Diplomarbeit

# Konzeption und Realisierung eines werkzeuggestützten Reviewprozesses für wissenschaftliche Softwareprojekte

Stefan Neumann
Fachhochschule Köln,
Campus Gummersbach

DLR

Fachhochschule Köln
Cologne University of Applied Sciences
Campus Gummersbach - Institut für Informatik

# Konzeption und Realisierung eines werkzeuggestützten Reviewprozesses für wissenschaftliche Softwareprojekte

**Diplomarbeit**

im Studiengang Technische Informatik

Eingereicht von: Stefan Neumann
Mat. Nr.: 11037161

Organisation: Deutsches Zentrum für Luft- und Raumfahrt e.V.
Simulations- und Softwaretechnik

Erstprüfer: Prof. Dr. Mario Winter, FH Köln

Zweitprüfer: Andreas Schreiber, DLR

Datum: 22. April 2008

# Inhaltsverzeichnis

## Abbildungsverzeichnis

# Tabellenverzeichnis

## Abkürzungsverzeichnis

| | |
|---|---|
| API | Application Programming Interface |
| CSRS | Collaborative Software Review System |
| DIN | Deutsche Industrie Norm |
| DLR | Deutsches Zentrum für Luft- und Raumfahrt |
| GUI | Graphical User Interface |
| IDE | Integrated Development Invironment |
| IEC | International Electrotechnical Commission |
| IEEE | Institute of Electrical and Electronics Engineers |
| ISO | International Organization for Standardization |
| ISTQB | International Software Testing Qualification Board |
| NASA | National Aeronautics and Space Administration |
| OSGi | Open Service Gateway Initiative |
| SISTEC | Simulations- und Softwaretechnik |
| UML | Unified Modeling Language |
| URL | Uniform Resource Locator |
| WSDL | Web Services Description Language |

# 1 Einleitung

## 1.1 Einführung und Motivation

Im Jahr 1962 weicht die Raumsonde Mariner I vom vorausberechneten Kurs ab und muss schließlich über dem Atlantik gesprengt werden. Eine Untersuchung ergibt, dass eine Formel, die ursprünglich von Hand auf einen Zettel geschrieben worden war, falsch in die Software übertragen wurde. [URL:Mariner]

Dieses Beispiel zeigt die Bedeutung von Qualitätssicherung in der Softwareentwicklung. Ziel der Qualitätssicherung ist es, solche Fehler zu vermeiden. Eine Methode der Softwarequalitätssicherung, auf der das Hauptaugenmerk dieser Arbeit liegt, ist das Review. Bei einem Review wird ein Dokument von mehreren Personen gelesen und auf Anomalien hin untersucht. Hätten 1962 die NASA-Mitarbeiter die Formel auf dem Blatt Papier mit der Formel im Programmcode verglichen, hätte die Raumsonde Mariner I ihren Flug Richtung Venus planmäßig fortsetzen können. Gerade solche Fehler, wie die falsche Umsetzung einer Formel, können mit der richtigen Anwendung von Reviews vermieden werden.

## 1.2 Das Deutsche Zentrum für Luft- und Raumfahrt

Das Deutsche Zentrum für Luft und Raumfahrt (DLR) ist das Forschungszentrum der Bundesrepublik Deutschland für Luft- und Raumfahrt. Seine umfangreichen Forschungs- und Entwicklungsarbeiten in Luftfahrt, Raumfahrt, Energie und Verkehr sind in nationale und internationale Kooperationen eingebunden. Über die eigene Forschung hinaus ist das DLR als Raumfahrtagentur im Auftrag der Bundesregierung für die Planung und Umsetzung der deutschen Raumfahrtaktivitäten zuständig.

Die Mission des DLR umfasst

- die Erforschung von Erde und Universum,
- die Forschung für den Erhalt der Umwelt,
- die Entwicklung umweltverträglicher Technologien zur Steigerung der Mobilität sowie für Kommunikation und Sicherheit.

In 28 Instituten und Einrichtungen an den dreizehn Standorten Köln-Porz (Sitz des Vorstandes), Berlin-Adlershof, Bonn-Oberkassel, Braunschweig, Bremen, Göttingen, Hamburg, Lampoldshausen, Neustrelitz, Oberpfaffenhofen, Stuttgart, Trauen und Weilheim beschäftigt das DLR ca. 5.600 Mitarbeiterinnen und Mitarbeiter. Das DLR unterhält Büros in Brüssel, Paris und Washington, D.C.

Der Etat des DLR für die eigenen Forschungs- und Entwicklungsarbeiten sowie für Betriebsaufgaben beträgt ca. 450 Millionen Euro; davon sind etwa ein Drittel im Wettbewerb erworbene Drittmittel.

Das vom DLR verwaltete deutsche Raumfahrtbudget beträgt insgesamt ca. 846 Millionen Euro. [URL:DLR]

### 1.2.1 Die Einrichtung Simulations- und Softwaretechnik

Die Querschnittseinrichtung Simulations- und Softwaretechnik (SISTEC) hat ihre Kompetenz im Bereich Software Engineering. Die derzeitigen Themenschwerpunkte sind die komponentenbasierte Softwareentwicklung für verteilte Systeme, Software für eingebettete Systeme und die Software-Qualitätssicherung.

SISTEC bietet sein Know-how in Form von Dienstleistungen, gemeinsamen Projekten und Produkten sowohl DLR-Instituten als auch externen Partnern an.

Durch Projekte mit technologisch führenden Partnerinstitutionen und Mitarbeit in internationalen Foren erkennt SISTECC frühzeitig wichtige Trends in der Informationstechnologie und baut entsprechendes Know-how im DLR bedarfsorientiert auf.

SISTEC ist an den DLR-Standorten Köln-Porz und Braunschweig vertreten. Die thematische Abteilungsgliederung fällt dabei nicht mit der regionalen Aufteilung zusammen. Allerdings liegt der Schwerpunkt der Abteilung „Verteilte Systeme und Komponentensoftware“ in Köln-Porz, während die Abteilung „Software-Qualitätssicherung und eingebettete Systeme“ überwiegend in Braunschweig angesiedelt ist. [URL:SC]

## 1.3 Zielsetzung und Aufgabenstellung

Zurzeit wird in der Einrichtung Simulations- und Softwaretechnik im Deutschen Zentrum für Luft- und Raumfahrt das Tool Jupiter zur Unterstützung von Codereviews eingesetzt, welches jedoch nicht den existierenden Reviewprozess abbildet.

