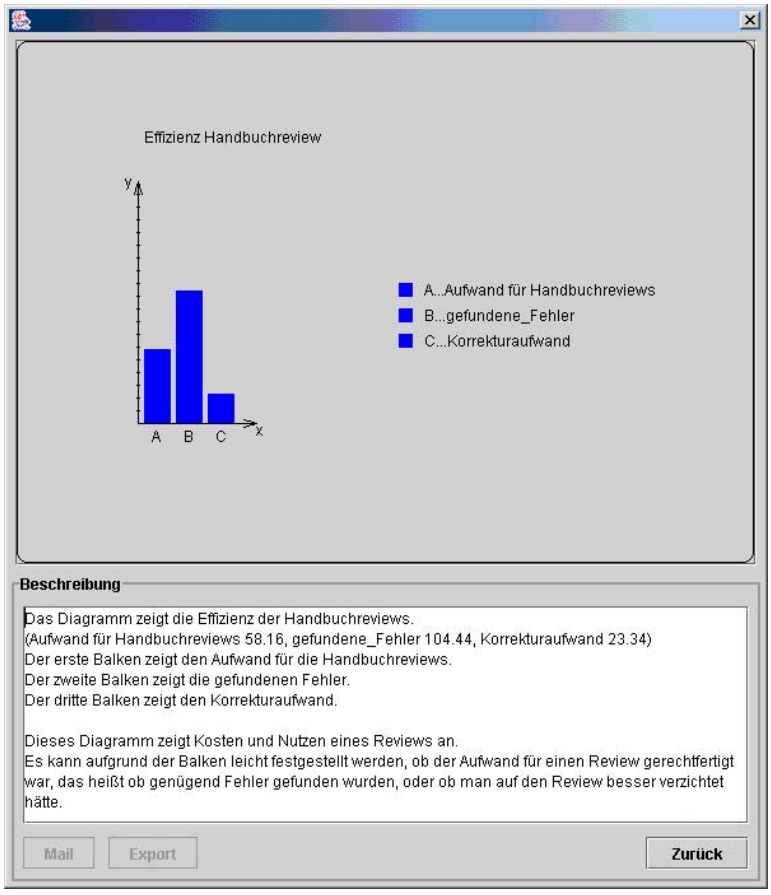



*Abb. 8.10:* Effizienz der Handbuchreviews

Die Abbildung 8.10 zeigt die Auswertung eines Spiels, bei dem sehr effiziente Handbuchreviews durchgeführt wurden. Es muss allerdings angemerkt werden, dass für die Darstellung der Effizienz nicht einzelne Handbuchreviews ausgewählt werden können, sondern dass die Daten durch die Summe über alle durchgeführten Handbuchreviews ermittelt werden. Es ist zwar möglich die Effizienz von Reviews auch auf der Ebene einzelner Reviews darzustellen, dies wurde jedoch zum aktuellen Zeitpunkt noch nicht realisiert. Wurde in den Handbuchreviews sehr effizient gearbeitet, so kann dies auch auf die Teilnahme des Kunden an einem der Reviews hinweisen.

Die Diagramme, die die Effizienz von Tests zeigen sollen, sind auf dieselbe Art und Weise aufgebaut. Sie enthalten ebenfalls die drei Balken (Aufwand Test, gefundene Fehler und Aufwand Korrektur).

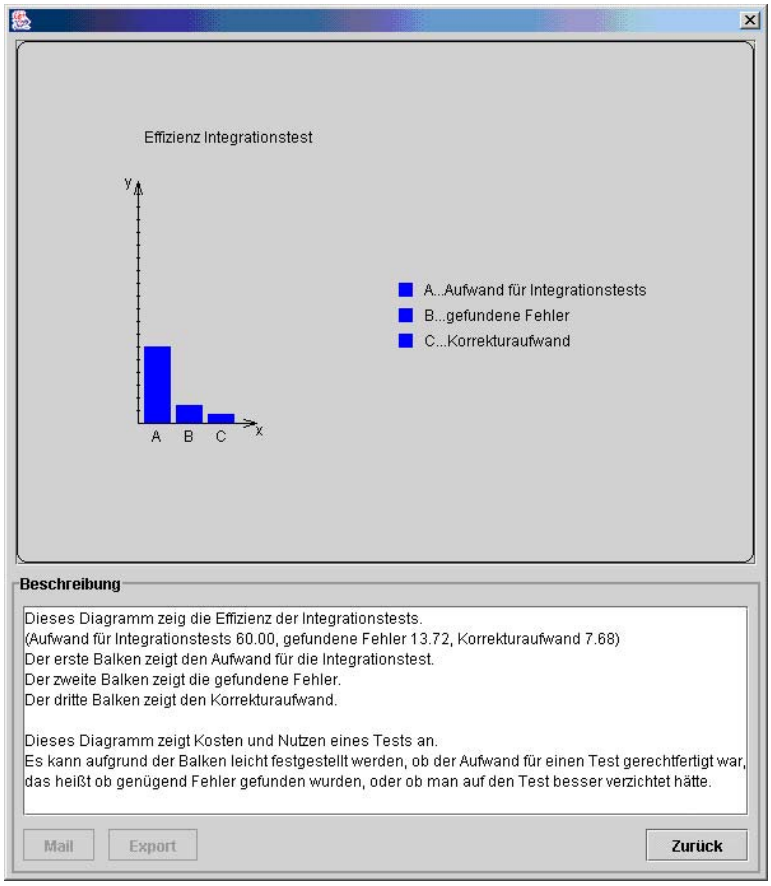



*Abb. 8.11:* Effizienz von Integrationstests

Abbildung 8.11 zeigt, dass der Spieler den Integrationstest nicht effizient durchgeführt hat. Der Grund dafür kann anhand dieses Diagramms jedoch nicht festgestellt werden.

### *Effektivität von Reviews*

Dieses Bewertungskriterium stellt die durch Reviews gefundenen Fehler den im Dokument enthaltenen Fehlern gegenüber. Dadurch kann festgestellt werden, ob der Spieler die richtigen Gutachter für die Reviews eingesetzt hat, das

heißt, wie viele der enthaltenen Fehler von den Gutachtern gefunden wurden.

Für dieses Bereich wurden bereits die folgenden Auswertungen entwickelt:

- Effektivität der Spezifikationsreviews
- Effektivität der Systemdesignreviews
- Effektivität der Moduldesignreviews
- Effektivität der Codereviews
- Effektivität der Handbuchreviews

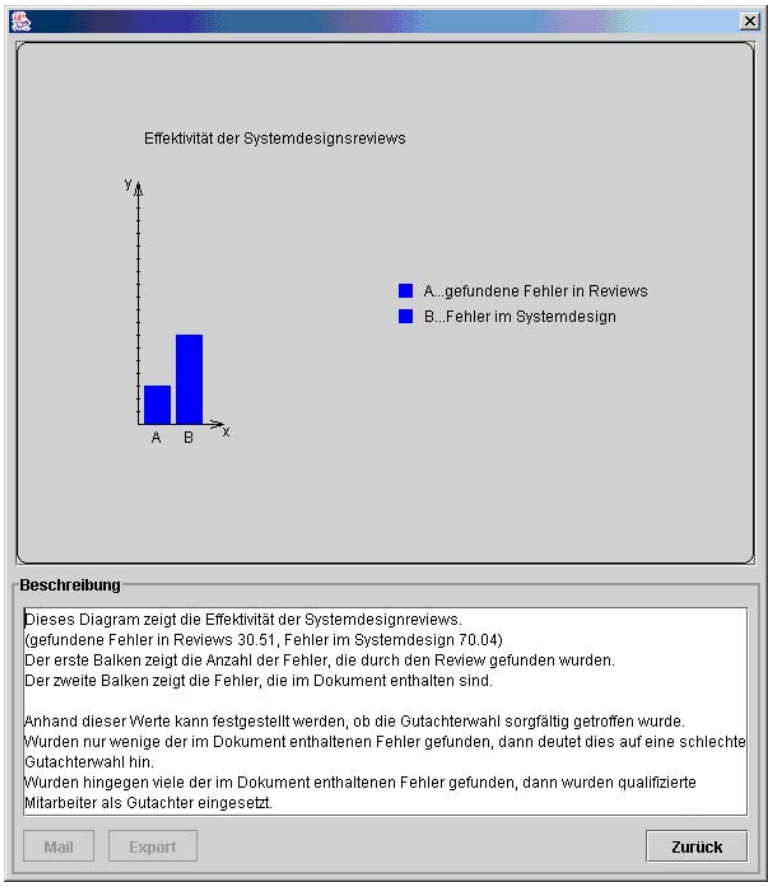



*Abb. 8.12:* Effektivität der Systemdesignreviews

Die Auswertung in Abbildung 8.12 stellt die Effektivität der Systemdesign-Reviews dar. Die Gutachter haben hier nicht einmal die Hälfte der enthalte-

nen Fehler im Dokument gefunden. Der Spieler sollte in diesem Fall die Zusammenstellung seiner Gutachter überprüfen, die er in den Systemdesignreviews eingesetzt hat. Es muss hier darauf hingewiesen werden, dass in dem Diagramm die Daten aus allen durchgeführten Systemdesignreviews zusammengefasst und angezeigt werden. Eine feingranularere Darstellung der Effektivität der einzelnen Systemdesignreviews wird noch realisiert.

*Verluste durch Reviews bzw. Tests*

Dieses Bewertungskriterium zeigt die Anzahl der AFPs auf, die durch Reviews und Tests verloren gingen. Realisiert wurden sowohl die Auswertung, die die Verluste durch Reviews darstellt als auch die, die Verluste durch Tests aufzeigt.

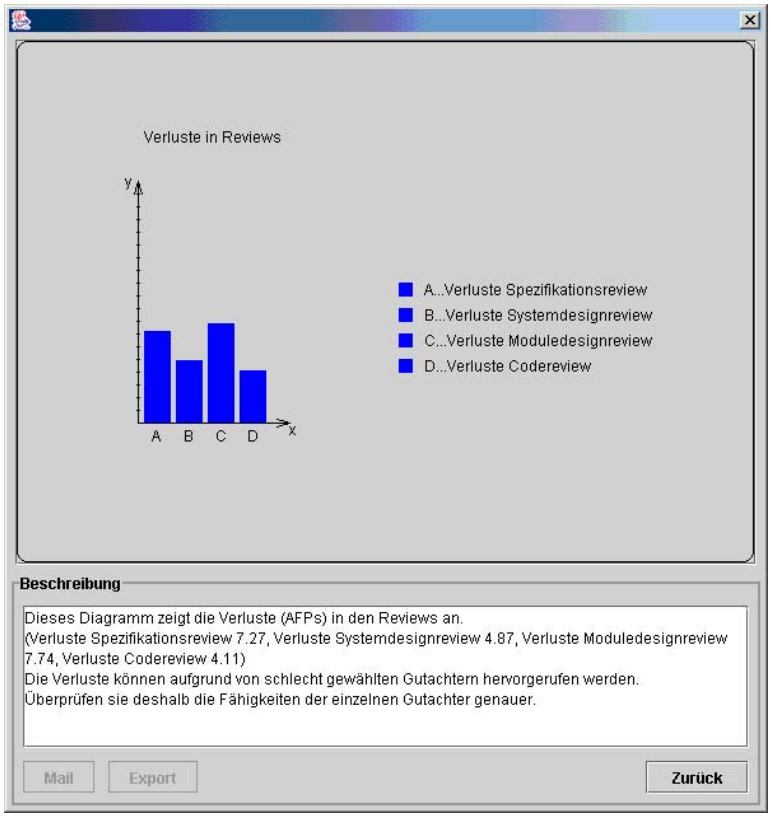



*Abb. 8.13:* Verluste durch Reviews

In Abbildung 8.13 wird die Auswertung der Verluste, die durch Reviews entstanden sind, gezeigt. In den Spezifikationsreviews sowie in den Moduldesignreviews gingen 7 AFPs verloren. Dieser Verlust kann durch eine schlechte Gutachterwahl entstanden sein. Der Spieler wird darum im Erklärungstext darauf hingewiesen, die Qualifikation der Gutachter der einzelnen Reviews zu kontrollieren.

*Aufwandsverteilung*

Dieses Bewertungskriterium zeigt die Verteilung des Aufwands über die Phasen: Spezifikation, Entwurf, Codierung, Test und Handbucherstellung.

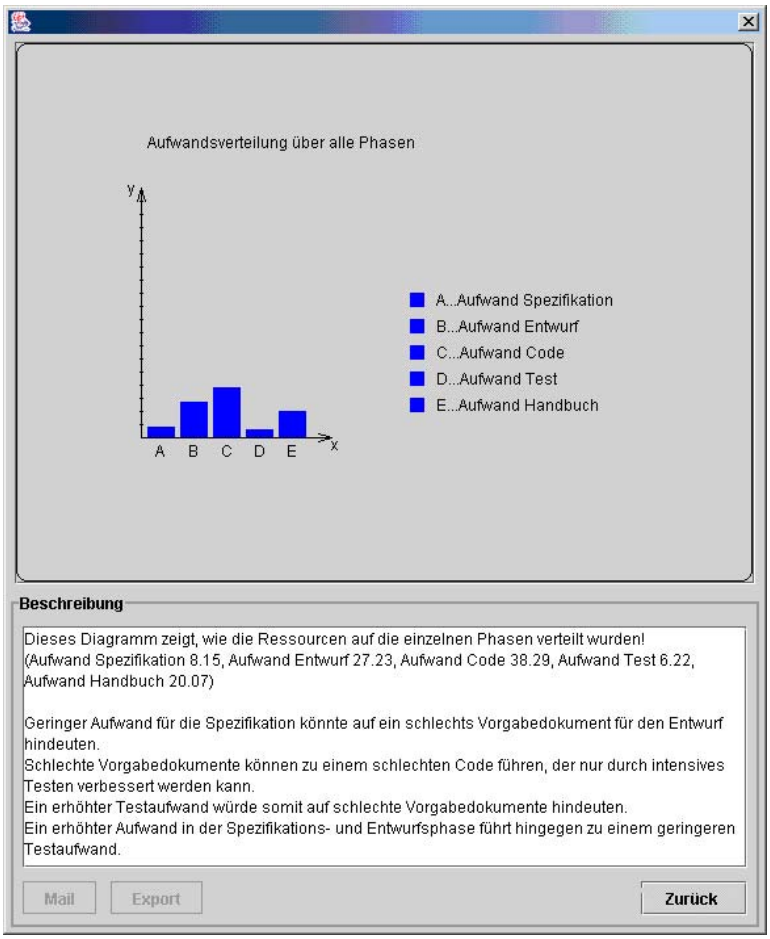



*Abb. 8.14:* Aufwandsverteilung

Die Abbildung 8.14 zeigt die Aufwandsverteilung eines Spielers. Leider gibt

es keine optimale Aufwandsverteilung, die den Spieler auf jeden Fall zum Abschluss eines erfolgreichen Projekts führt. Der Spieler kann aber durch dieses Diagramm erkennen, in welche Phasen besonders viel oder besonders wenig Aufwand geflossen ist. Der Erklärungstext weist den Spieler darauf hin, dass ein höherer Aufwand in den frühen Phasen (Spezifikation und Entwurf) zu korrekteren und vollständigeren Vorgabedokumenten führt. Dies hat positive Auswirkungen auf das Projekt, da sich dadurch weniger Fehler fortpflanzen können und somit die Qualität des Codes bereits durch einen geringen Testaufwand zufrieden stellend ist. Ein hoher Aufwand in der Testphase hingegen deutet auf schlampige Vorgabedokumente hin, die ein intensives Testen notwendig gemacht haben, um die Qualität des Codes zu verbessern.

*Autoren der Dokumente*

Diese Bewertungskriterien sollen prüfen, ob der Spieler für die einzelnen Phasen die richtigen Mitarbeiter eingesetzt hat.
Für diesen Bereich wurden bereits folgende Bewertungskriterien realisiert:

- Autor/Autoren der Spezifikation
- Autor/Autoren des Systemdesigns
- Autor/Autoren des Moduldesigns
- Autor/Autoren des Codes
- Autor/Autoren des Handbuchs

Die Abbildung 8.15 zeigt dem Spieler an, dass er den richtigen Mitarbeiter als Autor für den Systemdesign eingesetzt hat. Die Mitarbeiter müssen ihren Fähigkeiten entsprechend eingesetzt werden, da ihre Qualifikation Auswirkungen auf die Korrektheit und Vollständigkeit der Dokumente hat.


*Abb. 8.15:* Autor(en) des Systemdesigndokuments

### *8.4.1 Nicht realisierte Bewertungskriterien*

In diesem Abschnitt werden die Bewertungskriterien besprochen, die noch entwickelt werden bzw. diejenigen, die nicht realisiert werden.

Die folgende Liste enthält alle Bewertungskriterien, die dem Spieler in Zukunft zur Verfügung stehen sollen. Die speziellen Hilfsmittelkomponenten, die diese Bewertungskriterien beschreiben sowie die genaue Instanzensequenz mit den dazugehörigen Abfragen und Regeln, die für die Auswertung der Hilfsmittelkomponenten notwendig sind, wurden schon realisiert. Die Übertragung dieser Daten in die Datenbank sowie die Erstellung eines geeigneten Erklärungstextes muss jedoch noch vorgenommen werden.

- **Anzahl der Mitarbeiter pro Phase**
  Dieses Bewertungskriterium gibt an, wie viele Mitarbeiter gleichzeitig an einer Phase beschäftigt waren. Hat der Spieler in einer Phase mehrere Mitarbeiter eingesetzt, dann soll er darauf hingewiesen werden, dass der Kommunikationsaufwand mit steigender Anzahl von Mitarbeitern zunimmt und dadurch höhere Kosten entstehen.

- **unproduktive Zeit**
  Dabei wird sowohl die unproduktive Zeit der einzelnen Mitarbeiter angezeigt als auch die gesamte unproduktive Zeit während des Projekts. Eine zusätzliche Angabe der Kosten, die dadurch entstanden sind, ist denkbar und kann dem Spieler die Auswirkungen von Lücken im Personaleinsatz direkt vor Augen führen.

- **eingesetztes Reviewteam**
  Eine genaue Aufstellung der eingesetzten Reviewteams kann den Spieler auf eine falsche Gutachterwahl hinweisen.

- **erfolgreiche Durchführung der Korrekturphase**
  Dieses Bewertungskriterium ermittelt, ob die Korrekturphase abgebrochen oder erfolgreich beendet wurde. Damit kann die erfolglose Durchführung von Reviews begründet werden.

- **Korrektur durch den Autor**
  Es wird geprüft, ob der richtige Mitarbeiter mit der Korrektur des Dokuments beauftragt wurde. Der Spieler soll dadurch erkennen, dass qualifizierte Mitarbeiter schneller Fehler korrigieren können und weniger zusätzliche Fehler in das Dokument einfügen.

- **Anzahl erfolgreicher bzw. nicht erfolgreicher Reviews**
  Gibt die Anzahl der erfolgreich bzw. erfolglos durchgeführten Reviews zurück. Zusätzlich dazu kann der Spieler auf mögliche Gründe hingewiesen werden, warum Reviews nicht erfolgreich durchgeführt werden konnten.

- **Tester eines Dokuments**
  Dabei wird überprüft, ob der eingesetzte Mitarbeiter qualifiziert für diese Tätigkeit war.

- **Korrektur nach Test**
  Damit wird ermittelt, ob der Autor des Dokuments mit der Korrektur beauftragt wurde.

- **Parallelität von Phasen**
  Dieses Bewertungskriterium untersucht, ob der Spieler das Projekt strikt nach dem Wasserfallmodell oder phasenüberlappend durchgeführt hat. Die Spezialfälle „Modultest nach Codereview“ und „Integrationstest nach Modulspezifikationsreview“ werden ebenfalls noch realisiert. Der Spieler soll dabei auf die Probleme von überlappenden Phasen hingewiesen werden.

- **Beginn einer Phase erst nach Korrektur der Vorgängerphase**
  Dabei wird ermittelt, ob der Spieler auf das Ende der Korrektur gewartet hat, bevor er das Dokument in der nächsten Phase verwendet hat. Zusätzlich dazu kann dem Spieler vermittelt werden, welche Auswirkungen eine Verletzung dieser Regel mit sich bringt.

- **Teilnahme des Kunden**
  Hier wird geprüft, ob der Kunde am Spezifikations- bzw. Handbuchreviews teilgenommen hat. Der Spieler soll erkennen, dass es hilfreich ist den Kunden an Reviews zu beteiligen, da er Analysefehler aufdecken kann, die den Gutachtern nicht auffallen.

- **erfolgreicher Handbuchreview (mit Kunden) zu einem frühen Zeitpunkt**
  Dieses Bewertungskriterium untersucht, ob der Spieler bereits zu einem frühen Zeitpunkt im Projekt das Handbuch geprüft hat. Dabei kann dem Spieler vermittelt werden, dass durch die Prüfung des Handbuchs auch Analysefehler gefunden werden, die zu einem frühen Zeitpunkt noch mit geringem Aufwand korrigiert werden können.

Nicht realisiert werden folgende Bewertungskriterien:

- **Qualifikation der Mitarbeiter**
  Eine genaue Auflistung, welche Fähigkeiten die einzelnen Mitarbeiter besitzen wird nicht angegeben, da die Qualifikation bereits in mehrere Bewertungskriterien (Autor/Tester des Dokuments, Korrektur des Dokuments) integriert ist.

- **Anzahl der Gutachter in einem Review**
  Dieser Wert wird indirekt durch das eingesetzte Reviewteam repräsentiert. Darum genügt es, wenn im Erklärungstext des Bewertungskriteriums „eingesetztes Reviewteam“ auf die Unterschiede zwischen Reviews mit zwei bzw. drei Gutachtern hingewiesen wird.

- **Dauer von Reviewphasen**
  Dauert eine Reviewphase länger, dann hängt dies häufig damit zusammen, dass die Korrektur des vorherigen Reviews abgebrochen wurde und dadurch nicht mit dem neuen Review begonnen werden konnte. Da auf dieses Problem schon im Bewertungskriterium „erfolgreiche Durchführung der Korrekturphase“ hingewiesen wird, ist für diesen Aspekt kein eigenes Bewertungskriterium notwendig.

- **Anzahl der Inspektionen**
  Dass eine Vielzahl von Inspektionen auf einen chaotischen Projektplan hindeuten kann, konnte anhand der durchgeführten Testspiele nicht gezeigt werden. Darum wird dieses Bewertungskriterium nicht realisiert.

*Kapitel 9*

# GRENZEN UND ERWEITERUNGSMÖGLICHKEITEN

In diesem Kapitel werden Grenzen und Erweiterungsmöglichkeiten der generischen Erklärungskomponente aufgezeigt.

## *9.1 Grenzen*

In diesem Abschnitt werden zum einen Modellabhängigkeiten im Datenmodell diskutiert, und zum anderen Grenzen der Überprüfbarkeit von speziellen Hilfsmittelkomponenten aufgezeigt.

### *9.1.1 Datenbank*

Die Datenbank wurde modellunabhängig konstruiert, um auch Projektdaten zukünftiger Modelle speichern zu können. Bei den Entitäten gibt es keine Probleme, da sie je nach ZARMS-Pfad in die Datenbank überführt werden. Bei den Relationen kann die Modellunabhängigkeit nicht so einfach gezeigt werden. Es kann aber angenommen werden, dass es in zukünftigen Modellen auch Personen geben wird, die Dokumente erstellen. Lediglich die zusätzlichen Flags, die in Kapitel 6.3.3 beschrieben wurden, stellen eine Modellabhängigkeit dar. Einige der Flags werden möglicherweise in anderen Modellen aus anderen Gründen benötigt. Durch das automatische Einstellen der ZARMS-Filteroperation für ein beliebiges Modell, kann dieses Problem gelöst werden. Dieser Ansatz wird in Anhang E beschrieben. Durch das Einführen zusätzlicher Logik kann festgestellt werden, welche Daten für das Setzen welches Flags herangezogen werden sollen. Durch die Realisierung dieses Ansatzes können die Modellabhängigkeiten der zusätzlichen Flags in der Tabelle s_relation beseitigt werden.

### *9.1.2 Spezielle Hilfsmittelkomponenten*

Ein großes Problem, dass sich bei der Erstellung von speziellen Hilfsmittelkomponenten ergibt, ist die Überprüfbarkeit. Wenn der Modellbauer Hilfs-

mittelkomponenten zu einem Bewertungskriterium anlegt, kann er nicht feststellen, ob die Einträge für Instanzen, Abfragen und Regeln syntaktisch korrekt sind.
Lediglich die Einträge in den Abfragen (die SQL-Statements) können, vor dem Einfügen in die spezielle Hilfsmittelkomponente, auf der Datenbank getestet und somit auf syntaktische Korrektheit überprüft werden. Um zu Prüfen, ob eine spezielle Hilfsmittelkomponente korrekt ist, muss untersucht werden, ob die einzelnen Instanzen richtig sind. Jede Instanz muss einer bestimmten Regelart entsprechen, damit sie verarbeitet werden kann. Dabei ist es aber manchmal notwendig auch die Vorgänger- bzw. Nachfolgerinstanz zu betrachten, um die Regelart „Instanz ohne Regel“ identifizieren zu können. Es muss hier angemerkt werden, dass es natürlich in den Händen des Modellbauers liegt, ob die einzelnen Instanzen zweckmäßig sind und die Reihenfolge, in der sie abgearbeitet werden sollen, sinnvoll ist.
Das größte Problem, dass sich bei die Überprüfung von speziellen Hilfsmittelkomponenten ergibt, ist, dass es im Moment noch keine Möglichkeit gibt automatisch festzustellen, ob alle möglichen Fälle des Bewertungskriteriums durch eine Hilfsmittelkomponente abgedeckt werden. Dieses Problem ist das schwerwiegendste, da die Fehler in der speziellen Hilfsmittelkomponente erst zur Laufzeit aufgezeigt werden. Sollte der Modellbauer nämlich auf einen Fall vergessen haben, dann kann die generische Erklärungskomponente in einer bestimmten Situation zu diesem Bewertungskriterium keinen Erklärungstext bzw. kein Diagramm anzeigen.

## *9.2 Erweiterungsmöglichkeiten*

In diesem Abschnitt werden die Erweiterungsmöglichkeiten der Bewertungskriterien sowie der damit verbundenen Hilfsmittelkomponenten beschrieben. Auch die Präsentation der Ergebnisse der generischen Erklärungskomponente aus der Sicht des Spielers wird genauer unter die Lupe genommen und die dabei gefundenen Verbesserungsvorschläge werden diskutiert.

### *9.2.1 Bewertungskriterien*

Es ist denkbar, dass in Zukunft weitere Bewertungskriterien für das QS-Modell untersucht werden sollen, um zusätzliche Auswertungen ermöglichen zu können. Neue Bewertungskriterien können ohne Probleme in das bestehende System eingebettet werden. Es muss lediglich darauf geachtet werden, dass die Bezeichnung der speziellen Hilfsmittelkomponente noch nicht für die Auswertung eines anderen Bewertungskriteriums herangezogen wurde.

Das Auswerten von komplexeren Bewertungskriterien, für die zum Beispiel zusätzliche Berechnungen durchgeführt werden müssen, kann ebenfalls realisiert werden. Die dazu notwendigen Erweiterungen betreffen jedoch eher die speziellen Hilfsmittelkomponenten und werden daher im Abschnitt 9.2.2 genauer besprochen.

### *9.2.2 Hilfsmittelkomponenten*

Bei den speziellen Hilfsmittelkomponenten sind generell zwei Erweiterungen denkbar. Zum einen wäre es vorteilhaft, wenn Berechnungen von Werten ermöglicht werden könnten, und zum anderen wäre es sehr hilfreich, wenn der Erklärungstext nur auf die aktuellen Werte des Spielers zugeschnitten werden könnte. Diese beiden Erweiterungen werden im folgenden Abschnitt genauer untersucht.

#### *Durchführen von Berechnungen für die Auswertung*

Will man zum Beispiel wissen, wie effektiv ein Review geführt wurde, dann könnte es interessant sein, wenn man nicht nur die enthaltenen Fehler im Dokument den während des Reviews gefundenen Fehlern gegenüberstellt, sondern zusätzlich ermittelt, wie viel Prozent der enthaltenen Fehler von den Gutachtern gefunden wurden. Dieser Wert muss im Erklärungstext anstelle des „X“ eingefügt werden, bevor das Ergebnis dem Spieler angezeigt wird.
Um solche Berechnungen zu ermöglichen, müsste die Regel um ein Feld erweitert werden, dass die Berechnungsformel zum Beispiel in Java-Code enthält.

Abbildung 9.1 zeigt, wie eine derartige spezielle Hilfsmittelkomponente aussehen würde. Die Erklärungskomponente hat lediglich die Aufgabe die Platzhalter (Attribut1 und Attribut2) in der Formel durch die aktuellen Attributwerte, „FEHLER“ (gefundene Fehler) aus der ersten Instanz und „ANZ_FEHLER_GES“ (enthaltene Fehler) aus der zweiten Instanz, zu ersetzen und dieses Statement auszuführen. Das Resultat gibt an, wie viel Prozent von den im Dokument enthaltenen Fehlern von den Gutachtern gefunden wurden.

Sinnvoll wäre es, wenn diese Berechnung als zusätzliche spezielle Hilfsmittelkomponente zum Diagramm ausgewertet werden könnte. Wie eine derartige Auswertung möglich ist, wird im folgenden Abschnitt beschrieben, ist aber derzeit nicht implementiert.

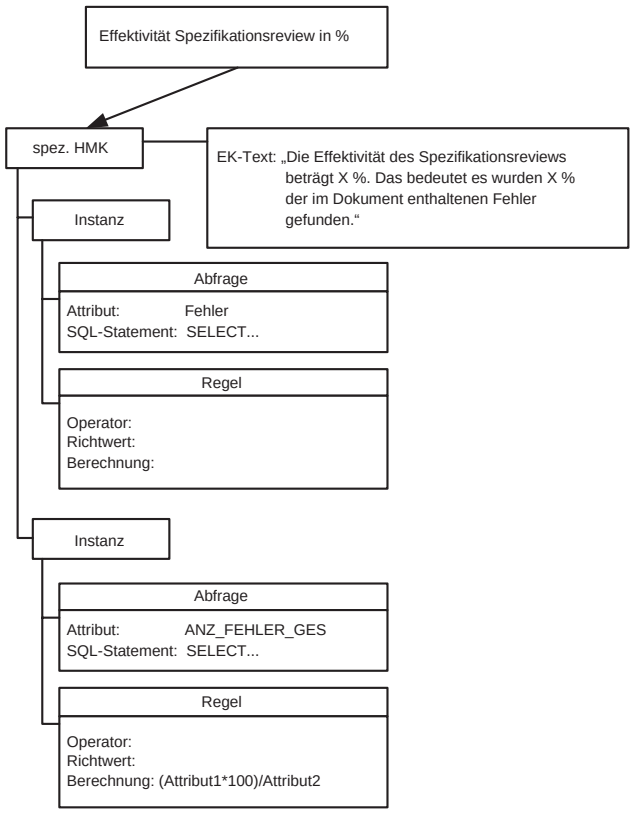



*Abb. 9.1:* Erweiterung der speziellen Hilfsmittelkomponente

*Erklärungstext nur zu den aktuellen Werten erstellen*

Der Erklärungstext, der zurzeit für Diagramme angezeigt wird, beschreibt, welche Besonderheiten in dem Diagramm auftreten können, und wie diese zu deuten sind. Für den Spieler wäre es allerdings besser, nur eine Erklärung zu seinen Werten zu bekommen, das heißt, zu seiner Situation. Dies ist machbar, indem man die Werte, die für das Diagramm zusammengestellt werden, gleichzeitig für die Auswertung anderer Bewertungskriterien heranzieht. Zurzeit wird für die Anzeige eines Diagramms ein Vektor generiert, der alle Daten enthält, die dargestellt werden sollten.
Diesen Vektor [Bezeichnung,Wert,Bezeichnung,Wert,..] müsste man durchlaufen, und für jede Bezeichnung nach einer eigenen speziellen Hilfsmittelkomponente suchen und diese auswerten. Der Erklärungstext würde dann aus dem Beschreibungstext des Diagramms und den Erklärungstexten der einzelnen speziellen Hilfsmittelkomponenten zusammengesetzt werden.

Nimmt man das Diagramm „Aufwandsverteilung“ genauer unter die Lupe, so kann der Aufbau der Instanzenkette, wie in Abbildung 9.2 dargestellt, erkannt werden.

Für die Auswertung des Bewertungskriteriums „Aufwandsverteilung“ werden alle dafür notwendigen Daten aus der Datenbank geholt und verarbeitet. Beim Durchlaufen der einzelnen speziellen Hilfsmittelkomponenten wird der folgenden Vektor erzeugt: [Aufwand Spezifikation, 5, Aufwand Entwurf, 20, Aufwand Codierung, 25, Aufwand Test, 40, Aufwand Handbuch, 10]
Dieser Vektor enthält nun alle Informationen damit wir das Diagramm erstellen können. Nun möchten wir überprüfen, ob der Spieler genug Zeit in die frühen Phasen investiert hat, oder ob er zu einem späteren Zeitpunkt durch intensives Testen Fehler entdeckt und korrigiert hat. Dazu können die Werte aus dem Vektor genutzt werden. Zusätzlich muss jedoch eine weitere spezielle Hilfsmittelkomponente erstellt werden, die überprüft, ob 5 % des Aufwands für die Spezifikation genug ist oder nicht. Diese Hilfsmittelkomponente wird in Abbildung 9.3 dargestellt.

Diese spezielle Hilfsmittelkomponente trägt den Namen „Aufwand Spezifikation“ und einen Erklärungstext, der, sofern diese Hilfsmittelkomponente zutrifft, zu der Beschreibung des Diagramms hinzugefügt wird. Es muss hier noch erwähnt werden, dass es in diesem Fall nicht unbedingt auch eine Hilfsmittelkomponente für den positiven Fall geben muss, da manchmal nur der negative Fall interessant ist. Gibt es zu einer bestimmten Situation keine zusätzliche Hilfsmittelkomponente, dann wird die Auswertung beim nächsten Eintrag im Vektor fortgeführt.

Die Abfrage muss nicht unbedingt angegeben werden, da der Wert bereits im Vektor enthalten ist. Allerdings kann die spezielle Hilfsmittelkomponente auch für andere Bewertungskriterien verwendet werden, wenn das SQL-Statement eingetragen ist.

Die Regel gibt an, dass mindestens 7 % des Projektaufwands in die Spezifikation fließen müssen. Das hilft uns noch nicht weiter, denn der Spieler könnte einen unfähigen Mitarbeiter für die Erstellung dieses Dokuments eingesetzt haben. Das bedeutet, dass zusätzlich noch die Vollständigkeit des Dokuments geprüft werden muss. Wir benötigen also eine weitere Instanz, die uns diesen Wert überprüft.

In unserem Fall hat der Spieler genügend Aufwand in die Erstellung des Spezifikationsdokuments investiert. Erreicht er mit der Vollständigkeit des

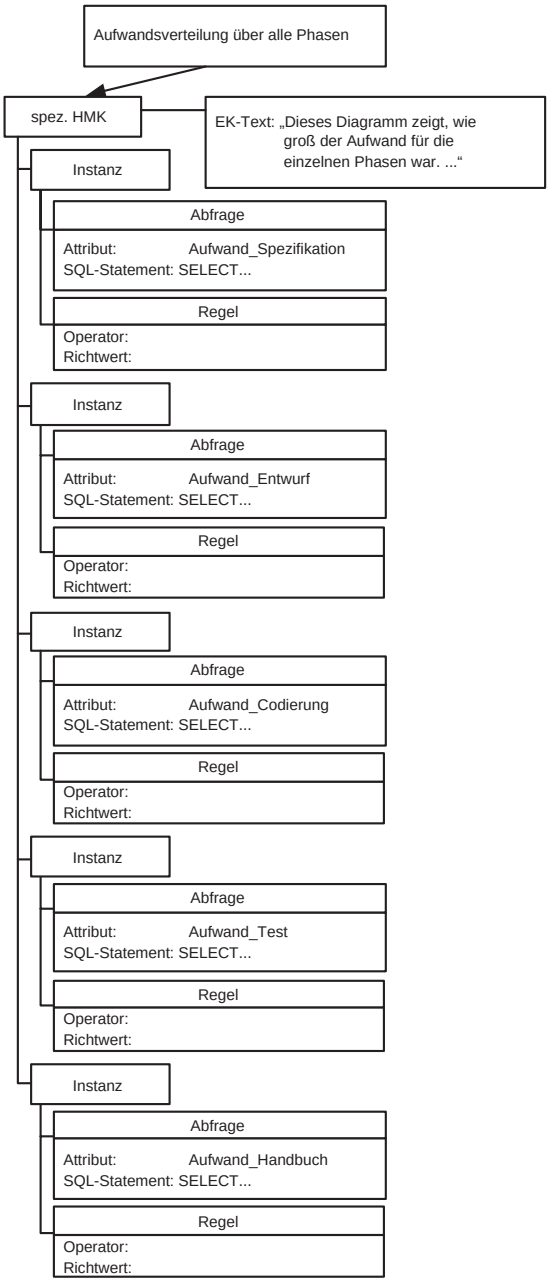



*Abb. 9.2:* spezielle Hilfsmittelkomponente - Aufwandsverteilung über alle Phasen

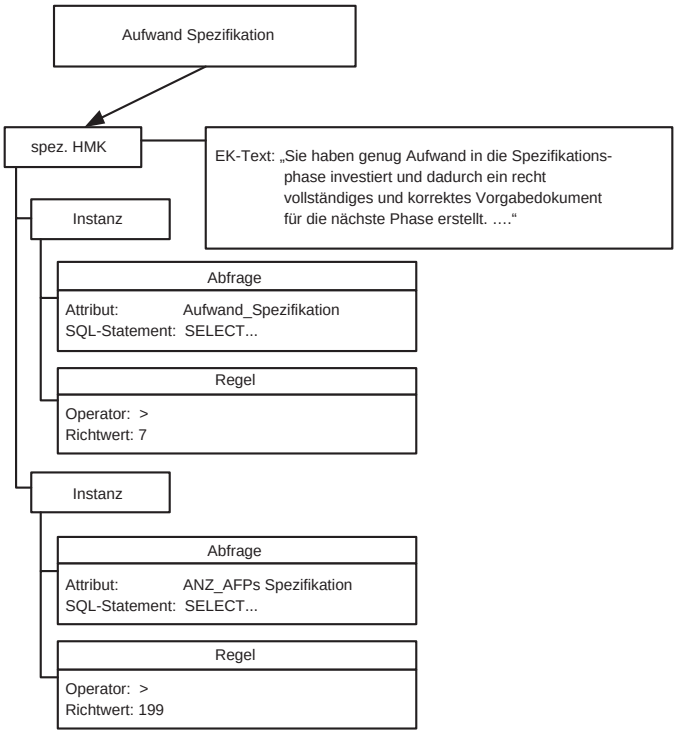



*Abb. 9.3:* spezielle Hilfsmittelkomponente - Aufwand Spezifikation

Dokuments auch den Richtwert der zweiten Instanz, dann kann der Erklärungstext dieser speziellen Hilfsmittelkomponente zu der Beschreibung des Diagramms hinzugefügt werden.