Zielsetzung dieser Arbeit ist daher die Konzeption und Realisierung eines werkzeuggestützten Reviewprozesses für die Einrichtung Simulations- und Softwaretechnik.

Dazu muss als Erstes der existierende Reviewprozess anhand dokumentierter Spezifikationen und anhand von Interviews mit Projektleitern und Entwicklern analysiert werden. Anschließend muss ein einheitlicher Reviewprozess entwickelt werden, der den verschiedenen Anforderungen an Reviews in der Einrichtung gerecht wird und sich zur Umsetzung in einer Software-Lösung eignet. Grundsätzlich muss der Reviewprozess die Phasen “Planung und Vorbereitung“ (inkl. Auswahl der Teilnehmer), „Durchführung“ und „Nachbereitung“ (hier insbesondere auch die Durchführung der vorgeschlagenen Codeänderungen) abbilden.

Besonders wichtig bei der Gestaltung der Software-Lösung ist die Handhabbarkeit. Das Reviewtool muss sich gut in den Ablauf eines Codereviews einfügen. Des Weiteren muss das Reviewtool so gestaltet sein, dass es den neu konzeptionierten Reviewprozess optimal abbildet.

Ein weiterer wichtiger Punkt ist die Integration des bestehenden Bugtrackingsystems Mantis. Anmerkungen zum Sourcecode müssen in das Bugtrackingsystem exportiert werden können.

Wünschenswert sind das automatische Generieren eines Reviewreports, eine „Reviewüberdeckungsmessung“, bei der gemessen wird, zu wie viel Prozent der Sourcecode schon geprüft wurde und die Vorschaltungen von Metriken zur Auswahl des Review-

prozesses. Ebenfalls wünschenswert ist ein allgemeiner Adapter zur Anbindung externer Tools.

## 1.4 Überblick

Diese Arbeit ist in drei logische Blöcke unterteilt, die sich aus der Zielsetzung und der Aufgabenstellungen ableiten.

Der erste Block setzt sich mit der Entwicklung eines Reviewprozesses für die Einrichtung Simulations- und Softwaretechnik des DLR auseinander (Kapitel 2 und 3). Der zweite Block beschäftigt sich mit der Umsetzung dieses Reviewprozesses in eine Softwarelösung (Kapitel 3 bis 9). Der abschließende dritte Block liefert einen kritischen Rückblick auf die Arbeit und einen Ausblick auf zukünftige Erweiterungen der Softwarelösung (Kapitel 10 bis 11).

Dabei wird bei den ersten beiden Blöcken zuerst eine kurze Einführung in das Thema gegebenen, danach der IST-Zustand dokumentiert und anschließend die Lösung der Problemstellung vorgestellt.

So gibt Kapitel 2 eine Einführung in Reviews und definiert Begriffe, die anschließend die Grundlage der weiteren Analysen sind. Darauf aufbauend wird dann in Kapitel 3 der Reviewprozess im DLR analysiert und ein neuer Reviewprozess definiert.

Kapitel 4 gibt eine kurze Einführung in werkzeuggestützte Codereviews und stellt zwei kommerzielle Reviewtools exemplarisch vor. Kapitel 5 beschäftigt sich dann mit der Eclipse-Plattform, in die das bestehende Tool Jupiter integriert ist und in die auch das zukünftige Tool integriert werden soll. Kapitel 6 betrachtet das Tool Jupiter und gibt eine abschließende Bewertung zu dessen Nutzbarkeit ab. In Kapitel 7 wird anschließend die Anforderungsermittlung des neuen Reviewtools durchgeführt. Kapitel 8 und 9 dokumentieren dann Konzeption sowie Feinentwurf.

Kapitel 10 liefert ein Resümee und Kapitel 11 einen Ausblick auf kommende Aufgaben.

# 2 Reviews

In diesem Kapitel wird eine Übersicht über Reviews im Allgemeinen und Codereviews im Speziellen gegeben. Es wird beschrieben, was ein Review ist und welche Nutzen es hat. Des Weiteren werden im Kapitel 2.3 grundlegende Begriffe definiert. Im Kapitel 2.4 wird abschließend beschrieben, wie man das Potential von Reviews voll ausschöpft.

## 2.1 Einführung

Ein Review ist eine manuelle Prüfmethode von Dokumenten unterschiedlichster Art. Dies können Anforderungsspezifikationen, Dokumentationen oder auch Programmcode sein. Im Deutschen ist die wohl treffendste Bezeichnung „Durchsicht“. D. h., ein Dokument wird von mehreren Personen durchgelesen und auf Anomalien hin geprüft. In einem Review werden Anomalien nur aufgedeckt, sie werden dort nicht behoben. Die Korrekturarbeiten sind in der Regel die Aufgabe des Autors des Dokuments. Ein Review sollte in jedem Abschnitt des V-Modells durchgeführt werden, sodass alle Dokumente in einem Entwicklungsprozess durch ein Review geprüft worden sind. Codereviews machen also nur Sinn, wenn auch die vorhergehenden Dokumente mit Reviews geprüft worden sind. Abbildung 2.1 zeigt den Einsatz von Reviews im V-Modell. Neben der Prüfung der Entwurfsdokumente sollten auch die Testfälle einem Review unterzogen werden.

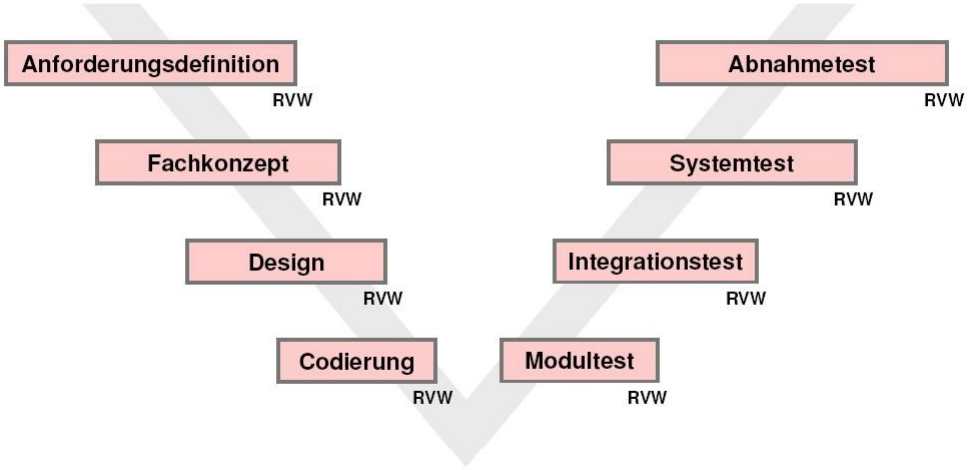



**Abbildung 2.1: Reviews im V-Modell**

Ziel von Codereviews im Allgemeinen ist die Sicherstellung der Softwarequalität. Softwarequalität wird in ISO 9126 [ISO9126] in „Gebrauchsqualität“ und „äußere und innere Qualität“ unterteilt.