Die Erklärungskomponente müsste nun jeden weiteren Wert in dem Vektor prüfen. Nun kann es sein, dass es für einige Attribute keine Hilfsmittelkomponente gibt, da sie nicht relevant sind, diese Attribute werden dann einfach übersprungen. Nachdem der Vektor vollständig abgearbeitet wurde, wird das Diagramm und der zusammengesetzte Erklärungstext dem Spieler präsentiert.

Durch diese Methode kann der Erklärungstext jedes Spielers eindeutig auf dessen Werte abgestimmt werden. Das umständliche Extrahieren der relevanten Information aus der Menge an Erklärungstexten, das der Spieler bisher vornehmen musste, erübrigt sich, da er nur für ihn relevante Erklärungen erhält. Dadurch kann der Aufwand des Spielers verringert und mögliche Verwirrungen durch die Informationsflut, die aufgrund der überlagernden Effekte

auftreten können, vermieden werden.

Dieser Ansatz wurde nur deshalb noch nicht verwirklicht, da zukünftig die Erstellung der speziellen Hilfsmittelkomponenten automatisch vorgenommen werden soll, damit gewährleistet werden kann, dass die speziellen Hilfsmittelkomponenten korrekt sind und die Probleme, die in Abschnitt 9.1.2 besprochen wurden, beseitigt werden können. Um das zu ermöglichen, muss zuvor die Sprache genauer untersucht werden. Diese Überlegungen sind jedoch nicht mehr Teil dieser Arbeit.

### *9.2.3 Auswertung*

Ein großer Vorteil der generischen Erklärungskomponente liegt darin, dass der Spieler schon während des Spiels, oder sofort nach Beendigung des Spiels eine Auswertung sehen kann. Bis jetzt lag oft ein Zeitraum von bis zu zwei Wochen zwischen Spiel und Auswertung, was bedeutet, dass der Spieler schon wieder vergessen hat, welchen Mitarbeiter er für welche Tätigkeit eingesetzt hat. Der Spieler hat die Möglichkeit sich sofort einen Überblick über sein Spiel zu verschaffen. Er kann sich außerdem spezielle Teile seines Spieles genauer betrachten, die ihn interessieren und bekommt Ratschläge, wie er Probleme im nächsten Spiel vermeiden kann.

Im Moment wird der Auswertungsaufwand des Tutors durch die Erklärungskomponente stark reduziert. Der Tutor kann aber noch nicht vollständig durch die Erklärungskomponente ersetzt werden. Dazu bedarf es noch einiger Erweiterungen.
Mit Hilfe der Erklärungskomponente kann der Spieler einen Überblick über sein Spiel bekommen. Die Erklärungskomponente kann aber zurzeit nur Probleme aufzeigen und Annahmen über mögliche Ursachen treffen. Es wäre denkbar, dass am Ende eines Spieles eine automatische Auswertung angestoßen wird, die untersucht, warum die Zielvorgaben nicht eingehalten wurden. Wurden zum Beispiel die Kosten überschritten, dann sollten automatisch alle Bewertungskriterien, die starke Auswirkungen auf die Kosten haben, ausgewertet und eventuell nach Beeinflussungsgrad gereiht, dem Spieler präsentiert werden. Zum Beispiel würde überprüft werden, ob der Spielverlauf Lücken im Personaleinsatz aufzeigt, die höhere Kosten verursacht hätten, oder ob der Spieler viele Reviews und Tests durchgeführt hat, um vollständige und korrekte Dokumente zu erstellen. Der Spieler würde somit nicht nur erfahren, dass er die Kosten überschritten hat, sondern auch mögliche Gründe dafür anhand von Diagrammen und Erklärungstexten präsentiert bekommen.

Im Moment muss sich der Spieler diese Informationen aus den einzelnen Bewertungskriterien selbst extrahieren. Bei der Auswertung jedes Bewertungskriteriums werden im Erklärungstext eine Vielzahl von möglichen Ursachen und deren Auswirkungen auf das Projekt angegeben. Da sich viele Effekte überlagern (zu sehen in Anhang A), ist eine erschöpfende Auflistung der Ursachen im Erklärungstext nicht möglich. Beim genaueren Betrachten der einzelnen Diagramme, wird der Spieler auf Probleme hingewiesen, die er genauer untersuchen kann. Unerfahrene Spieler können dabei jedoch schnell den Überblick verlieren. Deshalb ist dieser Prototyp für Spieler, die das erste Spiel spielen nicht geeignet. Durch eine automatische Anzeige von möglichen Gründen für bestimmte Effekte können auch Anfänger durch das Labyrinth von interessanten Bewertungskriterien hindurch geführt werden, und dadurch auf alle wichtigen Aspekte aufmerksam gemacht werden.

*Kapitel 10*

# ZUSAMMENFASSUNG

Im Rahmen dieser Arbeit wurde die Erweiterung des AMEISE-Systems, um die generische Erklärungskomponente beschrieben. Die Erklärungskomponente hat die Aufgabe Spielverläufe zu bewerten und dadurch den Tutor zu entlasten.
Um eine Auswertung vornehmen zu können, mussten Bewertungskriterien gesucht werden, anhand derer Spielverläufe analysiert werden können. Als Basis für diese Suche wurden Testspiele aus der Lehrveranstaltung „Systementwicklungsprozess“ aus dem Sommersemester 2002 herangezogen und genauer untersucht.
Bei der Entwicklung der Erklärungskomponente musste darauf geachtet werden, dass sie sowohl für bestehende, als auch für zukünftige Modelle einsetzbar ist. Durch die genauere Betrachtung der identifizierten Bewertungskriterien für das QS-Modell fand man heraus, dass alle Bewertungskriterien durch ähnliche Fragestellungen ausgewertet werden können. Darum bot sich für die Entwicklung der generischen Erklärungskomponente das Design Pattern „Interpreter“ als Lösungsansatz an, da mit dessen Hilfe die Generizität der Erklärungskomponente gewährleistet werden konnte. Es wurde eine geeignete Repräsentation für die verwendete Grammatik gesucht und schließlich in Form der speziellen Hilfsmittelkomponente realisiert.
Da man der Erklärungskomponente einfachen Zugriff auf die Daten, die sie für die Auswertung der Spielverläufe benötigt, gewähren wollte, wurden sowohl die Spieldaten als auch die speziellen Hilfsmittelkomponenten in einer Datenbank gespeichert.
Jede spezielle Hilfsmittelkomponente besteht aus einer Abfrage, einer Regel und einem Erklärungstext. Die Abfrage wird benötigt, um aktuelle Attributwerte aus der Datenbank laden zu können und anschließend in die Regel einzubauen. Die Regel gibt an, mit welchem Operator und mit welchem Richtwert der Attributwert verglichen werden muss. Die generische Erklärungskomponente fungiert lediglich als Regelinterpreter, der diese Repräsentation nutzt, um die Regel zu interpretieren und zu überprüfen. Trifft die Regel zum aktuellen Zeitpunkt zu, dann kann die Erklärungskomponente als

Ergebnis den Erklärungstext der speziellen Hilfsmittelkomponente an den Spieler weiterleiten. Um auch komplexere Bewertungskriterien analysieren zu können, ist es möglich mehrere Regeln zu einer Hilfsmittelkomponente zu verketten.
Die Auswertung der Spielverläufe wird dem Spieler in Form von Diagrammen und Erklärungstexten präsentiert. Dabei werden Fehlentscheidungen aufgezeigt und deren Auswirkungen auf das Projekt erläutert. Außerdem werden dem Spieler Tipps gegeben, wie diese Fehler vermieden werden können, und worauf im nächsten Spiel geachtet werden sollte.
Die generische Erklärungskomponente hat zum einen dazu geführt, den Betreuungsaufwand des Tutors zu minimieren, und zum anderen konnte dadurch auch ein großer Schritt in Richtung Selbststudium gemacht werden.

*Anhang A*

# KOMPLEXITÄT DER EFFEKTE DES QS-MODELLS

Der folgende Graph zeigt, wie komplex das System aus Bewertungskriterien im QS-Modell aufgebaut ist. Als Bezeichnungen wurden die Kürzel aus der Tabelle 5 in Kapitel 5 übernommen. Es muss an dieser Stelle angemerkt werden, dass nur die direkten Abhängigkeiten in diesem Graphen dargestellt werden. Natürlich gibt es auch viele indirekte Abhängigkeiten zwischen Bewertungskriterien. Da der Graph aber bereits mit den direkten Abhängigkeiten überladen ist, wurden diese nicht mehr eingezeichnet.

Abbildung A.1 zeigt die direkten Abhängigkeiten, die zwischen verschiedenen Bewertungskriterien bestehen. Jedes Bewertungskriterium wird als Knoten repräsentiert. Existiert zwischen zwei Bewertungskriterien eine Abhängigkeit, dann wird diese durch eine Kante dargestellt. Wie die Abbildung zeigt, kann ein Bewertungskriterium auch von vielen anderen abhängig sein. In diesem Fall werden die Knoten zu einer Gruppe zusammengefasst und nur durch eine Kante mit dem abhängigen Bewertungskriterium verbunden. Zwischen zwei Bewertungskriterien, die in derselben Gruppe liegen, aber nicht durch eine Kante miteinander verbunden sind, gibt es keine Abhängigkeit.

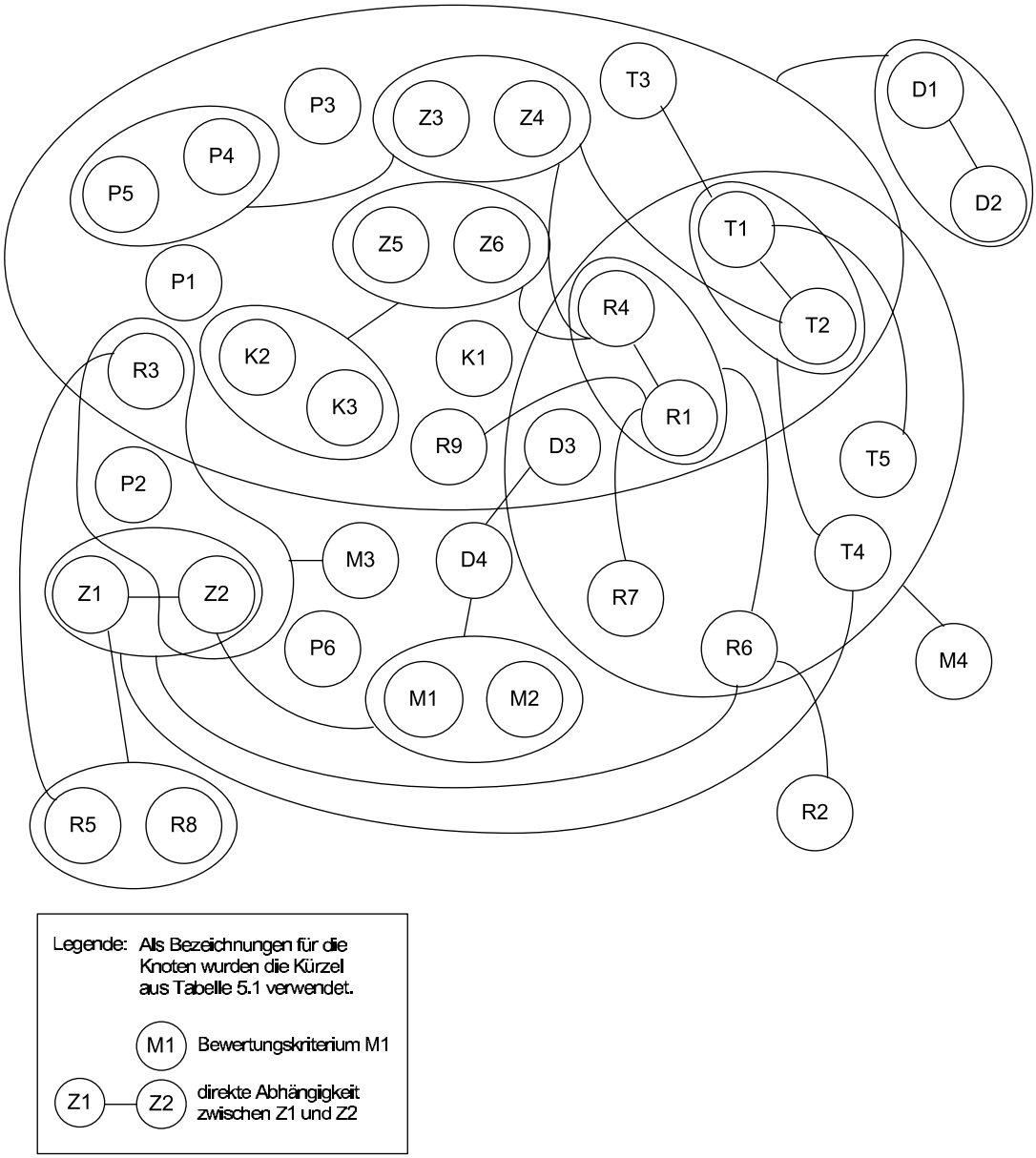



*Abb. A.1:* Abhängigkeiten zwischen Bewertungskriterien des QS-Modells

*Anhang B*

# DAS DATENMODELL

In Abbildung B.1 wird das vollständige Datenmodell[1] des AMEISE-Systems gezeigt. Die unterschiedlichen Farben sollen verschiedene Bereiche des Datenmodells von einander abgrenzen:

- **blau:** Benutzerdaten
  Der blaue Ausschnitt enthält alle Informationen über die Benutzer des AMEISE-Systems. Es gibt unterschiedliche Benutzergruppen, wie Administrator, Lehrveranstaltungsleiter, Spieler und Modellbauer, von denen jede eine andere Sicht auf die Datenbank hat. Den Lehrveranstaltungsleiter hat Zugriff auf die Kurse und die Spiele der Spieler. Der Spieler hingegen bekommt alle Informationen über Kurse und Turniere, deren Spiele er spielen darf. Der Modellbauer interessiert sich nur für den Modellbereich und die Hilfsmittelkomponenten, die für die Analyse der Bewertungskriterien des Modells benötigt werden. Der Administrator hat als einziger uneingeschränkten Zugriff auf alle Daten.

- **grün:** Kursdaten
  Der grüne Ausschnitt enthält alle wichtigen Informationen über die Kurse. Wichtig sind hier vor allem die Tabellen „tournament“ und „round“. Da in einem Kurs mehrere AMEISE-Spiele (Turniere) durchgeführt werden können, müssen diese eindeutig voneinander unterscheidbar sein. Im ersten Turnier wird zum Beispiel das QS-Modell gespielt, während im zweiten Turnier ein Projekt aufgrund des QSVA-Modell durchgeführt werden soll. Außerdem kann es vorkommen, dass mehrere Spiele desselben Modells gespielt werden müssen. Diese werden durch die Runde gekennzeichnet. Nehmen wir an, dass in einem Turnier das QS-Modell gespielt wird. Dann könnte es möglich sein, dass die Spieler ein erstes Spiel durchführen, um sich mit dem Simulator vertraut

[1] Eine genaue Beschreibung der einzelnen Tabellen des Datenmodells kann dem internen Dokument [Hipfl] „Projekt AMEISE - Beschreibung des konzeptuellen Datenbankschemas“ entnommen werden.

zu machen und das zweite Spiel erst dasjenige ist, welches vom Tutor bewertet werden soll. Das bedeutet, es gibt in diesem Turnier zwei Runden (Spiel1 und Spiel2).

- **rot:** Modelldaten
  Der rote Ausschnitt beschreibt das Modell zum Beispiel das klassische QS-Modell, für das es mehrere Instanzen geben kann, zum Beispiel das QSVA-Modell oder das QS-Modell mit Motivationseffekten. Zu den wichtigen Informationen, die über Modellinstanzen gespeichert werden, zählen sowohl die Aufgabenstellung als auch das Beginn- und das Endedatum, dass die Zeitspanne angibt in der das simulierte Projekt durchgeführt werden muss.

- **gelb:** Spieldaten
  Der gelbe Ausschnitt enthält alle wichtigen Informationen über ein Spiel. Es werden sowohl alle Spielzüge des Spielers gespeichert, als auch alle wichtigen ZARMS-Daten, die den aktuellen Zustand des Projekts widerspiegeln.

- **orange:** Daten für Hilfsmittelkomponenten
  Der orangene Ausschnitt entält alle speziellen Hilfsmittelkomponenten sowie alle Daten, die für die Auswertung der Hilfsmittelkomponenten benötigt werden.

- **hellblau und rosa:** Systemdaten
  Diese Teile enthalten zusätzliche Informationen, die für das AMEISE-System benötigt werden.

Für die generische Erklärungskomponente sind allerdings nur der orange und der gelbe Ausschnitt interessant. Der orange Teil enthält die Hilfsmittelkomponenten, die für die Auswertung der Bewertungskriterien herangezogen werden. Zusätzlich dazu werden auch die aktuellen ZARMS-Attributwerte aus dem gelben Teil für die Verarbeitung der Hilfsmittelkomponenten benötigt. Diese Ausschnitte aus der Datenbank wurden in Kapitel 6 und in Kapitel 7 genauer beschrieben.