Zu Gebrauchsqualität zählen:

- **Effektivität**: Aufgabenerfüllung innerhalb der Genauigkeit- und Vollständigkeitsgrenzen
- **Produktivität**: Aufgabenerfüllung innerhalb der Aufwandsgrenzen für Benutzer (Zeit, Kosten, ...)

- **Sicherheit**: Aufgabenerfüllung innerhalb der Risikogrenzen (Leben und Gesundheit, Business, ...)
- **Zufriedenheit**: subjektive Zufriedenheit der Benutzer innerhalb des Nutzungskontextes

Zu äußerer und innerer Qualität zählen:

- **Funktionalität**: Richtigkeit, Angemessenheit, Interoperabilität, (Daten-)Sicherheit, Ordnungsmäßigkeit
- **Zuverlässigkeit**: Reife, Fehlertoleranz, Wiederherstellbarkeit
- **Benutzbarkeit**: Verständlichkeit, Erlernbarkeit, Bedienbarkeit, Attraktivität
- **Effizienz**: Zeitverhalten, Verbrauchsverhalten
- **Änderbarkeit**: Analysierbarkeit, Modifizierbarkeit, Stabilität, Testbarkeit
- **Portierbarkeit**: Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit

Diese Qualitätskriterien sollten bei einem Review eines Dokumentes immer als Grundlage dienen. Ebenso wie bei den einzelnen Teststufen des V-Modells.

## 2.2 Nutzen von Reviews/Codereviews

Reviews können bereits lange vor einer dynamischen Testdurchführung durchgeführt werden. Deshalb können mit Reviews Fehlerzustände frühzeitig aufgedeckt werden. Diese Fehlerzustände, die in den frühen Phasen des Softwarelebenszyklus entdeckt werden, sind häufig bedeutend kostengünstiger zu beheben, also solche, die erst während der Testdurchführung gefunden werden. [ISTQB05]

Durch die frühe Aufdeckung von Fehlern wird die Entwicklungsdauer reduziert und die Softwareentwicklungproduktivität verbessert. Des Weiteren werden die Testkosten und die Testdauer reduziert.

Neben dem reinen Aufdecken von Fehlern im Sourcecode können Reviews im Gegensatz zu dynamischen Tests Auslassungen wie z. B. fehlende Funktionen aufdecken, die durch einen dynamischen Test vermutlich nicht gefunden worden wären. Typische Beispiele für Fehlerzustände, die effektiver und effizienter durch Reviews als durch dynamische Tests zu finden sind, sind: Abweichungen von Standards, Fehler in Anforderungen, Fehler im Design, unzureichende Wartbarkeit und falsche Schnittstellenspezifikationen. [ISTQB05]

Neben dem Aufdecken von Fehlerzuständen tragen Codereviews auch zum besseren Verständnis des Sourcecodes bei allen beteiligten Personen bei.

## 2.3 Definitionen

Die folgenden Definitionen stammen aus dem Certified Tester Foundation Level Lehrplan des International Software Testing Qualification Board (ISTQB) [ISTQB05] und sollen als Grundlage für die spätere Analyse des Reviewprozesses dienen. Diese Definitionen beruhen ihrerseits auf dem IEEE Std. 1028 – 1997, der Softwarereviews noch etwas detaillierter betrachtet.

### 2.3.1 Phasen eines formalen Reviews

Die folgenden Phasen beziehen sich auf ein typisches formales Review. Je nach Art des Reviews fehlen einige dieser Phasen.

- **Planung**: Auswahl der beteiligten Personen, Besetzung der Rollen; Festsetzung der Eingangsbedingung und Endebedingung bei mehr formalen Reviewarten (z. B. Inspektion) und Festlegung der zu betrachtenden Dokumententeile.
- **Kick-off**: Verteilung der Dokumente; Erläuterung der Ziele, des Prozesses und der Dokumente den Teilnehmern gegenüber; und Prüfung der Eingangsbedingungen (bei mehr formalen Reviewarten).
- **Individuelle Vorbereitung**: Tätigkeit, die von jedem einzelnen Teilnehmer für sich allein vor der Reviewsitzung durchgeführt wird, Notierung von Fehlern, Fragen und Kommentaren.
- **Reviewsitzung**: Diskussion oder Protokollierung, mit dokumentierten Ergebnissen oder Protokollen (bei mehr formalen Reviewarten). Die Teilnehmer der Sitzung können die Fehler notieren, Empfehlungen zum Umgang mit ihnen machen oder Entscheidungen über die Fehler treffen.
- **Überarbeitung**: Behebung der gefundenen Fehlerzustände typischerweise durch den Autor.
- **Nachbereitung**: Prüfung, dass Fehlerzustände zugeordnet wurden, Sammeln von Metriken und Prüfung von Testendekriterien (bei mehr formalen Reviewarten).

### 2.3.2 Rollen und Verantwortlichkeiten

Die Rollen und Verantwortlichkeiten beziehen sich wie auch bereits die Phasen eines Reviews auf ein typisches formales Review. Auch hier entfallen je nach Art des Reviews einige der Rollen.

- **Manager**: entscheidet über die Durchführung von Reviews, stellt Zeit im Projektplan zur Verfügung und überprüft, ob die Reviewziele erfüllt sind.
- **Moderator**: die Person, die das Review eines Dokuments, bzw. von einigen zusammengehörenden Dokumenten leitet, einschließlich der Reviewplanung, der Leitung der Sitzung und der Nachbereitung der Sitzung. Falls nötig, kann der Moderator zwischen den verschiedenen Standpunkten vermitteln. Er ist häufig die Person, von der der Erfolg des Reviews abhängt.
- **Autor**: der Verfasser oder die Person, die für das/die zu überprüfende/n Dokument/e hauptverantwortlich ist.
- **Gutachter**: Personen mit einem spezifischen technischen oder fachlichen Hintergrund (auch Prüfer oder Inspektor genannt), die nach der nötigen Vorbereitung im Prüfobjekt Befunde identifizieren und beschreiben (z. B. Fehler). Gutachter sollten so gewählt werden, dass verschiedene Sichten und Rollen im Reviewprozess vertreten sind und sie an allen Reviewsitzungen teilnehmen können.

- **Protokollant**: Dokumentiert alle Ergebnisse, Probleme und offene Punkte, die im Verlauf der Sitzung identifiziert werden.

### 2.3.3 Reviewarten

Die Auswahl der Reviewarten ist abhängig von der Kritikalität des zu überprüfenden Dokuments. Je kritischer ein Fehler in einem bestimmten Dokument sein kann, desto formaler sollte das Review sein. Die folgende Auflistung reicht von eher informell bis hin zu sehr formell.

**Informelles Review**

Hauptmerkmale:

- kein formaler Prozess
- kann integriert im Pair Programming sein oder ein technischer Leiter unterzieht Entwurf und Quelltext einem Review
- wahlweise kann dokumentiert werden
- Nutzen kann abhängig von Gutachtern variieren
- Hauptzweck: kostengünstiger Weg um hohen Nutzen zu erhalten

**Walkthrough**

Hauptmerkmale:

- Sitzung geleitet durch den Autor
- Szenarien, Probeläufe, im Kreis gleichgestellter Mitarbeiter
- Open-End Sitzung
- wahlweise der Sitzung vorausgehende Vorbereitung der Gutachter, Reviewbericht, Liste der Befunde und Protokollant (der aber nicht der Autor ist)
- kann in der Praxis von informell bis sehr formell variieren
- Hauptzweck: Lernen, Verständnis erzielen, Fehler finden

**Technisches Review**

Hauptmerkmale:

- Dokumentierter, definierter Fehlerfindungsprozess, der gleichgestellte Mitarbeiter und technische Experten einschließt
- Kann als Peer Review ohne Teilnahme des Managements ausgeführt werden
- Idealerweise durch einen geschulten Moderator geleitet (nicht der Autor)
- Vorbereitung der Sitzung
- Wahlweise: Nutzung von Checklisten, Reviewberichten, Liste der Befunde und Managementteilnahme
- kann in der Praxis von informell bis sehr formell variieren

- Hauptzweck: Diskussion, Entscheidungen treffen, Alternativen bewerten, Fehler finden, technische Probleme lösen und prüfen, ob Übereinstimmung mit Spezifikationen und Standards existiert

**Inspektion**

Hauptmerkmale:

- Geleitet durch einen geschulten Moderator (nicht der Autor)
- Gewöhnlich Prüfung durch gleichgestellte Mitarbeiter
- definierte Rollen
- beinhaltet Metriken
- formaler Prozess basierend auf Regeln und Checklisten und Eingangsbedingungen und Endebedingungen
- Vorbereitung vor der Sitzung
- Inspektionsbericht, Liste der Befunde
- Formaler Prozess für Folgeaktivitäten
- Wahlweise: Prozessverbesserung und Vorleser
- Hauptzweck: Fehler finden

## 2.4 Erfolgsfaktoren von Reviews

Um ein Review erfolgreich zu gestalten und einen größtmöglichen Nutzen von ihm zu haben, ist es wichtig, dass gewisse Regeln und Vorgaben eingehalten werden.

### 2.4.1 Ziele

Ein Review sollte immer ein klares Ziel haben. Dieses Ziel sollte je nach Art des Reviews vom Manager, Moderator oder Autor vorgegeben werden. Das Erreichen des Zieles sollte am Ende der Reviewsitzung überprüft werden. Um diese Ziele zu erreichen, sollten die geeigneten Personen ausgewählt werden. Ideal wäre es Spezialisten für jeden Teilbereich des zu untersuchenden Sourcecodes zu haben, sodass ein breites Spektrum von möglichen Fehlern abgedeckt wird.

### 2.4.2 Kommunikation

Gefundene Fehler sollten objektiv kommuniziert werden, es darf nur das Dokument, niemals der Autor kritisiert werden. Wird das Review zu einer negativen Erfahrung für den Autor, so hat das Review nicht den erwünschten Effekt des Lernens und der Prozessverbesserung erfüllt.

### 2.4.3 Management

Ebenfalls kann das Management einen guten Reviewprozess unterstützen, indem bspw. eine angemessene Zeit für Reviewaktivitäten im Projektplan berücksichtigt wird. Des Weiteren sollten Schulungen in Reviewtechniken stattfinden.

### 2.4.4 Lernen und Prozessverbesserung

Die Betonung des Reviews sollte auf Lernen und Prozessverbesserung liegen. Dazu gehört natürlich auch, dass die Reviews gut dokumentiert sind, sodass man im Nachgang des Reviews Verbesserungsvorschläge in den Reviewprozess einarbeiten kann.

# 3 Der Codereviewprozess in der Einrichtung SISTEC

Dieses Kapitel setzt sich mit dem Codereviewprozess in der Einrichtung Simulations- und Softwaretechnik im DLR auseinander. Dazu wird als Erstes der IST-Zustand analysiert. Darauf aufbauend werden drei Reviewprozesse von sehr formell bis informell definiert, um den verschiedenen Ansprüchen an ein Codereview gerecht zu werden.

## 3.1 Der IST-Zustand

### 3.1.1 Codereviews im Softwareentwicklungsprozess

Codereviews sind ein fester Bestandteil des Softwareentwicklungsprozesses der Einrichtung SISTEC. Sie bilden den Abschluss einer jeden neuen Teilimplementierung. Sie finden immer dann statt, wenn eine neue logische Einheit fertiggestellt wurde.

Abbildung 3.1 zeigt den Softwareentwicklungsprozess in der Einrichtung SISTEC. Dabei wird jedes neue Feature aus dem Designdokument in das Bugtrackingsystem eingepflegt. Nur Features, die auch im Bugtrackingsystem eingetragen sind, werden auch umgesetzt.