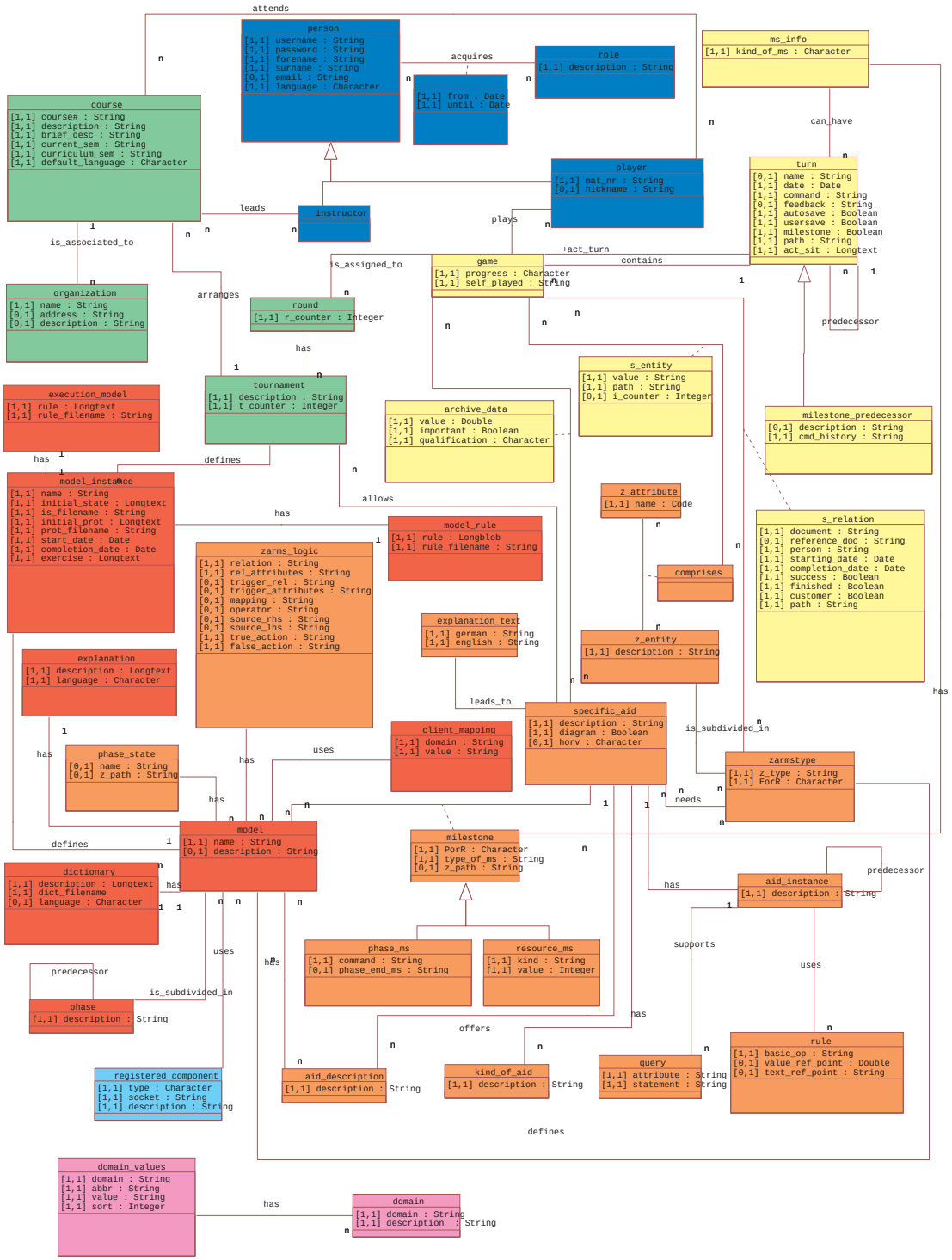



*Abb. B.1:* Datenbankmodell des AMEISE-Systems

*Anhang C*

# AUFBAU/VERARBEITUNG VON SPEZIELLEN HILFSMITTELKOMPONENTEN

Zum aktuellen Zeitpunkt werden die speziellen Hilfsmittelkomponenten einer der folgenden Komponenten zugeordnet:

1. Auswertung (Erklärungskomponente)
2. Ratgeber
3. väterlicher Freund
4. Milestone

Im Weiteren wird nur auf die speziellen Hilfsmittelkomponenten eingegangen, die für die Auswertung benötigt werden. Die Hilfsmittelkomponenten für Ratgeber und väterlichen Freund sind gleich aufgebaut und werden zusammen mit den Milestones in der Diplomarbeit von Markus Nusser [NUSSER] genauer besprochen. Der Aufbau der Hilfsmittelkomponenten ist in allen vier Fällen der gleiche. Jede Hilfsmittelkomponente ist einer Hilfsmittelkomponenten-Art und einer generischen Beschreibung zugeordnet. Außerdem besteht jede Hilfsmittelkomponente aus einer Sequenz von einer oder mehreren Instanzen, die jeweils eine zugeordnete Abfrage und eine (oder keine) Regel besitzen. Im Folgenden werden die einzelnen Tabellen sowie deren Attribute näher beschrieben.

Die Art der speziellen Hilfsmittelkomponente wird in der **Tabelle „kind_of_aid“** festgelegt. Diese Tabelle hat folgende Attribute:

- **kaidid:** kind_of_aid-ID
- **description:** Beschreibung der Hilfsmittelkomponenten-Art (Diese Beschreibung muss eindeutig sein!!)

Beispiel:
INSERT INTO kind_of_aid (kaidid,description) VALUES (1, „Auswertung“);
INSERT INTO kind_of_aid (kaidid,description) VALUES (2, „Ratgeber“);
INSERT INTO kind_of_aid (kaidid,description) VALUES (3, „Väterlicher Freund“);
INSERT INTO kind_of_aid (kaidid,description) VALUES (4, „Milestone“);

Außerdem gibt es noch eine generische Beschreibung zu jeder speziellen Hilfsmittelkomponente (dies gilt allerdings nur für „Auswertung“, „Ratgeber“ und „Väterlicher Freund“). Die Festlegung einer generischen Beschreibung erlaubt einen leichteren Zugriff auf zusammengehörige spezielle Hilfsmittelkomponenten. Mit Hilfe der generischen Beschreibung „Zielvorgaben“ zum Beispiel können alle speziellen Hilfsmittelkomponenten gefunden werden, die die einzelnen Zielvorgaben für das Modell auswerten. Die generischen Beschreibungen müssen in der **Tabelle „aid_description“** festgelegt werden. Diese Tabelle besteht aus den folgenden Attributen:

- **aiddid:** aid_description-ID
- **description:** generische Beschreibung (diese Beschreibung muss eindeutig sein!!)

Beispiel:
INSERT INTO aid_description (aiddid,description) VALUES (1, „Zielvorgaben“);

Beim Einfügen einer speziellen Hilfsmittelkomponente kann sowohl eine Hilfsmittelkomponenten-Art („kind_of_aid“) als auch eine generische Beschreibung („aid_description“) festgelegt werden.

Jede spezielle Hilfsmittelkomponente enthält die wichtigen Grundinformationen:

- Was soll mit der speziellen Hilfsmittelkomponente geprüft werden?
- Welcher Erklärungstext soll zurückgegeben werden?
- Handelt es sich bei dieser Hilfsmittelkomponente um ein Diagramm?

Die **Tabelle specific_aid** besteht aus folgenden Attributen:

- **spaidid:** specific_aid-ID

- **description:** Beschreibung (für die Anzeige im AuswertungsGui notwendig)
- **german:** deutscher Erklärungstext
- **english:** englischer Erklärungstext
- **diagram:** 1 wenn es sich um ein Diagramm handelt, sonst 0
- **horv:** 'V' wenn das Diagramm vertikal gezeichnet werden soll, 'H' für horizontal
- **mid:** Modell-ID
- **aiddid:** aid_description-ID
- **kaidid:** kind_of_aid-ID

Beispiel: INSERT INTO specific_aid (spaidid, description, german, english, diagram, horv, mid, aiddid, kaidid) VALUES (1,"Dauer", „Sie haben die vorgegebene Zeit noch nicht überschritten.", „",0,null,1, 1, 1);

ACHTUNG: Jeder Eintrag in specific_aid.description muss eindeutig sein, da die Komponente „AuswertungsGui" am Client alle „descriptions" aus der Datenbank lädt, und dem Spieler anzeigt. Kommt dann ein Begriff zweimal vor, wird er nur einmal angezeigt. Es wurde so realisiert, damit es auch zwei spez. HMKs geben kann, die auf denselben Wert abfragen, d.h. es gibt zwei Hilfsmittelkomponenten mit „description = Dauer", bei der einen wird geprüft, ob der Spieler die Dauer schon überschritten hat, bei der zweiten, ob der Spieler die Dauer noch nicht überschritten hat. Wählt der Spieler in der Liste den Eintrag „Dauer" aus, dann werden beide speziellen Hilfsmittelkomponenten überprüft, um immer eine Aussage treffen zu können.

Jede spezielle Hilfsmittelkomponente besteht aus einer oder mehrerer Instanzen. Die Instanzen werden durch die **Tabelle aid_instance** repräsentiert und beinhalten die folgenden Attribute:

- **instid:** aid_instance-ID
- **description:** Beschreibung (wird im Moment nicht verwendet)
- **predecessor:** Vorgänger dieser Instanz
- **spaidid:** specific_aid-ID

- **qid:** query-ID
- **rlid:** rule-ID

Beispiel:
INSERT INTO aid_instance (instid, description, predecessor, spaidid, qid, rlid) VALUES (1, „“, null, 1, 1, 1);

Jeder Instanz muss eine Abfrage zugeordnet werden. Die Abfrage enthält alle Informationen, damit der für die Auswertung der Hilfsmittelkomponente notwendige Wert aus der Datenbank geladen werden kann. Die **Tabelle „query“** besteht aus folgenden Attributen:

- **qid:** Query-ID
- **attribute:** Name des Attributs, das mit dem „statement“ aus der Datenbank geladen wird
- **statement:** SQL-Statement

Beispiel:
INSERT INTO query (qid, attribute, statement) VALUES (1, „Beginndatum“,
'select model_instance.start_date
from model, model_instance
where model_instance.mid = model.mid AND ');

ACHTUNG: Alle Queries enden deshalb mit einem AND, weil sie zur Laufzeit noch um die aktuelle Game-Id, und um den aktuellen Pfad erweitert werden müssen, damit der aktuelle Wert eines bestimmten Spielers aus der Datenbank geladen werden kann.

Jeder Instanz kann weiters eine Regel zugeteilt werden, die beschreibt, was mit dem Wert aus der Tabelle „query“ geschehen soll. Die **Tabelle „rule“** besteht aus folgenden Attributen:

- **rlid:** rule-ID
- **zort:** 'Z' wenn der Richtwert eine Zahl ist,
  'T' wenn der Richtwert ein Text (eine Liste) ist

- **basic_op:** folgende Operatoren wurden realisiert
  beim Vergleich von Werten („<“, „<=“, „>“, „>=“, „==“, „!=“)
  beim Vergleich von Texten („IN“, „NOT IN“, „ALL IN“, „==“, „!=“)
  Sonderfall: Dauer („AFTER“, „BEFORE“)

- **value_ref_point:** Zahl (Richtwert)

- **text_ref_point:** Text (Richtwert)

Beispiel:
INSERT INTO rule (rlid, zort, basic_op, value_ref_point, text_ref_point)
VALUES (1, 'Z', 'AFTER',270, '');

### Zusammenstellung einer neuen speziellen Hilfsmittelkomponente (am Beispiel eines Diagramms)

Am folgenden Beispiel soll das Anlegen einer neuen speziellen Hilfsmittelkomponente gezeigt werden:

Die neue spezielle Hilfsmittelkomponente trägt den Namen „Restfehler im Systemdesign“ und ist eines der Diagramme, die der generischen Beschreibung „Restfehler in Dokumenten“ zugeordnet wird.
INSERT INTO aid_description (aiddid,description) VALUES (3,“Restfehler in Dokumenten“);

Die spezielle Hilfsmittelkomponente wird durch folgendes INSERT in die Datenbank eingefügt:
INSERT INTO specific_aid (spaidid,description,german,english,diagram,
horv,mid,aiddid,kaidid) VALUES (16,'Restfehler im Systemdesign',
'Das Diagramm zeigt die Restfehler im Systemdesign....',
'The diagram shows the errors in the system design....',
1,'V',1,3,1);

Nun müssen noch die dazugehörigen Instanzen sowie die Abfragen angelegt werden. Auf Regeln kann in diesem Fall verzichtet werden, da bei Diagrammen nur Daten aus der Datenbank geholt und angezeigt werden müssen.

INSERT INTO aid_instance (instid,description,predecessor,spaidid,qid,rlid)
VALUES (25,““,null,16,19,null);

```
INSERT INTO query (qid,attribute,statement)
VALUES (19, „Restfehler Systemdesign gesamt“,
'select s_entity.value
from zarmstype, z_entity, z_attribute, comprises, s_entity, game,
spaid_needs_zt, specific_aid
where spaid_needs_zt.spaidid = specific_aid.spaidid AND
spaid_needs_zt.zid = zarmstype.zid AND
zarmstype.eor = „E“ AND
zarmstype.z_type = „SYSTEMENTWURF“ AND
zarmstype.zid = z_entity.zid AND
z_entity.description = „Systemdesign“ AND
z_entity.zeid = comprises.zeid AND
comprises.zaid = z_attribute.zaid AND
z_attribute.name = „ANZ_FEHLER_GES“ AND
comprises.compid = s_entity.compid AND
s_entity.gid = game.gid AND ');

INSERT INTO aid_instance (instid,description,predecessor,spaidid,qid,rlid)
VALUES (26,““,25,16,20,null);

INSERT INTO query (qid,attribute,statement)
VALUES (20, Restfehler Systemdesign AF",
'select s_entity.value
from zarmstype, z_entity, z_attribute, comprises, s_entity, game,
spaid_needs_zt, specific_aid
where spaid_needs_zt.spaidid = specific_aid.spaidid AND
spaid_needs_zt.zid = zarmstype.zid AND
zarmstype.eor = „E“ AND
zarmstype.z_type = „SYSTEMENTWURF“ AND
zarmstype.zid = z_entity.zid AND
z_entity.description = „Systemdesign“ AND
z_entity.zeid = comprises.zeid AND
comprises.zaid = z_attribute.zaid AND
z_attribute.name = „ANZ_AF“ AND
comprises.compid = s_entity.compid AND
s_entity.gid = game.gid AND ');

INSERT INTO aid_instance (instid,description,predecessor,spaidid,qid,rlid)
VALUES (27,““,26,16,21,null);
```

```
INSERT INTO query (qid,attribute,statement)
VALUES (21, „Restfehler Systemdesign GF“,
'select s_entity.value
from zarmstype, z_entity, z_attribute, comprises, s_entity, game,
spaid_needs_zt, specific_aid
where spaid_needs_zt.spaidid = specific_aid.spaidid AND
spaid_needs_zt.zid = zarmstype.zid AND
zarmstype.eor = „E“ AND
zarmstype.z_type = „SYSTEMENTWURF“ AND
zarmstype.zid = z_entity.zid AND
z_entity.description = „Systemdesign“ AND
z_entity.zeid = comprises.zeid AND
comprises.zaid = z_attribute.zaid AND
z_attribute.name = „ANZ_GF“ AND
comprises.compid = s_entity.compid AND
s_entity.gid = game.gid AND ');
```

### Suchen nach passenden speziellen Hilfsmittelkomponenten während des Spiels

Wählt der Spieler in der Auswertungsliste einen Eintrag aus, dann werden alle dazugehörigen Hilfsmittelkomponenten aus der Datenbank geladen, und ausgewertet. Es kann zu jedem Zeitpunkt nur eine spezielle Hilfsmittelkomponente geben, die gültig ist, um zu ermöglichen, dass der Spieler zu jedem Zeitpunkt genau eine Auswertung erhalten kann. Jede dieser Hilfsmittelkomponenten wird gleich verarbeitet:

Zuerst werden alle notwendigen Daten (alle dazu existierenden specific_aid-, aid_instance-, rule- und query-Einträge) aus der Datenbank geladen. Dann wird geprüft, welche Daten in der Tabelle „rule“ gefüllt sind. Es gibt mehrere Möglichkeiten:

1. **die Regel besteht aus Operator(basic_op) und Richtwert (zort)**
   In diesem Fall wird der Wert aus dem SQL-Statement in „query“ ermittelt und mit dem jeweiligen Richtwert (Zahl oder Text) verglichen. Wenn dieses Statement wahr ist, dann werden der Wert und die dazugehörige Beschreibung des Attributes (query.attribute) in einem Vektor gespeichert. Dieser Vektor wird an den Client übermittelt.

2. **die Regel besteht nur aus einem Operator (basic_op)**
   In diesem Fall wird der Wert aus dem SQL-Statement in „query“ ermittelt und zusammen mit dem aktuellen Richtwert in einem Vektor

gespeichert. Dieser Wert wird dann mit dem nächsten Wert aus der „query“ der nächsten Instanz verglichen. Dann wird wie bei 1. vorgegangen. (Achtung: Es wird angenommen, dass im Fall 2 immer nur Werte miteinander verglichen werden. Sollte in Zukunft ein Vergleich von Texten notwendig werden, dann muss dieser Fall noch erweitert werden.

3. **es existiert keine Regel für diese Instanz** (es gibt keinen „rule“-Eintrag)
   In diesem Fall handelt es sich um eine Instanz für ein Diagramm. Dabei wird nur der Wert aus dem SQL-Statement in „query“ ermittelt und in einem Vektor zusammen mit dem Attributnamen (query.attribute) gespeichert. Dieser Vektor wird an den Client geschickt und dort als Diagramm dargestellt.

Am Client wird dann der Erklärungstext der speziellen Hilfsmittelkomponente (specific_aid.descripiton) durchlaufen und nach einem „*“-Zeichen gesucht. Wird ein „*“-Zeichen im Text gefunden, dann werden an dieser Stelle die Attributbeschreibungen und die dazugehörigen Werte aus dem Vektor eingefügt. Sollte das „*“-Zeichen nicht vorkommen, dann werden die Werte im Vektor nicht beachtet und auch nicht für den Spieler angezeigt. Der Spieler erhält dann nur den normalen Erklärungstext, der je nach ausgewählter Sprache in Deutsch oder Englisch angezeigt werden kann.

*Anhang D*

# RELEVANTE ZARMS-DATEN FÜR DIE AUSWERTUNG

Die folgenden Attributwerte müssen bei jedem „Proceed“ aus dem ZARMS-Datenstrom extrahiert und in die Datenbank geschrieben werden.