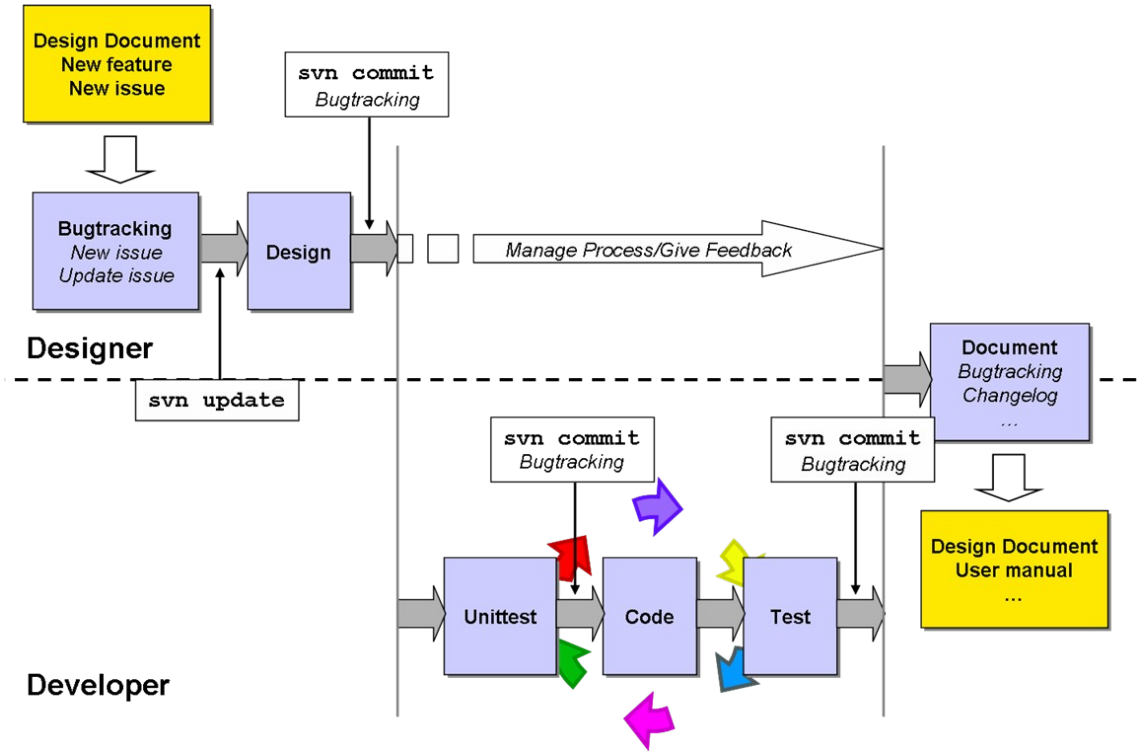



**Abbildung 3.1: SISTEC: Softwareentwicklungprozess [SESIS07]**

Abbildung 3.2 verdeutlicht noch einmal in welchem Entwicklungsumfeld Codereviews durchgeführt werden. Codereviews werden in einem komplexen Umfeld von qualitätssichernden Maßnahmen durchgeführt.

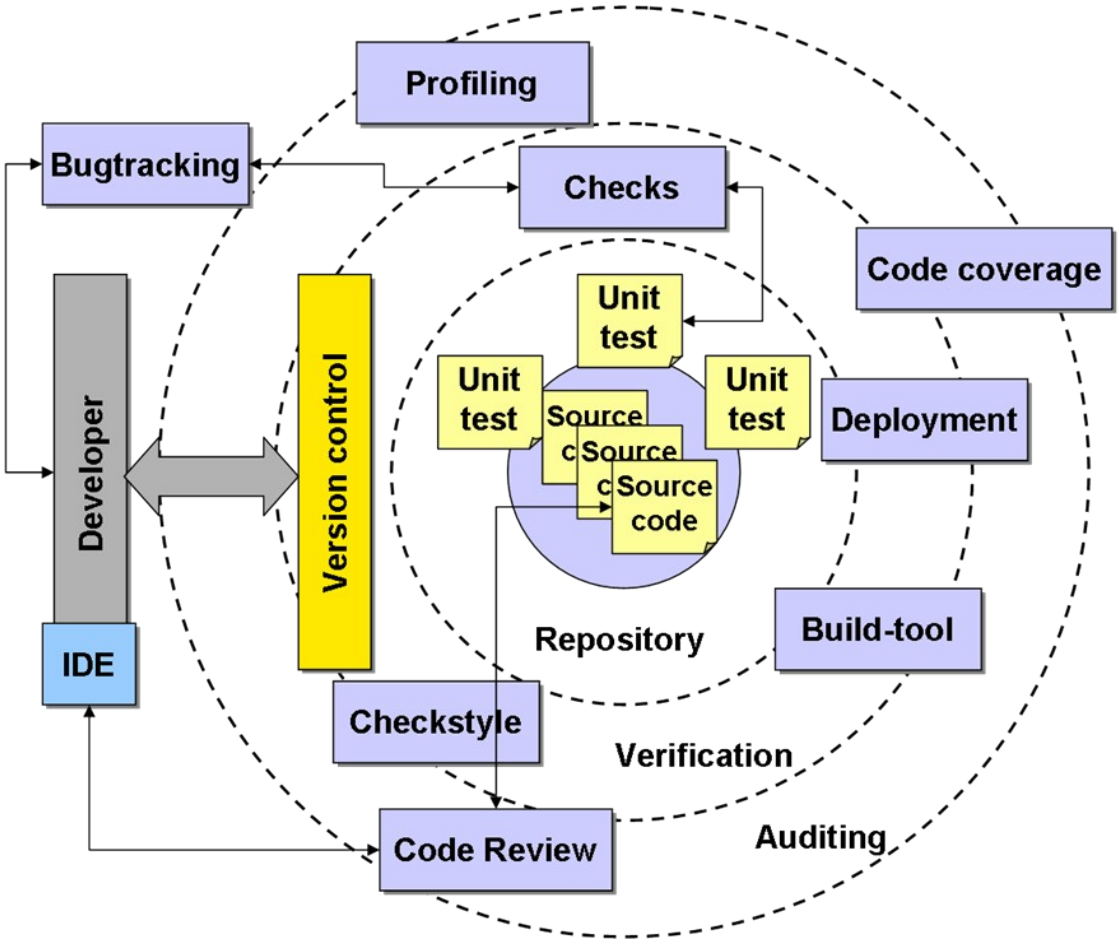



**Abbildung 3.2: SISTEC: Codereviews im Softwareentwicklungprozess [SESIS07]**

### 3.1.2 Teilnahme an einem Codereview

Die Teilnahme an einem Codereview zeigt, dass Codereviews in der Einrichtung am ehesten den in Kapitel 2 vorgestellten Walkthroughs entsprechen. Dabei muss jedoch nicht zwingend der Autor die Person sein, die durch den Sourcecode leitet. Das Codereview besteht aus einer Reviewsitzung, in der der entsprechende Sourcecode durchgearbeitet wird. Dabei wird der Sourcecode auf die korrekte Umsetzung des Designs, auf Effektivität, korrekte Kommentierung usw. überprüft. Ein ebenfalls wichtiger Nutzen neben der Fehlerfindung ist das bessere Verständnis des Sourcecodes bei allen Beteiligten. Dabei werden im Codereview Fragen der Gutachter vom Autor beantwortet, die dann bspw. verdeutlichen, dass der Sourcecode nicht ausreichend kommentiert ist.

Insgesamt werden in den Codereviews Klassen, Methode für Methode, durchgearbeitet und Details, die den Reviewern auffallen diskutiert. Dabei kommt es nicht selten vor, dass der Autor bei der Präsentation seines Sourcecodes Anomalien selbst entdeckt.

### 3.1.3 Spezifikationen

Die einzige Spezifikation, die zum Thema Reviews vorliegt, ist der IEEE Std. 1028-1997. Dieser hat jedoch keinen praktischen Nutzen für die Einrichtung, da der IEEE Std.

1028-1997 zwar verschiedene Reviewarten formal definiert, jedoch keine genauen Hinweise auf deren Durchführung gibt. Es fehlt also eine Prozessdefinition, die den Softwareentwicklern als Handbuch und Richtlinie dient. Codereviews werden also hauptsächlich nach der persönlichen Erfahrung der Reviewteilnehmer gestaltet.

## 3.2 Erarbeitung eines neuen Codereviewprozesses

Um den verschiedenen Anforderungen an ein Codereview gerecht zu werden und um in Zukunft definierte Reviewprozesse nutzen zu können, werden an dieser Stelle drei verschiedene Reviewprozesse definiert. Sie reichen von sehr formell bis informell und können je nach Kritikalität des zu überprüfenden Sourcecodes angewandt werden.

### 3.2.1 Formelles Codereview mit individueller Prüfung

Das formelle Codereview mit individueller Prüfung entspricht zu großen Teilen der Inspektion, wie sie in Kapitel 2.3.3 Reviewarten beschrieben wird. Dieser Reviewprozess dient vor allem dazu, Fehler zu finden. Er ist daher sehr formell und strukturiert und arbeitet mit Hilfsmitteln wie Checklisten, um den Sourcecode gezielt auf Anomalien zu untersuchen. Dazu wird der Sourcecode im Vorfeld der eigentlich Reviewsitzung von den einzelnen Reviewern untersucht. Dabei ist es sinnvoll, dass jeder Reviewer den Sourcecode unter einem anderen Gesichtspunkt betrachtet, damit ein breites Spektrum an möglichen Fehlern abgedeckt ist.

Abbildung 3.3 zeigt den Ablauf des formellen Codereviews mit individueller Prüfung. Das Codereview unterteilt sich hierbei in die Phasen „Vorbereitung“, „Kick-off“, „individuelle Prüfung“, „Reviewsitzung“ und „Nachbereitung“.

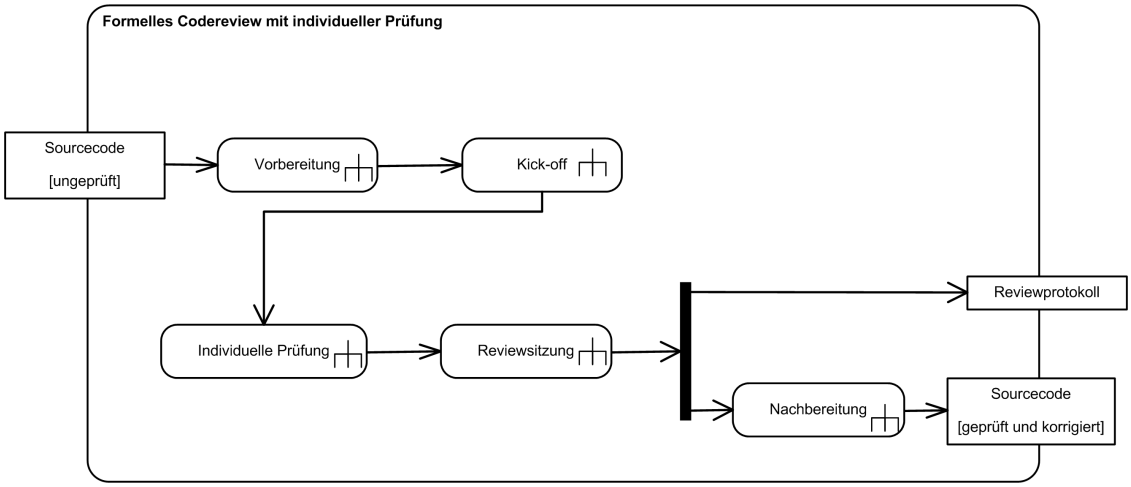



**Abbildung 3.3: Ablauf des formellen Codereviews mit individueller Prüfung**

Die Vorbereitung ist Aufgabe des Moderators oder des Projektleiters. Dazu lässt er sich den zu prüfenden Sourcecode vom Autor zur Verfügung stellen und stellt ein Reviewteam zusammen. Danach legt er einen Termin für ein erstes Meeting, dem Kick-off, fest.