**Attribute bezüglich Abnahmetestbericht:**
zarmstype.z_type = ABNAHMETESTBERICHT
z_entity = Acceptancetestreport
z_attribute.name = FEHLER
z_attribute.name = VERLUSTE
z_attribute.name = NUMMER
z_attribute.name = ANZ_FEHLER_PRO_PRF
z_attribute.name = ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Code:**
zarmstype.z_type = CODE
z_entity = Code
z_attribute.name= ANZ_AFP
z_attribute.name= ANZ_FEHLER_GES
z_attribute.name= ANZ_AF
z_attribute.name= ANZ_GF
z_attribute.name= ANZ_FF
z_attribute.name= ANZ_IF
z_attribute.name= FEHLER_PRO_KLOC

**Attribute bezüglich Codereviewbericht:**
zarmstype.z_type = CODEREVIEWBERICHT
z_entity = Codereviewreport
z_attribute.name= FEHLER
z_attribute.name= VERLUSTE
z_attribute.name= NUMMER
z_attribute.name= ANZ_FEHLER_PRO_PRF
z_attribute.name= ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Entwickler:**
zarmstype.z_type = Entwickler
z_entity = Axel, Bernd, Christine, Diana, Richard, Stefanie, Thomas
Für jede dieser Entitäten sind die folgenden Attribute wichtig:
z_attribute.name= IST_EINGESTELLT
z_attribute.name= KOSTEN

**Attribute bezüglich Handbuch:**
zarmstype.z_type = HANDBUCH
z_entity = Manuals
z_attribute.name= ANZ_AFP
z_attribute.name= ANZ_FEHLER_GES
z_attribute.name= PROZENT_AFP_100
z_attribute.name= ANZ_AF
z_attribute.name= ANZ_HF
z_attribute.name= FEHLER_PRO_SEITE

**Attribute bezüglich Handbuchreviewbericht:**
zarmstype.z_type = HANDBUCHREVIEWBERICHT
z_entity = Manualsreviewreport
z_attribute.name= FEHLER
z_attribute.name= VERLUSTE
z_attribute.name= NUMMER
z_attribute.name= ANZ_FEHLER_PRO_PRF
z_attribute.name= ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Integrationstestbericht:**
zarmstype.z_type = INTEGRATIONSTESTBERICHT
z_entity = Integrationstestreport
z_attribute.name= FEHLER
z_attribute.name= VERLUSTE
z_attribute.name= NUMMER
z_attribute.name= ANZ_FEHLER_PRO_PRF
z_attribute.name= ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Modulspezifikation:**
zarmstype.z_type = MODULSPEZIFIKATION
z_entity = Moduledesign
z_attribute.name= ANZ_AFP
z_attribute.name= ANZ_FEHLER_GES

z_attribute.name= PROZENT_AFP_100
z_attribute.name= ANZ_AF
z_attribute.name= ANZ_GF
z_attribute.name= ANZ_FF

**Attribute bezüglich Modulespezifikationsreviewbericht:**
zarmstype.z_type = MODULSPEZREVIEWBERICHT
z_entity = Moduledesignreviewreport
z_attribute.name= FEHLER
z_attribute.name= VERLUSTE
z_attribute.name= NUMMER
z_attribute.name= ANZ_FEHLER_PRO_PRF
z_attribute.name= ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Moduletestbericht:**
zarmstype.z_type = MODULTESTBERICHT
z_entity = Moduletestreport
z_attribute.name= FEHLER
z_attribute.name= VERLUSTE
z_attribute.name= NUMMER
z_attribute.name= ANZ_FEHLER_PRO_PRF
z_attribute.name= ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Projektlogbuch:**
zarmstype.z_type = PROJEKTLOGBUCH
z_entity = Projectlog
z_attribute.name= SPEZIFIKATION_AUFWAND
z_attribute.name= SPEZIFIKATION_BEGINN
z_attribute.name= SPEZIFIKATION_ENDE
z_attribute.name= ENTWURF_AUFWAND
z_attribute.name= ENTWURF_BEGINN
z_attribute.name= ENTWURF_ENDE
z_attribute.name= MODSPEZ_AUFWAND
z_attribute.name= MODSPEZ_BEGINN
z_attribute.name= MODSPEZ_ENDE
z_attribute.name= CODE_AUFWAND
z_attribute.name= CODE_BEGINN
z_attribute.name= CODE_ENDE
z_attribute.name= INTEGRATION_BEGINN
z_attribute.name= INTEGRATION_ENDE
z_attribute.name= HANDBUCH_AUFWAND

z_attribute.name= HANDBUCH_BEGINN
z_attribute.name= HANDBUCH_ENDE

**Attribute bezüglich Projektzustand:**
zarmstype.z_type = PROJEKTZUSTAND
z_entity = Projectstatus
z_attribute.name= KOSTEN
z_attribute.name= KOSTEN_PRO_MM
z_attribute.name= PROJEKTBGINN
z_attribute.name= PROJEKTENDE
z_attribute.name= AUFWAND_SPEZIFIKATIONSPHASE
z_attribute.name= AUFWAND_ENTWURFSPHASE
z_attribute.name= AUFWAND_CODIERUNGSPHASE
z_attribute.name= AUFWAND_TESTPHASE
z_attribute.name= AUFWAND_HANDBUCHERSTELLUNG
z_attribute.name= AUFWAND_REVIEWS
z_attribute.name= AUFWAND_TESTS
z_attribute.name= AUFWAND_KORREKTUR_REVIEWS
z_attribute.name= AUFWAND_KORREKTUR_TESTS
z_attribute.name= ANZAHL_ABNAHMETESTS
Achtung: AUFWANDSVERTEILUNG_PHASEN
Dieses Attribut enthält ein Array mit Werten, die einzeln gespeichert werden müssen.
z_attribute.name= AUFWANDSVERTEILUNG_SPEZIFIKATION
z_attribute.name= AUFWANDSVERTEILUNG_ENTWURF
z_attribute.name= AUFWANDSVERTEILUNG_CODE
z_attribute.name= AUFWANDSVERTEILUNG_TEST
z_attribute.name= AUFWANDSVERTEILUNG_HANDBUCH

**Attribute bezüglich Reviewlogbuch:**
zarmstype.z_type = REVIEWLOGBUCH
z_entity = Reviewlog
z_attribute.name= SREVIEW_AUFWAND
z_attribute.name= SREVIEW_BEGINN
z_attribute.name= SREVIEW_ENDE
z_attribute.name= SREVIEW_K_AUFWAND
z_attribute.name= SREVIEW_K_BEGINN
z_attribute.name= SREVIEW_K_ENDE
z_attribute.name= EREVIEW_AUFWAND
z_attribute.name= EREVIEW_BEGINN
z_attribute.name= EREVIEW_ENDE

z_attribute.name= EREVIEW_K_AUFWAND
z_attribute.name= EREVIEW_K_BEGINN
z_attribute.name= EREVIEW_K_ENDE
z_attribute.name= MREVIEW_AUFWAND
z_attribute.name= MREVIEW_BEGINN
z_attribute.name= MREVIEW_ENDE
z_attribute.name= MREVIEW_K_AUFWAND
z_attribute.name= MREVIEW_K_BEGINN
z_attribute.name= MREVIEW_K_ENDE
z_attribute.name= CREVIEW_AUFWAND
z_attribute.name= CREVIEW_BEGINN
z_attribute.name= CREVIEW_ENDE
z_attribute.name= CREVIEW_K_AUFWAND
z_attribute.name= CREVIEW_K_BEGINN
z_attribute.name= CREVIEW_K_ENDE
z_attribute.name= HREVIEW_AUFWAND
z_attribute.name= HREVIEW_BEGINN
z_attribute.name= HREVIEW_ENDE
z_attribute.name= HREVIEW_K_AUFWAND
z_attribute.name= HREVIEW_K_BEGINN
z_attribute.name= HREVIEW_K_ENDE

**Attribute bezüglich Spezifikation:**
zarmstype.z_type = SPEZIFIKATION
z_entity = Specification
z_attribute.name= ANZ_AFP
z_attribute.name= ANZ_FEHLER_GES
z_attribute.name= PROZENT_AFP_100
z_attribute.name= ANZ_AF

**Attribute bezüglich Spezifikaitonsreviewbericht:**
zarmstype.z_type = SPEZIFIKATIONSREVIEWBERICHT
z_entity = Specificationreviewreport
z_attribute.name= FEHLER
z_attribute.name= VERLUSTE
z_attribute.name= NUMMER
z_attribute.name= ANZ_FEHLER_PRO_PRF
z_attribute.name= ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Systementwurf:**
zarmstype.z_type = SYSTEMENTWURF

z_entity = Systemdesign
z_attribute.name= ANZ_AFP
z_attribute.name= ANZ_FEHLER_GES
z_attribute.name= PROZENT_AFP_100
z_attribute.name= ANZ_AF
z_attribute.name= ANZ_GF

**Attribute bezüglich Systementwurfsreviewbericht:**
zarmstype.z_type = SYSTEMENTWURFSREVIEWBERICHT
z_entity = Systemdesignreviewreport
z_attribute.name= FEHLER
z_attribute.name= VERLUSTE
z_attribute.name= NUMMER
z_attribute.name= ANZ_FEHLER_PRO_PRF
z_attribute.name= ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Systemtestbericht:**
zarmstype.z_type = SYSTEMTESTBERICHT
z_entity = Systemtestreport
z_attribute.name= FEHLER
z_attribute.name= VERLUSTE
z_attribute.name= NUMMER
z_attribute.name= ANZ_FEHLER_PRO_PRF
z_attribute.name= ANZ_KORR_FEHLER_PRO_PRF

**Attribute bezüglich Testlogbuch:**
zarmstype.z_type = TESTLOGBUCH
z_entity = Testlog
z_attribute.name= MTEST_AUFWAND
z_attribute.name= MTEST_BEGINN
z_attribute.name= MTEST_ENDE
z_attribute.name= MTEST_K_AUFWAND
z_attribute.name= MTEST_K_BEGINN
z_attribute.name= MTEST_K_ENDE
z_attribute.name= ITEST_AUFWAND
z_attribute.name= ITEST_BEGINN
z_attribute.name= ITEST_ENDE
z_attribute.name= ITEST_K_AUFWAND
z_attribute.name= ITEST_K_BEGINN
z_attribute.name= ITEST_K_ENDE
z_attribute.name= STEST_AUFWAND

z_attribute.name= STEST_BEGINN
z_attribute.name= STEST_ENDE
z_attribute.name= STEST_K_AUFWAND
z_attribute.name= STEST_K_BEGINN
z_attribute.name= STEST_K_ENDE
z_attribute.name= ATEST_AUFWAND
z_attribute.name= ATEST_BEGINN
z_attribute.name= ATEST_ENDE
z_attribute.name= ATEST_K_AUFWAND
z_attribute.name= ATEST_K_BEGINN
z_attribute.name= ATEST_K_ENDE

*Anhang E*

# AUTOMATISCHE EINSTELLUNG DER ZARMS-FILTEROPERATION

Damit das System weiß, welche ZARMS-Einträge es aus dem ZARMS-Datenstrom extrahieren muss, werden die zu filternden Attribute vorher aus der Datenbank geladen. Dazu wurde folgender Ansatz überlegt, der aber derzeit noch nicht realisiert wurde.

1. **Holen der Entitäten**
   Im ersten Schritt müssen die Entitäten für das jeweilige Modell aus der Datenbank geholt werden. Dies geschieht mit dem SQL-Statement:

   SELECT ∗
   FROM zarmstype, z_entity, z_attribute, comprises
   WHERE zarmstype.eorr = 'E' AND
   zarmstype.zid = z_entity.zid AND
   z_entity.zeid = comprises.zeid AND
   z_attribute.zaid = comprises.zaid AND
   zarmstype.mid = AKTUELLE MODELL-ID;

   Die Daten müssen anschließend in einer Hashtabelle (entity_ht) gespeichert werden. Diese Hashtabelle (entity_ht) sollte folgendermaßen aufgebaut sein:

| | | |
|---|---|---|
| z_type1 | z_type2 | ... |
| description1 | descripton2 | ... |
| attribute-string1 | attribute-string2 | ... |

   Die Einträge in „z_type“ und „description“ entsprechen dem ZARMS-Pfad. Die Liste in „attribute_string“ beinhaltet alle relevanten Attribute, die es in diesem Pfad gibt.

Ein Beispiel:
z_type1 = PROJEKTLOGBUCH
description1 = Projectlog
attribute_string1 = „SPEZIFIKATION_BEGINN ∗ SPEZIFIKATION_ENDE ∗ ENTWURF_BEGINN ∗ ENTWURF_ENDE ∗ ...“

2. **Holen der Relationen**
   Im zweiten Schritt müssen die Relationen aus der Datenbank geladen werden. Dazu kann das folgende SQL-Statement verwendet werden:

   SELECT ∗
   FROM zarmstype
   WHERE zarmstype.eorr = 'R' AND
   zarmstype.mid = AKTUELLE MODELL-ID;

   Die Daten müssen ebenfalls wieder in einer Hashtabelle (relation_ht) gespeichert werden. Diese Hashtabelle (relation_ht) sollte folgendermaßen aussehen:

| z_type1 | z_type2 | ... |
|---|---|---|
| attribute-string1 | attribute-string2 | ... |

   Der Eintrag in „z_type“ entspricht dem ZARMS-Pfad (aber nur der ersten Ebene). Die Liste in „attribute_string“ beinhaltet alle relevanten Attribute, die es in diesem Pfad gibt.

   Ein Beispiel:
   z_type1 = PRODUZIERT
   attribute_string1 = „WER ∗ WAS ∗ ...“

3. **Laden der ZARMS-Logik**
   Im dritten Schritt muss noch die Logik zu den ZARMS-Daten aus der Datenbank geladen werden. Dazu kann das folgende SQL-Statement verwendet werden:

```
SELECT *
FROM zarms_logic
WHERE mid = AKTUELLE MODELL-ID;
```

Die Daten müssen anschließend in einem Vektor (logic_vector) gespeichert werden. Aus zwei Gründen muss ein Vektor verwendet werden. Zum einen da es kein eindeutiges Schlüsselattribut gibt, und zum anderen muss gewährleistet werden, dass zuerst die „komplexen“ Logikeinträge bearbeitet werden und erst im Anschluss die „einfachen“ Logikeinträge. Der Vektor (logic_vector) sollte folgendermaßen aussehen:

| | | |
|---|---|---|
| relation1 | relation2 | ... |
| rel_attributes1 | rel_attributes2 | ... |
| trigger_rel1 | trigger_rel2 | ... |
| trigger_attributes1 | trigger_attributes2 | ... |
| mapping1 | mapping2 | ... |
| source_lhs1 | source_lhs2 | ... |
| source_rhs1 | source_rhs2 | ... |
| operator1 | operator2 | ... |
| true_action1 | true_action2 | ... |
| false_action1 | false_action2 | ... |

Der Vektor (logic_vector) sollte wie folgt aufgebaut sein:

- **relation:** beinhaltet den Namen der Relation, für die dieser Logikeintrag gilt
- **relation_attributes:** enthält eine Mapping-Liste, die beschreibt welches Attribut der „relation“, welchem Attribut der Tabelle „s_relation“ in der Datenbank entspricht. Die Liste sieht wie folgt aus:
  relation_attribute1 * db_attribute ^ relation_attribute2 * db_attribute ^ ...
- **trigger_rel:** enthält die Relation, deren Werte mit den Werten von „relation“ zusätzlich zum noch folgenden Statement verglichen werden müssen.
- **trigger_attributes:** enthält eine Mapping-Liste, die beschreibt welches Attribut der „trigger_rel“, welchem Attribute der Tabelle s_relation in der Datenbank entspricht. Die Liste sieht wie folgt aus:

trigger_attribute1 ∗ db_attribute ^ trigger_attribute2 ∗ db_attribute ^ ...

- **mapping:** dieser Eintrag enthält eine Liste die beschreibt, wie „relation“ und „trigger_rel“ miteinander verglichen werden müssen. Verglichen wird über die Datenbank-Attribute! Die Liste sieht wie folgt aus:
  db_attribute1 ∗ db_attribute2
  Achtung! Wenn ein db_attribute einen bestimmten Wert haben muss, damit es verglichen werden darf, dann muss dieser Wert in der Liste angegeben werden:
  Die Liste würde dann wie folgt aussehen:
  „person ∗ document # Specification“.
- **operator:** beinhaltet den Operator mit dem „source_rhs“ und „source_lhs“ verknüpft werden.
- **source_rhs:** stellt den rechten Teil des Statements dar
- **source_lhs:** stellt den linken Teil des Statements dar
- **true_action:** enthält die SQL-Query, die abgesetzt wird, wenn das Statement („source_lhs“ „operator“ „source_rhs“) wahr ist und es diesbezüglich bereits einen Eintrag in der Datenbank gibt. In diesem Fall ist nur ein UPDATE des „completion_date“-Eintrags auf das aktuelle Datum notwendig.
- **false_action:** enthält die SQL-Query, die abgesetzt wird, wenn das Statement („source_lhs“ „operator“ „source_rhs“) wahr ist und es diesbezüglich noch keinen Eintrag in der Datenbank gibt. In diesem Fall ist ein INSERT auszuführen.

Ein einfachstes Beispiel:
relation: „PRODUZIERT“
rel_attributes: „WER ∗ person ^ WAS ∗ document“
trigger_rel:
trigger_attributes:
mapping: „person ∗ document # Specification“
operator:= „>“
source_lhs: „SPEZIFIKATION_ENDE“
source_rhs: „SYSTEM_DATE“
true_action: „UPDATE completion_date ... „
false_action: “INSERT .... success = 1.... „

Wenn „trigger_rel“ leer ist, dann werden die Attribute aus „relation“ nach der Mapping-Liste (rel_attributes) verglichen. Danach wird das Statement zusammengebaut und ausgeführt. Ist das Statement wahr, dann wird geprüft, ob es bereits einen Eintrag in der Datenbank gibt. Sollte das der Fall sein, dann wird die „true_action“ ausgeführt, sonst die „false_action“.

4. **Laden der Phasenenden**
   Im nächsten Schritt müssen noch alle Phasenenden aus der Datenbank geladen werden. Diese Werte werden für die Auswertung der „zarms_logic“-Statements benötigt. Die Phasenenden werden in der Tabelle „phase_state“ gespeichert. Jedes Phasenende besitzt einen Namen („name“) und einen Pfad („z_path“) unter dem es in der Datenbank gefunden werden kann. Bei der Initialisierung des Wrappers[1] sollten also alle Phasenenden aus der Tabelle „phase_state“ in einer Hashtabelle (phase_state_ht) geladen werden. Dieser Vektor sollte wie folgt aufgebaut sein:

| | | |
|---|---|---|
| name1 | name2 | ... |
| z_path1 | z_path2 | ... |

   Beispiel:
   name = ENTWURF_ENDE
   z_path = PROJEKTLOGBUCH ∗ Projectlog ∗ ENTWURF_ENDE

5. **Erstellen der Hashtabelle für den Spieler**
   Zuletzt wird noch eine Hashtabelle (player_ps_ht) benötigt, die die aktuellen Phasenenden des Spielers aktualisiert und speichert. Diese Hashtabelle wird bei jedem Spielzug im jeweiligen CommandObjekt verpackt und vom Client an den Load Balance Manager[2] (LBM) übergeben. Dort werden die Phasenenden aktualisiert und nach der Verarbeitung der ZARMS-Daten wieder an den Client retourniert.