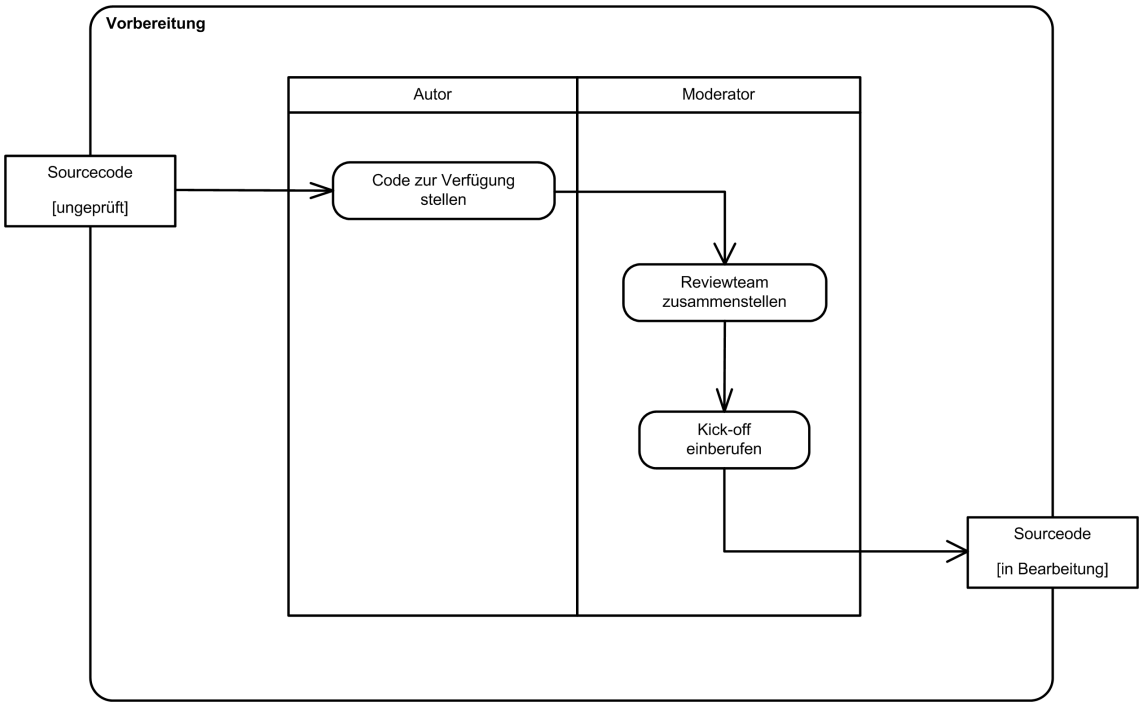



**Abbildung 3.4: Vorbereitung des formellen Codereviews mit individueller Prüfung**

Beim Kick-off wird ein erster Überblick über das bevorstehende Codereview gegeben. Dazu legt der Moderator einen verbindlichen Zeitplan für das Codereview fest und stellt den Gutachtern die nötigen Dokumente zur Verfügung. Der Autor stellt nun den zu prüfenden Sourcecode vor und gibt den Gutachtern die Möglichkeit fragen zu stellen, bevor diese in die individuelle Prüfung übergehen. Die Leitung des Kick-offs obliegt dem Moderator.

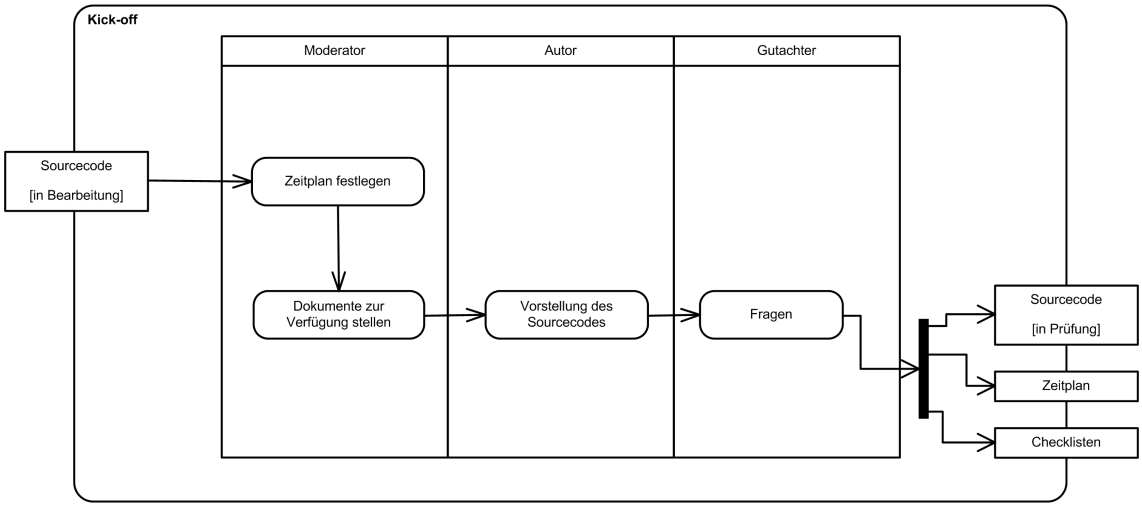



**Abbildung 3.5: Kick-off des formellen Reviews mit individueller Prüfung**

Nach Abschluss des Kick-offs wird der Sourcecode mithilfe der im Kick-off ausgebenden Checklisten von den einzelnen Gutachtern untersucht. Jeder Gutachter erstellt seine individuelle Liste mit Anomalien.