[1] Der Wrapper ist die Komponente des AMEISE-Systems, die den SESAM-Simulator enthält. Alle Benutzerkommandos werden vom Client an den Wrapper übermittelt, der sie an den SESAM-Simulator weiterleitet. Der Simulator verarbeitet die Befehle und generiert eine Antwort, die anschließend durch den Wrapper wieder zurück an den Client übermittelt wird.

[2] Der Load Balance Manager wurde eingeführt, um die Mehrbenutzerfähigkeit des AMEISE-Systems zu realisieren. Er hat die Aufgabe die Last mehrerer Clients auf mehrere Wrapper aufzuteilen.

### Vorgehen beim Verarbeiten eines „Proceeds“

Zuerst wird das CommandObjekt an den Wrapper weitergeleitet und dort verarbeitet. Nach der erfolgreichen Verarbeitung des Kommandos hat der LBM die Aufgabe die aktuellen ZARMS-Daten zu verarbeiten. Dabei sollte er wie folgt vorgehen:

1. Im ersten Schritt durchläuft er den ZARMS-Entitätenstring. Zuerst wird der Wert der Entität in die Datenbank geschrieben. Danach muss noch geprüft werden, ob die Entität auch in der „player_ps_ht“ vorkommt. Wenn ja, dann muss der Wert in der Hashtabelle aktualisiert werden. Sollte die Entität nicht in der Hashtabelle vorkommen, dann handelt es sich nicht um ein relevantes Phasenende.

2. Im zweiten Schritt muss der ZARMS-Relationenstring durchlaufen werden.
   Beispiel: Relationenstring
   .....§ REVIEW_MIT_KUNDE_FINDET_STATT ∗ REVIEW_MIT_KUNDE_FINDET_STATT#1 ^ Prüfling ∗ Specification ^ Gutachter1 ∗ Axel ^ Gutachter2 ∗ Bernd § REVIEW_MIT_KUNDE_FINDET_STATT ∗ REVIEW_MIT_KUNDE_FINDET_STATT#2 ^ Prüfling ∗ Manuals ^ Gutachter1 ∗ Diana ^ Gutachter2 ∗ Thomas § ..... § BEGUTACHTET ∗ BEGUTACHTET#1 ^ WER ∗ Axel ^ WAS ∗ Specification § BEGUTACHTET ∗ BEGUTACHTET#2 ^ WER ∗ Bernd ^ WAS ∗ Specification § BEGUTACHTET ∗ BEGUTACHTET#3 ^ WER ∗ Diana ^ WAS ∗ Manuals § BEGUTACHTET ∗ BEGUTACHTET#4 ^ WER ∗ Thomas ^ WAS ∗ Manuals § .....

   Dabei wird wie folgt vorgegangen: Zuerst wird geprüft, ob es für die Relation (zarmstype.z_type) einen Eintrag im logic_vector gibt. Wenn ja, dann können zwei Fälle unterschieden werden:
   1. Fall: einfache Logik
   2. Fall: komplexe Logik

   Im ersten Fall wird das Statement aus „source_lhs“, „operator“ und „source_rhs“ zusammengebaut. Dazu müssen die Werte aus der „player_ps_ht“ entnommen werden. Sollte in „source_lhs“ oder in „source_rhs“ der String „SYSTEM_DATE“ enthalten sein, dann muss dieser durch das aktuelle Simulationsdatum ersetzt werden. Ist dieses Statement

wahr, dann muss noch geprüft werden, ob der Eintrag bereits in der Datenbank existiert. Sollte dies der Fall sein, dann wird das SQL-Statement in der „true_action“ ausgeführt, sonst das in der „false_action“.

Sollte das Statement falsch sein, dann passiert nichts d.h. dieser Logikeintrag wurde richtig abgearbeitet. Nun muss weiter im „logic_vector“ nach einem passenden Eintrag gesucht werden. Wird noch einer gefunden, dann beginne von vorne. Gibt es keinen weiteren Logikeintrag mehr, dann gehe weiter zur nächsten Relation im ZARMS-Relationenstring.

Ein Beispiel für den zweiten Fall könnte wie folgt aussehen:
relation: „BEGUTACHTET“
rel_attributes:“WER ∗ person ^ WAS ∗ document“
trigger_rel: „REVIEW_MIT_KUNDE_FINDET_STATT“
trigger_attributes: „PRUEFLING ∗ document ^ GUTACHTER1 ∗ person ^GUTACHTER2 ∗ person“
mapping: „person ∗ document # Specification“
operator:= „>“
source_lhs: „SREVIEW_ENDE“
source_rhs: „SYSTEM_DATE“
true_action: „UPDATE .... „
false_action: „INSERT .... „

Die Ausführung ist in diesem Fall recht aufwendig. Wenn die Relation „BEGUTACHTET“ im ZARMS-Relationenstring auftritt, werden die Attribute nach der Mapping-Liste (rel_attributes) in zwei Vektoren eingetragen, d.h. ein Vektor für das Datenbankattribut „person“ und ein Vektor für „document“. Dann wird nach der Relation „REVIEW_MIT_KUNDE_FINDET_ STATT“ im ZARMS-Relationenstring gesucht. Falls diese gefunden wird werden ihre Attribute nach der Mapping-Liste (trigger_attributes) in zwei Vektoren eingetragen. Auch hier wird wieder ein Vektor für „person“ und ein Vektor für „document“ verwendet. Dies deutet allerdings darauf hin, dass dieser Ansatz nur solange eingesetzt werden kann, solange es nur zwei „Vergleichsattribute“ (person und document) gibt. Nun muss in den beiden Relationen („BEGUTACHTET“ und „REVIEW_MIT_KUNDE_FINDET_ STATT“) das gemeinsame Vorkommen von Attributen geprüft werden. Dazu wird der „mapping“-Eintrag verwendet. In unserem Fall besteht der „mapping“-Eintrag aus „person ∗ document # Specification“, d.h.

im ersten Schritt wird jede Person in „relation“ mit jeder Person in „trigger_rel“ mit Hilfe der vorher angelegten Vektoren verglichen. Beinhalten beide dieselbe Person, dann wird im zweiten Schritt verglichen, ob das Dokument auch in beiden Fällen die Spezifikation ist.

Gibt es eine Übereinstimmung, dann kann nun auch noch das Logik-Statement ausgeführt werden. Dazu muss, wie bereits im Fall 1 geschildert, aus „source_lhs“, „operator“ und „source_rhs“ ein Statement zusammengebaut und anschließend ausgewertet werden. Ist das Statement wahr, dann wird geprüft, ob es diesbezüglich bereits einen Eintrag in der Datenbank gibt. Ist das der Fall, dann wird die SQL-Anweisung in „true_action“ ausgeführt, sonst die in „false_action“.
Anmerkung: Sollte keine Übereinstimmung gefunden werden, dann wird der zweite Teil des Logikeintrags (das Statement) nicht mehr ausgeführt.

Wenn keine Übereinstimmung gefunden wurde, dann muss im ZARMS-Relationenstring nach einem weiteren „REVIEW_MIT_KUNDE_FINDET_STATT“-Eintrag gesucht werden. Wird einer gefunden, dann beginne von vorne. Gibt es keinen mehr und wurde noch keine Übereinstimmung gefunden, dann gehe im „logic_vector“ weiter und suche dort nach dem nächsten passenden Logikeintrag. Gibt es auch keinen Logikeintrag mehr, dann ist die Abarbeitung dieser Relation abgeschlossen und es kann die nächste Relation im ZARMS-Relationenstring überprüft werden.

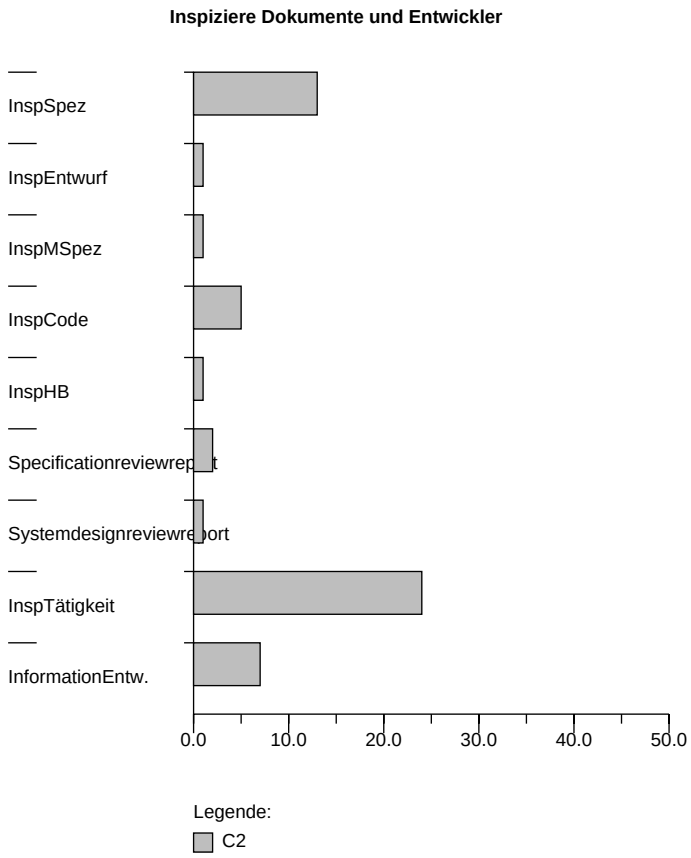
**Anmerkung zu den SQL-Statements in true/false_action**
Wenn eine Relation zum ersten Mal auftritt, dann wird ein neuer Eintrag (mittels INSERT) in die Datenbank geschrieben. Dabei wird das Beginndatum (starting_date) und das Endedatum (completion_date) der Relation auf das aktuelle Simulationsdatum gesetzt. Beim nächsten „Proceed“ des Spielers gibt es die Relation immer noch im ZARMS-Relationenstring. Nun existiert aber auch bereits ein Eintrag in der Datenbank. Das bedeutet, dass das Endedatum (completion_date) dieser Relation auf das aktuelle Simulationsdatum (mittels UPDATE) gesetzt werden muss. Außerdem wird durch das Statement im Logik-Eintrag geprüft, ob eine Relation erfolgreich durchgeführt werden kann oder nicht. Dementsprechend wird in der true/false_action dann das „success“-Flag auf 1 oder 0 gesetzt. Für die Auswertung ist es außerdem entscheidend zu wissen, ob der Kunde am Spezifikationsreview oder am Handbuchreview teilgenommen hat. Um dies festzustellen, muss

für jeden „BEGUTACHTET“-Eintrag geprüft werden, ob es auch einen „REVIEW_MIT_KUNDE_FINDET_STATT“-Eintrag gibt, in dem dasselbe Dokument und derselbe Mitarbeiter eingetragen ist. Ist dies der Fall, so kann nach der Überprüfung des SQL-Statements des Logikeintrages, das „customer“-Flag auf 1 gesetzt werden, sonst bleibt es 0.

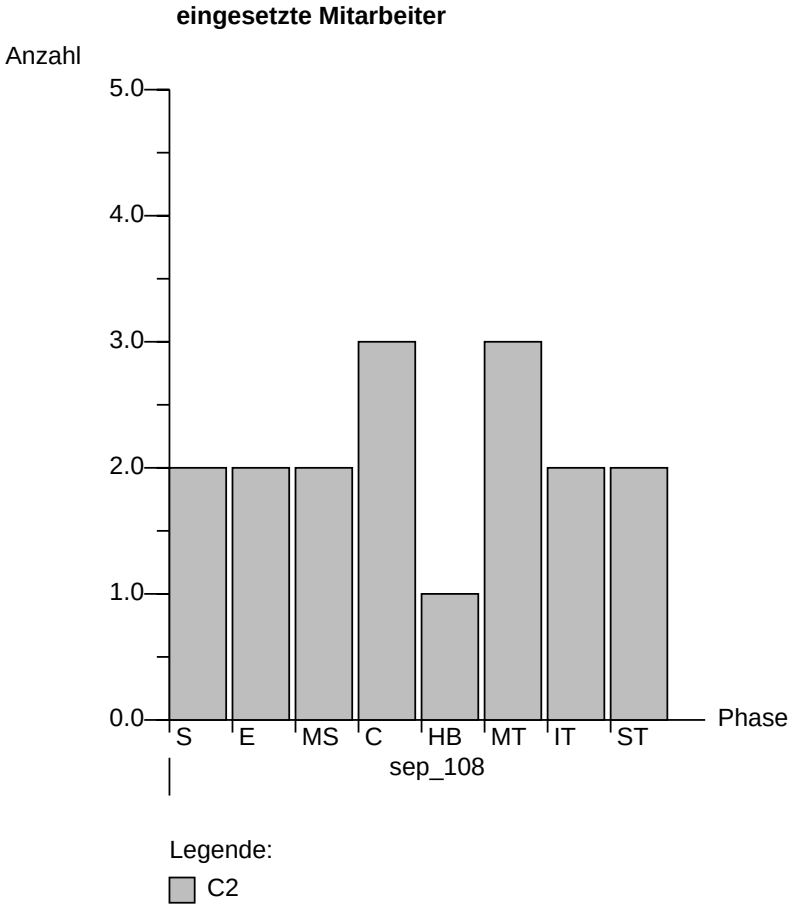
*Anhang F*

# AUSWERTUNG EINES SPIELS MIT SESAMSCORE

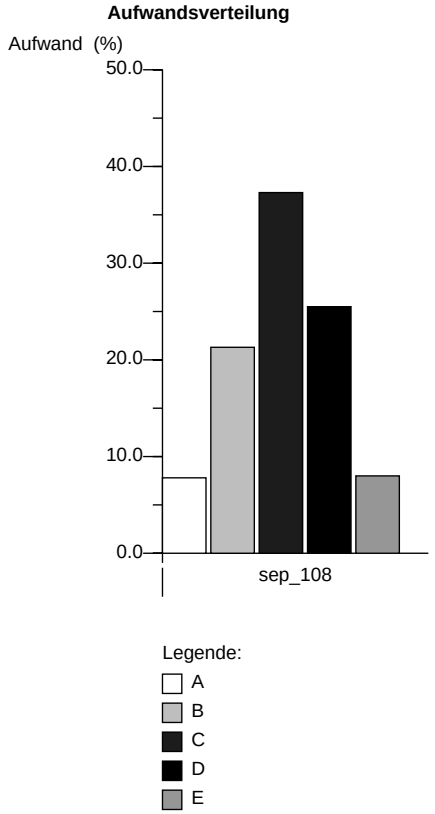
## *Endauswertung Spieler „sep_108“*




+ Anzahl_Inspektionen
  Als Balkendiagramm wird dargestellt, wie oft der Spieler die einzelnen Dokumente inspiziert hat.



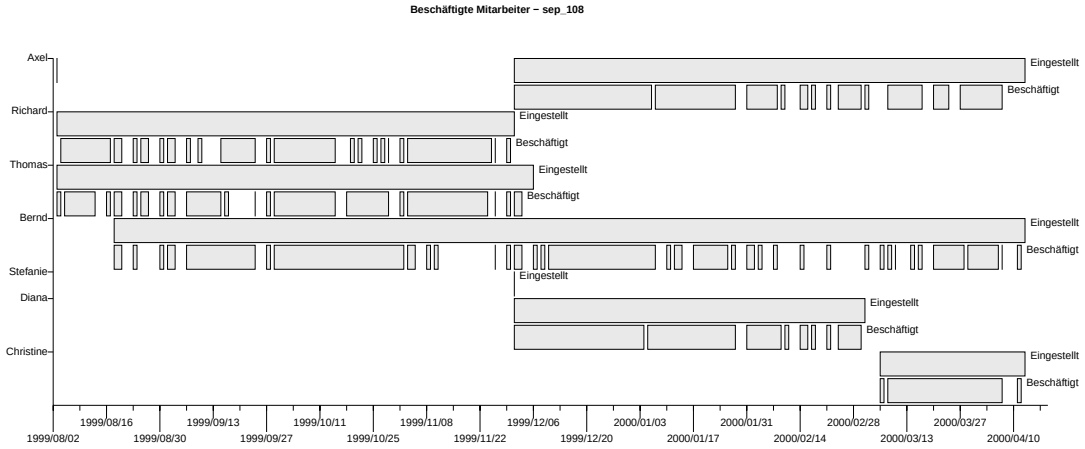

```
+ Anzahl_Mitarbeiter_pro_Phase
  Das Balkendiagramm zeigt, wie viele Mitarbeiter an der Erstellung
  der verschiedenen Dokumente beteiligt waren.

  Aliase
        C2      Anzahl
```



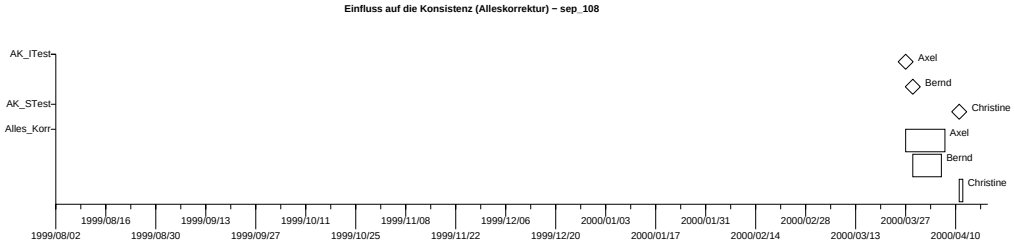

+ Aufwandsverteilung_Balken
  Die Aufwandsverteilung in Prozent wird für die Phasen
  Spezifikation, Entwurf, Codierung, Tests und Handbucherstellung
  gezeigt.

  Aliase

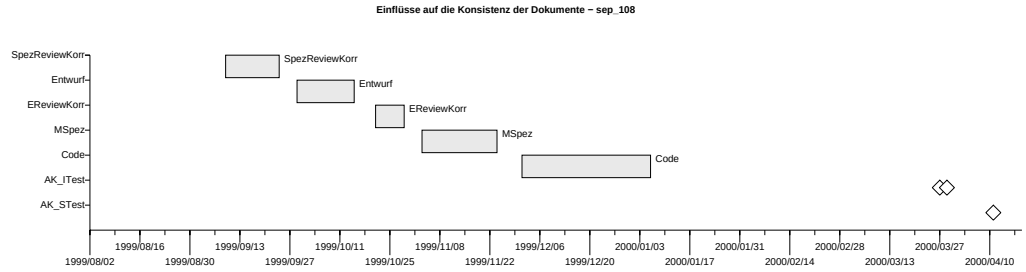
| | |
|---|---|
| A | Spezifikation |
| B | Entwurf |
| C | Codierung |
| D | Test |
| E | Handbuch |




\+ Beschaeftigte_MA
Dieses Ganttdiagramm zeigt, wann Mitarbeiter eingestellt und entlassen wurden, und wann die eingestellten Mitarbeiter beschäftigt waren.




\+ Einfluss_Konsistenz_AK
Das Ganttdiagramm zeigt, wann eine Alleskorrektur stattgefunden hat und nach welchem Review die Korrektur durchgeführt wurde.




+ Einfluss_Konsistenz
  Das Ganttdiagramm zeigt die Reviewphasen und die Korrektur der Dokumente. Die Alleskorrektur wird im Überblick dargestellt.

**Einfluss unproduktiver Zeit**

| | sep_108 |
|---|---|
| K | 415560.0 |
| | |
| A | |
| | 17.36 |
| k | |
| | 23938.0 |

+ Einfluss_unprod_Zeit
  Die Tabelle zeigt die Gesamtkosten, den Gesamtaufwand und die Kosten pro Personenmonat an.

**Gefundene Fehler in Prüf– und Korrekturmaßnahmen**

| | sep_108 | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | C | D | E | F | G | H |
| SpezReview | 73.51 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 74 | | |
| | | | | | | 0.0 | |
| SpezReviewKorr | 67.63 | 0.0 | 0.0 | 0.0 | | | true |
| EReview | 16.69 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 17 | | |
| | | | | | | 109.264 | |
| EReviewKorr | 14.99 | 0.0 | 0.0 | 0.0 | | | false |
| MReview | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 0 | | |
| | | | | | | 0.0 | |
| MReviewKorr | 0.0 | 0.0 | 0.0 | 0.0 | | | true |
| CReview | 40.17 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 41 | | |
| | | | | | | 59.4204 | |
| CReviewKorr | 29.44 | 0.0 | 0.0 | 0.0 | | | false |
| HBReview | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 0 | | |
| | | | | | | 0.0 | |
| HBReviewKorr | 0.0 | 0.0 | 0.0 | 0.0 | | | true |

+ Fehler_Prüf_Korr_Massnahmen
  Die Tabelle zeigt für Reviews der verschiedenen Dokumente an, wie viele Fehler (in bis zu vier Reviews für dasselbe Dokument) gefunden wurden, wie viele Fehler insgesamt im Dokument durch Reviews gefunden wurden, wie viele Fehler korrigiert wurden, den geprüften Umfang in AFP und ob die Korrektur vollständig durchgeführt wurde.

  Aliase

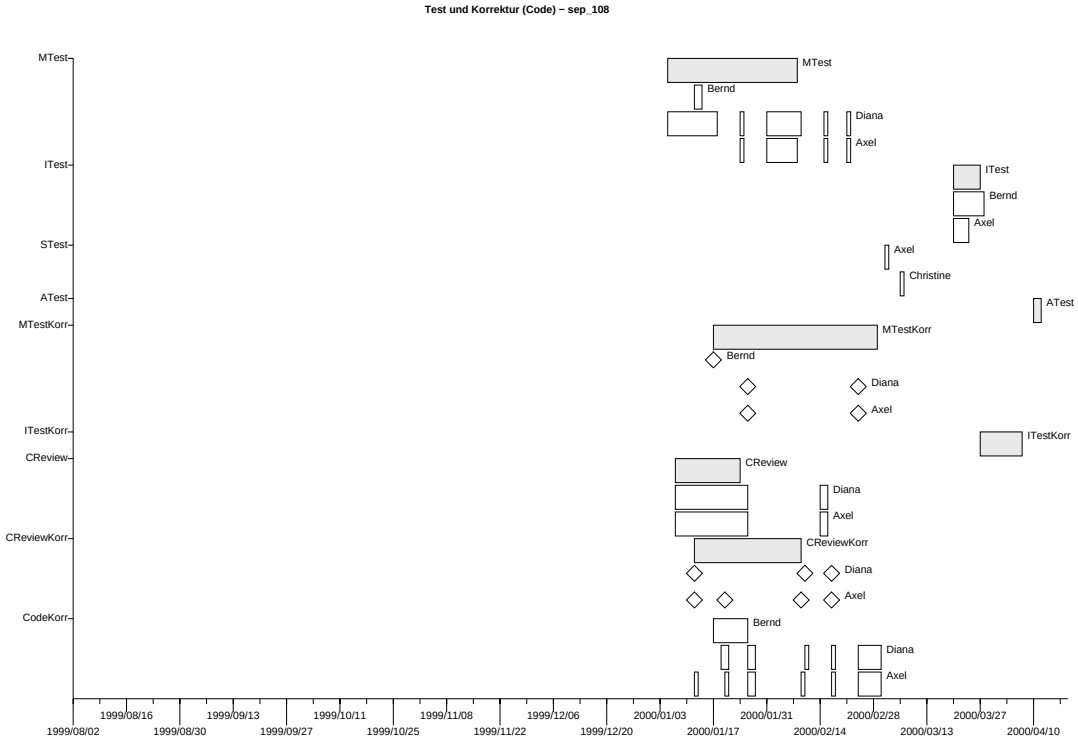
  | | |
  |---|---|
  | B | #F. 1 |
  | C | # F. 2 |
  | D | # F. 3 |
  | E | # F. 4 |
  | F | # F. insgesamt |
  | G | AFP |
  | H | vollständig |

**Gefundene Fehler in Test− und Korrekturmaßnahmen**

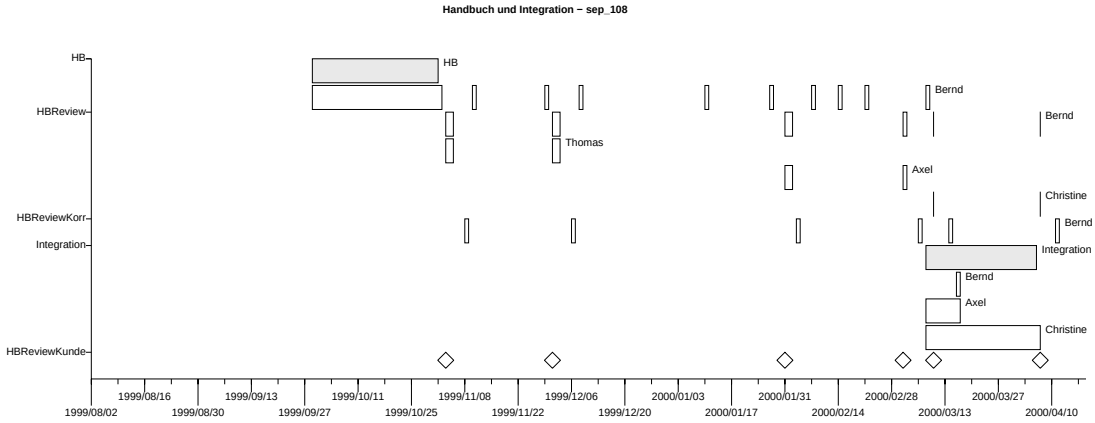
| | sep_108 | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | C | D | E | F | G | H |
| MTest | 120.1 | 49.22 | 0.0 | 0.0 | | | |
| | | | | | 170 | | |
| | | | | | | 0.0 | |
| MTestKorr | 104.95 | 38.43 | 0.0 | 0.0 | | | true |
| ITest | 47.26 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 48 | | |
| | | | | | | 0.0 | |
| ITestKorr | 37.33 | 0.0 | 0.0 | 0.0 | | | true |
| STest | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 0 | | |
| | | | | | | 0.0 | |
| STestKorr | 0.0 | 0.0 | 0.0 | 0.0 | | | true |
| ATest | 37.12 | 0.0 | 0.0 | 0.0 | | | |
| | | | | | 38 | | |
| | | | | | | 188.273 | |
| ATestKorr | 0.0 | 0.0 | 0.0 | 0.0 | | | false |

+ Fehler_Test_Korr_Massnahmen
  Die Tabelle zeigt für die unterschiedlichen Tests jeweils
  gefundene und korrigierte Fehler, den geprüfter Umfang in AFP
  und ob die Korrektur vollständig war.

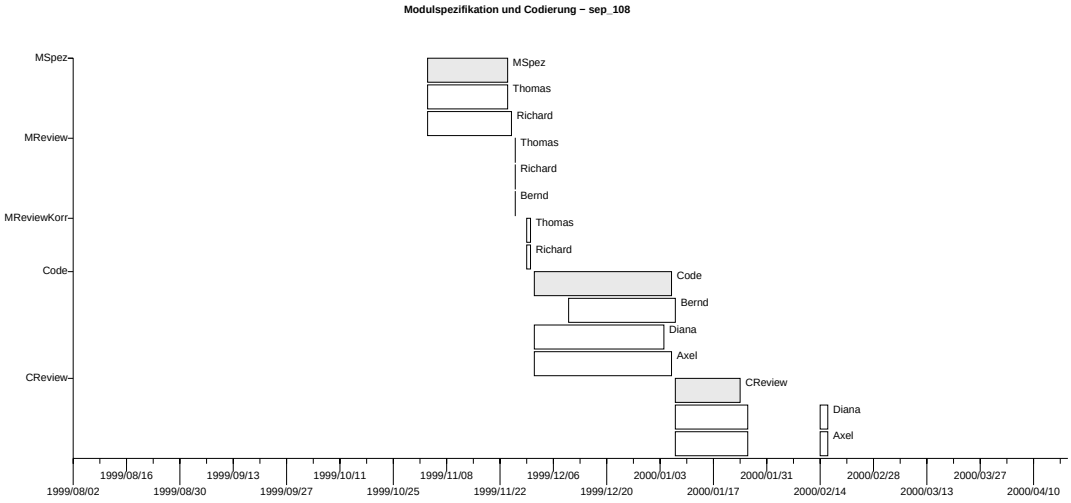
  Aliase
  B # F. 1
  C # F. 2
  D # F. 3
  E # F. 4
  F # F. insgesamt
  G AFP
  H vollständig durchgeführt



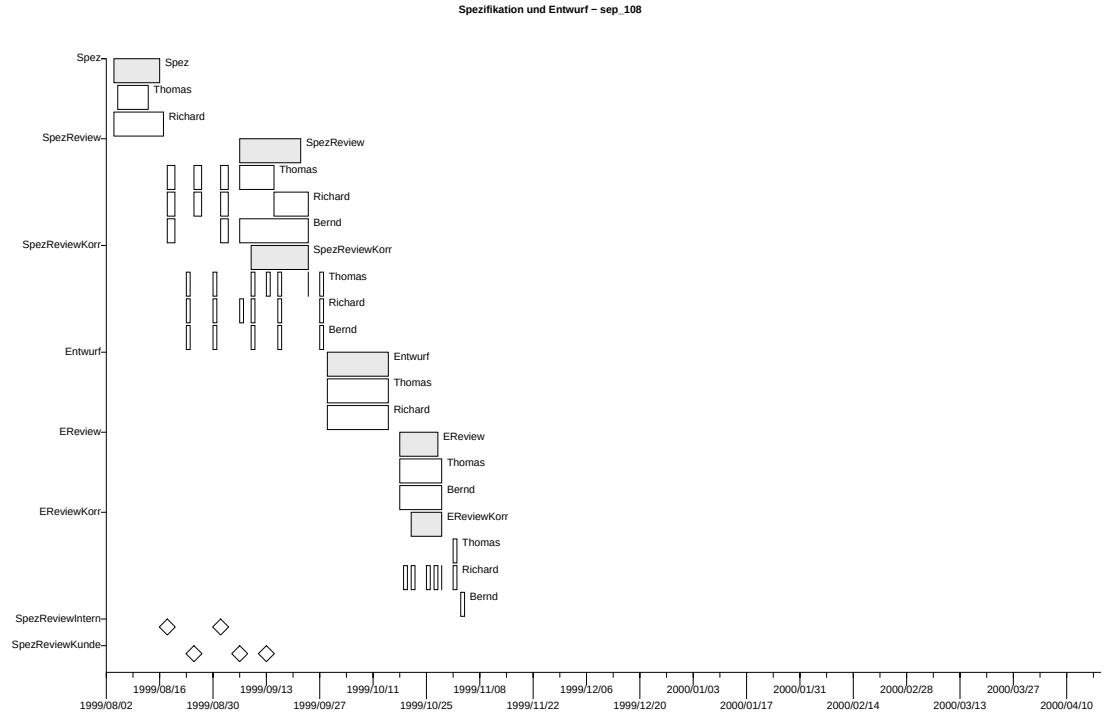

+ Gantt_Code_Test
  Das Ganttdiagramm zeigt alle Tests des Codes (Modultest, Integrationstest, Systemtest und Abnahmetest). Die Tätigkeiten werden jeweils für die gesamte Phase und aufgeteilt nach Entwicklern dargestellt.



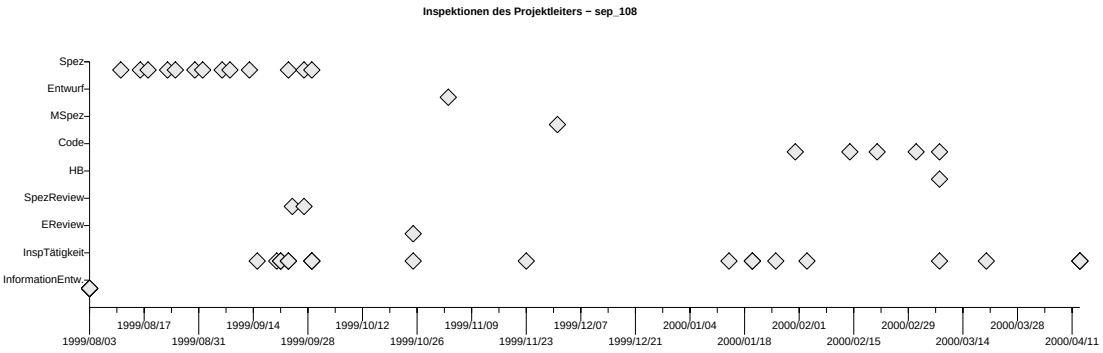

+ Gantt_HB_Int
  Im Diagramm werden die Phasen Integration und Erstellung des Handbuchs gezeigt, für die gesamte Phase und aufgetrennt nach Entwicklern. Für das Handbuch wird zusätzlich Review und Korrektur dargestellt.



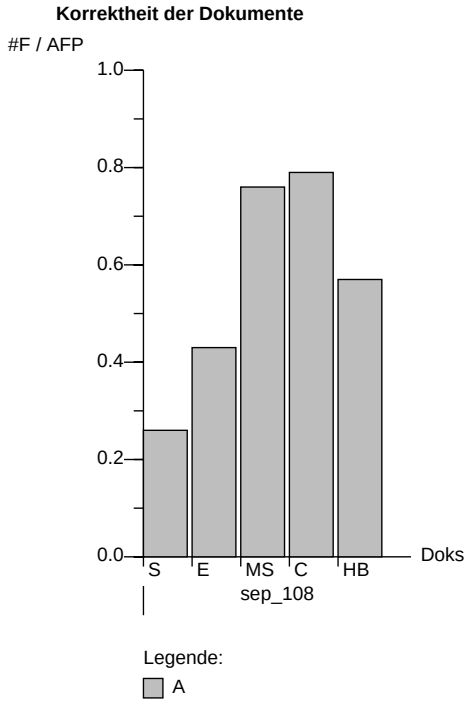

+ Gantt_MSpec_Code
  In diesem Ganttdiagramm werden Feinentwurf und Codierung gezeigt, jeweils für die gesamte Phase und aufgetrennt nach beteiligten Entwicklern. Feinentwurfsreview und Codereview werden jeweils mit Korrektur gezeigt.




+ Gantt_Spec_Entwurf
  Das Ganttdiagramm zeigt die Phasen Spezifikation und Grobentwurf. Dargestellt wird jeweils die Dauer der gesamten Phase und zusätzlich die Tätigkeiten für die einzelnen Mitarbeiter. Review und Korrektur der Spezifikation und des Grobentwurfs werden dargestellt.

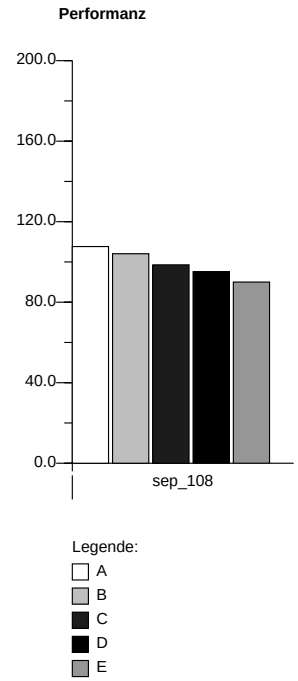



+ Inspect_all
  In diesem Diagramm werden alle Kommandos zur Inspektion des Projekts gezeigt.

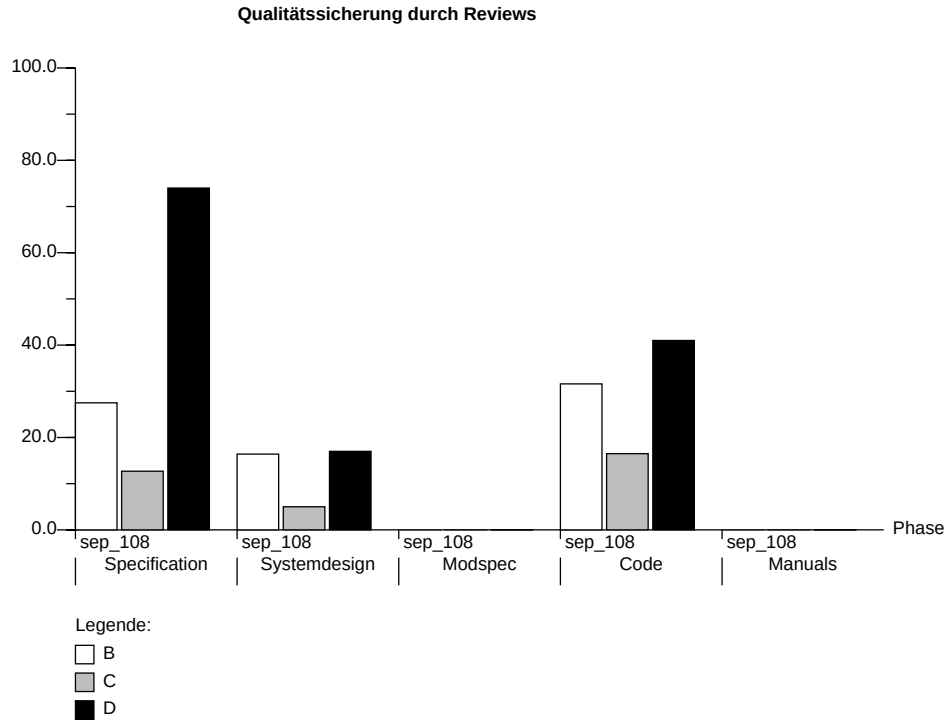



+ Korrektheit_Doks_Balken
  Für die Spielverläufe wird ein Balkendiagramm mit der Anzahl Fehler pro AFP für die entstandenen Dokumente erstellt.

  Aliase
  A Fehler pro AFP (Spez, Entwurf, ModulSpez, Code, Handbuch)

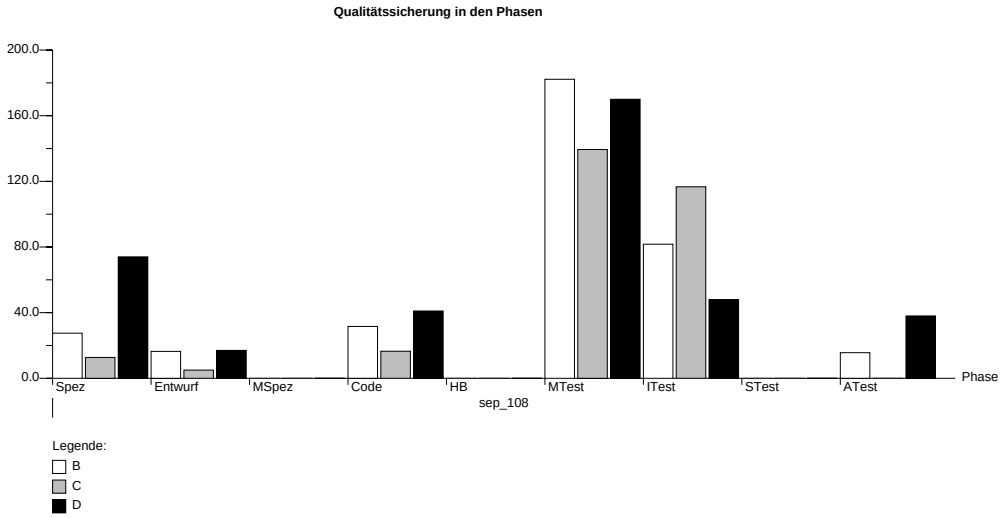



```
+ Performanz_Balken
  Das Balkendiagramm gibt einen Überblick über die erreichten
  Zielvorgaben: Kosten und Dauer in Prozent, Mittelwerte für
  Vollständigkeit und Korrektheit werden angezeigt.

  Aliase
        A       Zielvorgaben Kosten (%)
        B       Zielvorgaben Dauer (%)
        C       Mittelwert Vollständigkeit Zielvorgaben
        D       Mittelwert Vollständigkeit
        E       Mittelwert Korrektheit Zielvorgaben
```



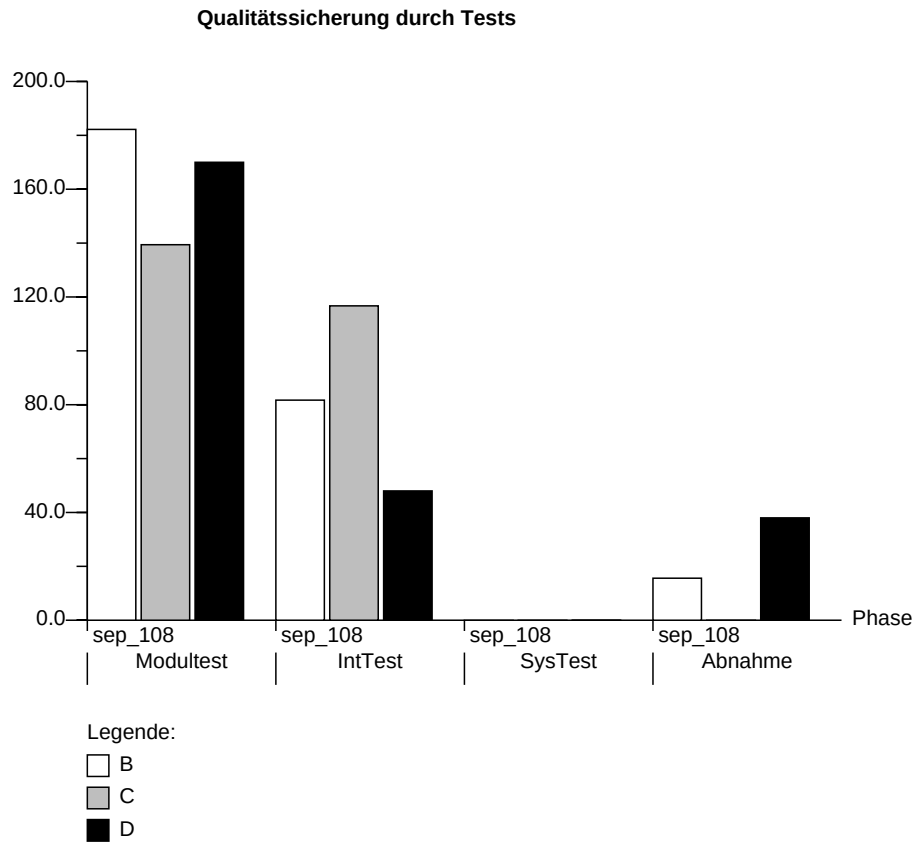

+ QS_Reviews_Phasen
  Die Tabelle zeigt den Aufwand für Prüfungen durch Reviews,
  den Aufwand für die Korrektur und die im Review gefundene Fehler,
  jeweils für die unterschiedlichen Reviews.

  Aliase
  
  | | |
  |---|---|
  | B | Prfg. Aufwand |
  | C | Korr. Aufwand |
  | D | # gefundene Fehler |




+ QS_Spieler_Phasen
  Für einen Spieler werden für die Reviews und Tests jeweils
  Aufwand für die Prüfung, Aufwand für die Korrektur und die
  Anzahl der gefundenen Fehler gezeigt.

  Aliase
  B  Prfg. Aufwand
  C  Korr. Aufwand
  D  # gefundene Fehler

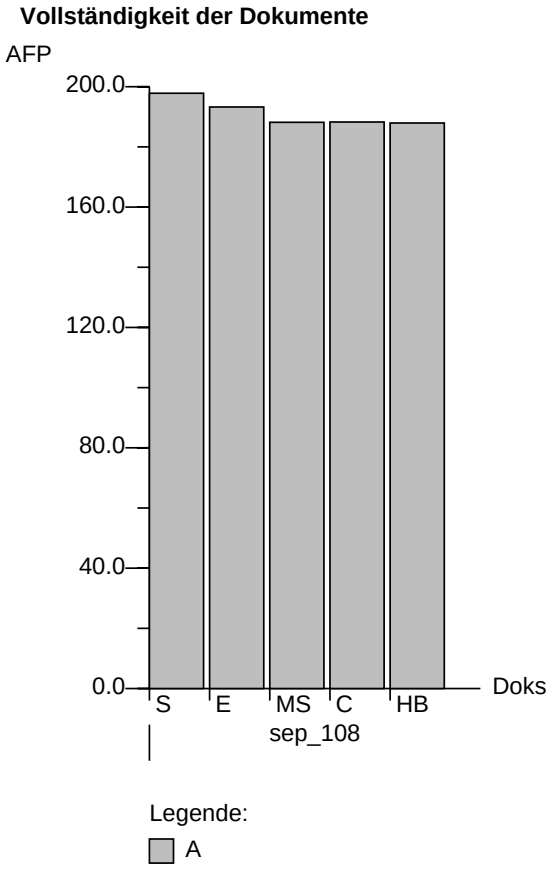



+ QS_Tests_Phasen_Balken
  Das Balkendiagramm zeigt den Aufwand für Tests, für Korrektur und die durch den Test gefundenen Fehler.

**Restfehler im Dokument nach Fehlerarten**

| | | Spez | Entwurf | MSpez | Code | HB |
|---|---|---|---|---|---|---|
| sep_108 | A | 51 | 46 | 45 | 45 | 48 |
| | G | | 37 | 38 | 31 | |
| | F | | | 60 | 31 | |
| | I | | | | 41 | |
| | H | | | | | 59 |

```
+ Restfehler_1_VP
  Die Tabelle zeigt die Fehler in den verschiedenen Dokumenten nach
  Abschluss des Projekts. Die Fehler werden aufgeschlüsselt nach
  Fehlerart (Analysefehler, Grobentwurfsfehler, Feinentwurfsfehler,
  Implementierungsfehler und Handbuchfehler) gezeigt.

  Aliase
        A       AF
        G       GF
        F       FF
        I       IF
        H       HF
```




+ Vollständigkeit_Doks_Balken
  Das Balkendiagramm zeigt den erreichten Umfang in AFP für die verschiedenen Dokumente.

  Aliase
  A AFP (Spez, Entwurf, ModulSpez, Code, Handbuch)

**Erreichen der Zielvorgaben**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| sep_108 | 259 | 415560.0 | | | | |
| | | | 93.72 | | | |
| | | | | 16.04 | | |
| | | | | | 93.56 | |
| | | | | | | 0.43 |

+ Zielvorgaben
  Das Schema gibt einen Überblick über das Erreichen der Zielvorgaben. Es werden Dauer und Aufwand, erreichter Umfang in AFP und die Anzahl der Fehler für den Code und das Handbuch ausgegeben.

  Aliase

| | |
|---|---|
| A | Dauer |
| B | Kosten |
| C | AFP C % |
| D | #F C/KLOC |
| E | AFP HB % |
| F | #F HB/Seite |

# GLOSSAR

### Adjusted Function Points (AFPs)

Das Function-Point-Verfahren wurde erstmals von Albrecht (1979) vorgestellt. Die Idee besteht darin, die Spezifikation mittels so genannter Adjusted Function Points (AFP) zu bewerten, um daraus den für das gesamte Software-Entwicklungsprojekt zu erwartenden Aufwand abzuleiten. Dabei hilft eine durch Auswertungen bereits abgeschlossener Projekte gewonnene Tabelle, in der jeweils die Zahl der geforderten AFPs den tatsächlich für die Entwicklung des Systems entstandenen Aufwand gegenübergestellt wird.

### Anforderungen

Im QS-Modell wird der Inhalt von Dokumenten durch Anforderungen repräsentiert. Die Komplexität der einzelnen Anforderungen wird in AFPs beschrieben.

### Arbeitspaket

Arbeitspakete enthalten eine Menge von Anforderungen, die von Entwicklern bearbeitet werden können. Im QS-Modell werden Arbeitspakete genutzt, um zu ermöglichen, dass mehrere Entwickler ein Dokument gleichzeitig bearbeiten können, ohne sich gegenseitig zu stören.

### Auswertungswerkzeuge

Für ein SESAM-Spiel stehen SesamScore und SesamAlyzer als Auswertungswerkzeuge zur Verfügung. In AMEISE wurden diese Werkzeuge durch eine generische Erklärungskomponente ersetzt.

### Fehler

Fehler, die in Dokumenten enthalten sind, werden im QS-Modell nach Fehlerart aufgeschlüsselt.

### Fehlerart

Durch die Fehlerart wird unterschieden, bei welcher Tätigkeit ein Fehler eingefügt wurde. Im QS-Modell existieren folgende Fehlerarten: Analysefehler, Grobentwurfsfehler, Feinentwurfsfehler, Implementierungsfehler und Handbuchfehler.

### Hilfsmittel

Zu den Hilfsmitteln des AMEISE-Spiels zählen Ratgeber, väterlicher Freund, Vergleich, Rollback und Erklärungskomponente.

### Hilfsmittelkomponente/spezielle Hilfsmittelkomponente

Jedes Bewertungskriterium wird durch spezielle Hilfsmittelkomponenten im System repräsentiert, die die möglichen Ausprägungen des Bewertungskriteriums darstellen. Die speziellen Hilfsmittelkomponenten sind modellabhängig, das bedeutet sie können nur für ein bestimmtes Modell bzw. für verschiedene Instanzen eines Modells eingesetzt werden, die auf denselben ZARMS-Daten beruhen.

### Mängel

Zu den Mängeln eines Dokuments zählen Fehler bzw. fehlende AFPs, die von den Gutachtern bei der Prüfung eines Dokuments aufgedeckt werden.

### Mitarbeiter: Synonym für Entwickler

Der Spieler hat im QS-Modell die Möglichkeit bis zu sieben Mitarbeiter für die Durchführung des Projekts einzusetzen. Er kann die Mitarbeiter einstellen, ihnen Tätigkeiten zuteilen und sie wieder entlassen.

### Modellbauer

Der Modellbauer ist für alle modellrelevanten Aspekte eines AMEISE-Spiels verantwortlich. Dazu zählt das Einspielen bzw. Löschen von neuen Modellen und Dolmetscher-Dateien sowie das Erweitern der speziellen Hilfsmittelkomponenten.

### Modulspezifikation: Synonym für Moduldesign

Bei der Modulspezifikation bzw. beim Moduldesign handelt es sich um den Feinentwurf.

### Proceed

Der Befehl „Proceed“ führt zum Fortschreiten der Simulation um eine Zeiteinheit. Im QS-Modell handelt es sich bei einer Zeiteinheit um einen Tag.

### Prüfobjekt

Als Prüfobjekt wird ein Teil eines Dokuments bezeichnet, der von Gutachtern im Rahmen eines Reviews geprüft werden soll.

### Referenzdokument

Als Referenzdokument wird das Vorgabedokument einer Phase verstanden, gegen das die Gutachter das Prüfobjekt vergleichen. Für einen Review des Grobentwurfsdokuments zum Beispiel stellt das Spezifikationsdokument die Vorgabe dar.

### Spiel: Synonym für Projekt

Unter Spiel versteht man das Projekt, dass vom Spieler im Rahmen der Simulation durchgeführt werden soll.

### Spieler

Bei einem Spieler handelt es sich entweder um einen Einzelspieler oder eine Gruppe von Spielern, die ein simuliertes Projekt durchführen.

### Spielverlauf: Synonym für Projektverlauf

Der Spielverlauf zeigt die einzelnen Entscheidungen des Spielers auf, die vom Tutor bzw. von der generischen Erklärungskomponente ausgewertet werden können.

### Tätigkeit

Die Entwickler führen während der Simulation Tätigkeiten aus. Zu diesen Tätigkeiten zählen das Erstellen, Korrigieren, Testen und Prüfen von Dokumenten.

**Tutor: Synonym für Lehrveranstaltungsleiter**

**ZARMS (Zustandsanalyse und Regel - Monitor für SESAM)**

ZARMS ermöglicht die Anzeige aller Entitäten und Relationen des aktuellen Zustands der Simulation. Zusätzlich werden die Werte der Attribute eines Objekts angezeigt.

# LITERATURVERZEICHNIS


[Boehm] Boehm, B. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, 1981.

[Drappaa] Drappa, A. *Beschreibung der Implementierung des QS-Modells*. Universität Stuttgart, Institut für Informatik, projektinternes Implementierungsdokument, Februar 1998.

[Drappab] Drappa, A. *Quantitative Modellierung von Softwareprojekten - Eine Materialsammlung*. Universität Stuttgart, Institut für Informatik, interner Bericht, 1997.

[Drappac] Drappa, A. *Quantitative Modellierung von Softwareprojekten. Doctoral Dissertation*. Shaker-Verlag, Aachen, 2000.

[Drappad] Drappa, A.; Ludewig, J. *Quantitative Modelierung for the Interactive Simulation of Software Projects*. The Journal of Systems and Software 46, April 1999, S. 113-122.

[Dudler] Dudler, A. *Quantitative Modellierung von Verhaltensaspekten in Softwareprojekten*. Diplomarbeit Nr. 1819, Universität Stuttgart, Institut für Informatik, 2000.

[Hampp] Hampp, T. *Eine feingranulare SESAM-Variante*. Diplomarbeit Nr. 1931, Universität Stuttgart, Institut für Informatik, 2001.

[Hipfl] Hipfl, S. *Projekt AMEISE - Beschreibung des konzeptuellen Datenbankschemas*. Universität Klagenfurt, Institut für Informatik-Systeme, projektinterne Dokumentation, 2002.

[Jones] Jones, C. *Applied Software Measurement - Assuring Productivity and Quality*. McGraw-Hill, 2. Auflage, New York, 1996.

[Joos] Joos, M. *Konzeption und Realisierung eines Simulationsmodells für kleine Softwareprojekte*. Diplomarbeit Nr. 1543, Universität Stuttgart, Institut für Informatik, 1997.

[LUDEWIGa] LUDEWIG, J. *Woran scheitert der Projektleiter?*. INFORMATIK (Zeitschrift der Schweizerischen Informatik-Organisation), Nr. 5, Oktober 1999, S. 10-15.

[LUDEWIGb] LUDEWIG, J.; BASSLER, TH.; DEININGER M.; SCHNEIDER K.; SCHWILLE J. *SESAM - Simulating Software Projects*. Proceedings of the Software Engineering and Knowledge EngineeringConference (SEKE '92), Capri, Mai 1992, pp. 608 - 615.

[MANDL-STRIEGNITZa] MANDL-STRIEGNITZ, P.; LICHTER, H. *Defizite im Software-Projektmanagement - Erfahrungen aus einer industriellen Studie*. Informatik/Informatique Nr. 5, Oktober 1999, S. 4-9.

[MANDL-STRIEGNITZb] MANDL-STRIEGNITZ, P.; LICHTER, H. *Software-Projektmanagement in der Industrie - Erfahrungen und Analysen*. Bericht SL-2/96 des Software-Labors Stuttgart, 1996.

[MANDL-STRIEGNITZc] MANDL-STRIEGNITZ, P. *How to Successfully Use Software Project Simulation for Educating Software Project Managers*. in Proceedings of the 31st Frontiers in Education Conference (FIE 2001), Reno, Nevada, October 2001.

[MELCHISEDECH] MELCHISEDECH, R.; DEININGER, M.; DRAPPA A. *SESAM - A Software Engineering Education Tool Based on Graph Grammars*. Bulletin of the European Association for Theoretical Computer Science (EATCS), No. 58, February 1996, pp. 198 - 221.

[NOTTER] NOTTER, A. *Eine Untersuchung zur Wirksamkeit der Projektmanagement-Ausbildung am Simulator*. Diplomarbeit Nr. 1724, Universität Stuttgart, Institut für Informatik, 2003.

[NUSSER] NUSSER, M. *Projektunterstützung durch Verlaufsanalysen und Agenten*. Universität Klagenfurt, Institut für Informatik-Systeme, 2003.

[ROHRBACH] ROHRBACH, J. *Feinentwurf Teilsystem Dolmetscher*. internes Dokument, Abteilung Software Engineering, Institut für Informatik, Universität Stuttgart, 1998.

[ROYCE] ROYCE, W. *Managing the Development of Large Software Systems: Concepts and Techniques*. 1970 WESCON Technical Papers, Western Electronic Show and Convention, Los Angeles, A/1-1-A/1-9, 1970.

[SPIELGEL] SPIELGEL, A. *Konstruktion eines Dolmetschers*. Diplomarbeit Nr. 1304, Institut für Informatik, Universität Stuttgart, 1995.

[Vlissides] Vlissides, J. M.; Coplien, J. O.; Kerth N. L. *Pattern Languages of Program Design 2*. Addison-Wesley, 1996.