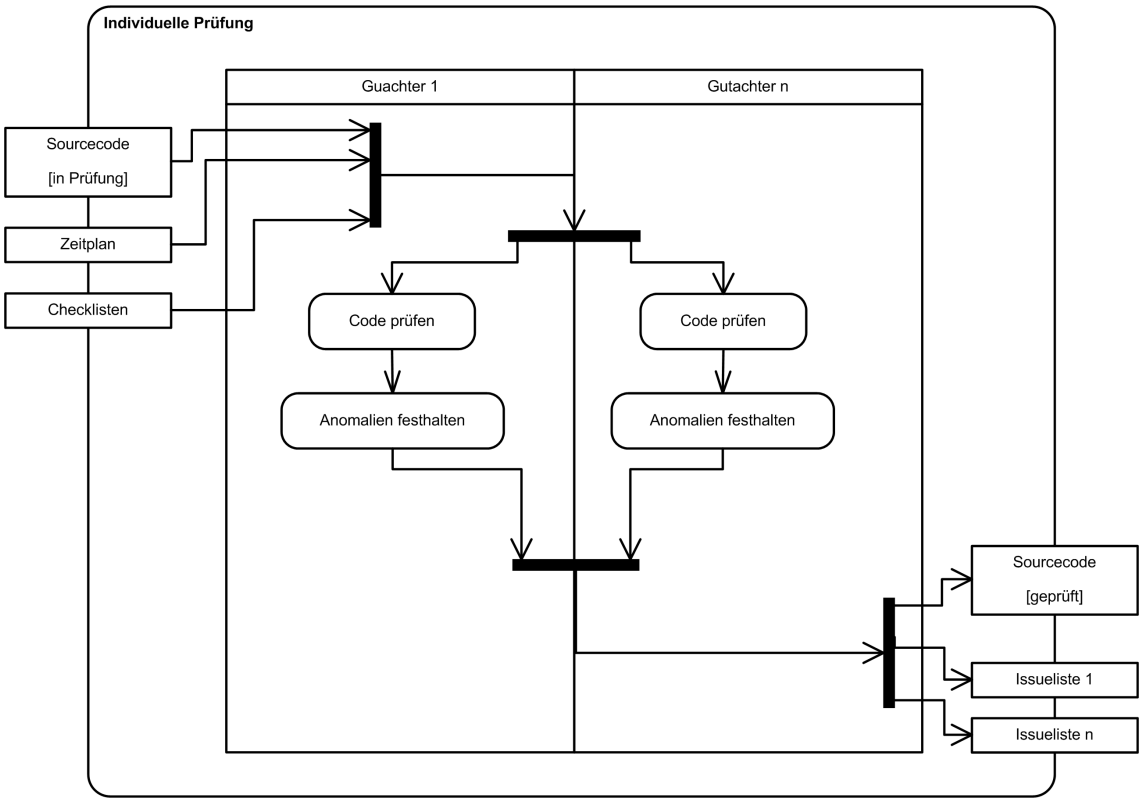



**Abbildung 3.6: Ind. Prüfung des formellen Codereviews mit individueller Prüfung**

Nach Abschluss der individuellen Prüfung werden die einzelnen Issuelisten in der Reviewsitzung zusammengetragen und dort diskutiert. Auch die Leitung dieser Sitzung liegt in der Verantwortung des Moderators. Die einzelnen Gutachter präsentieren ihre individuellen Issuelisten und diskutieren die Anomalien mit dem Autor. Verbesserungsvorschläge können ebenfalls angebracht werden. Lange Diskussionen über Lösungsmöglichkeiten sollten jedoch vermieden werden. Des Weiteren sollten so genannte „lessons learned“ gesammelt werden. Die „lessons learned“ beinhalten Anmerkungen zum eigentlichen Reviewprozess (was ist gut, was ist weniger gut gelaufen, was kann verbessert werden) und dienen der Prozessverbesserung. Zum Ende der Reviewsitzung teilt der Moderator die Verantwortlichkeiten zur Beseitigung der Anomalien zu. Im Normalfall sind die Verantwortlichen der oder die Autor/en. Es können aber auch andere Mitglieder des Entwicklungsteams sein. Die komplette Reviewsitzung wird vom Protokollant protokolliert.

Er erstellt den abschließenden Reviewreport, der die folgenden Daten enthält:

- Datum und Dauer des Reviews
- Reviewsitzungsteilnehmer
- Issuelisten
- Verantwortlichkeiten der Anomalienbeseitigung
- Kenngrößen (Anzahl Gutachter, Anzahl gefundener Fehler, Fehlerschweren)

Am Ende der Sitzung stehen dem Autor (oder den anderen Verantwortlichen der Anomalienbeseitigung) eine ToDo-Liste und der Reviewreport zur Beseitigung der Anomalien zur Verfügung.

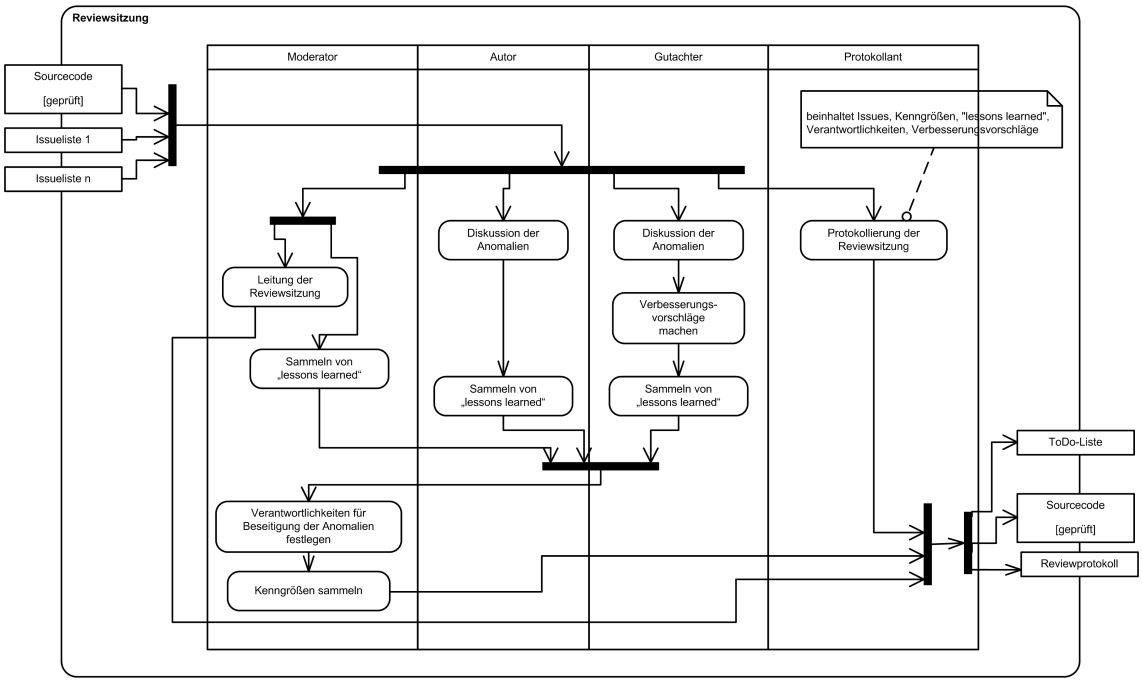



**Abbildung 3.7: Reviewsitzung des formellen Reviews mit individueller Prüfung**

Die letzte Phase eines jeden Reviewprozesses ist die „Nachbereitung“, in der der Autor die Anomalien beseitigt. Zusätzlich könnte die korrekte Beseitigung der Anomalien in einer weiteren Reviewsitzung überprüft werden.

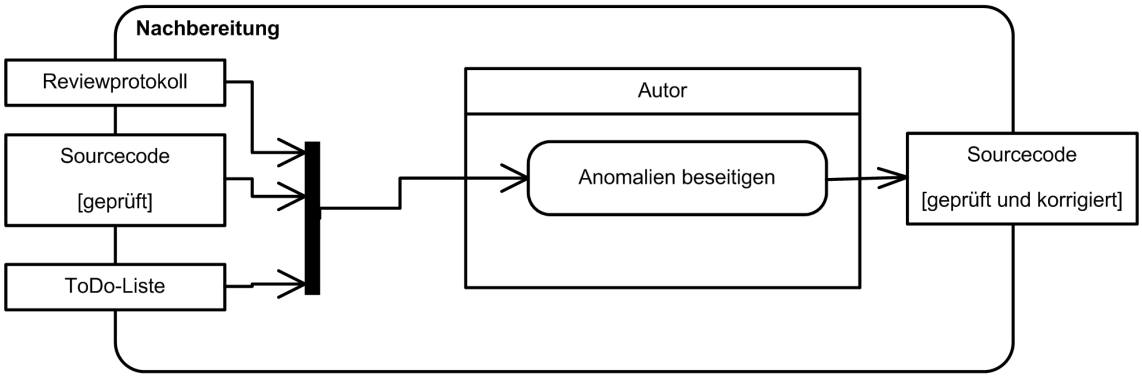



**Abbildung 3.8: Nachbereitung des formellen Reviews mit individueller Prüfung**

Um die Zusammenhänge der einzelnen Rollen und Dokumente zu verdeutlichen, erfolgt an dieser Stelle eine Darstellung als statisches Modell.

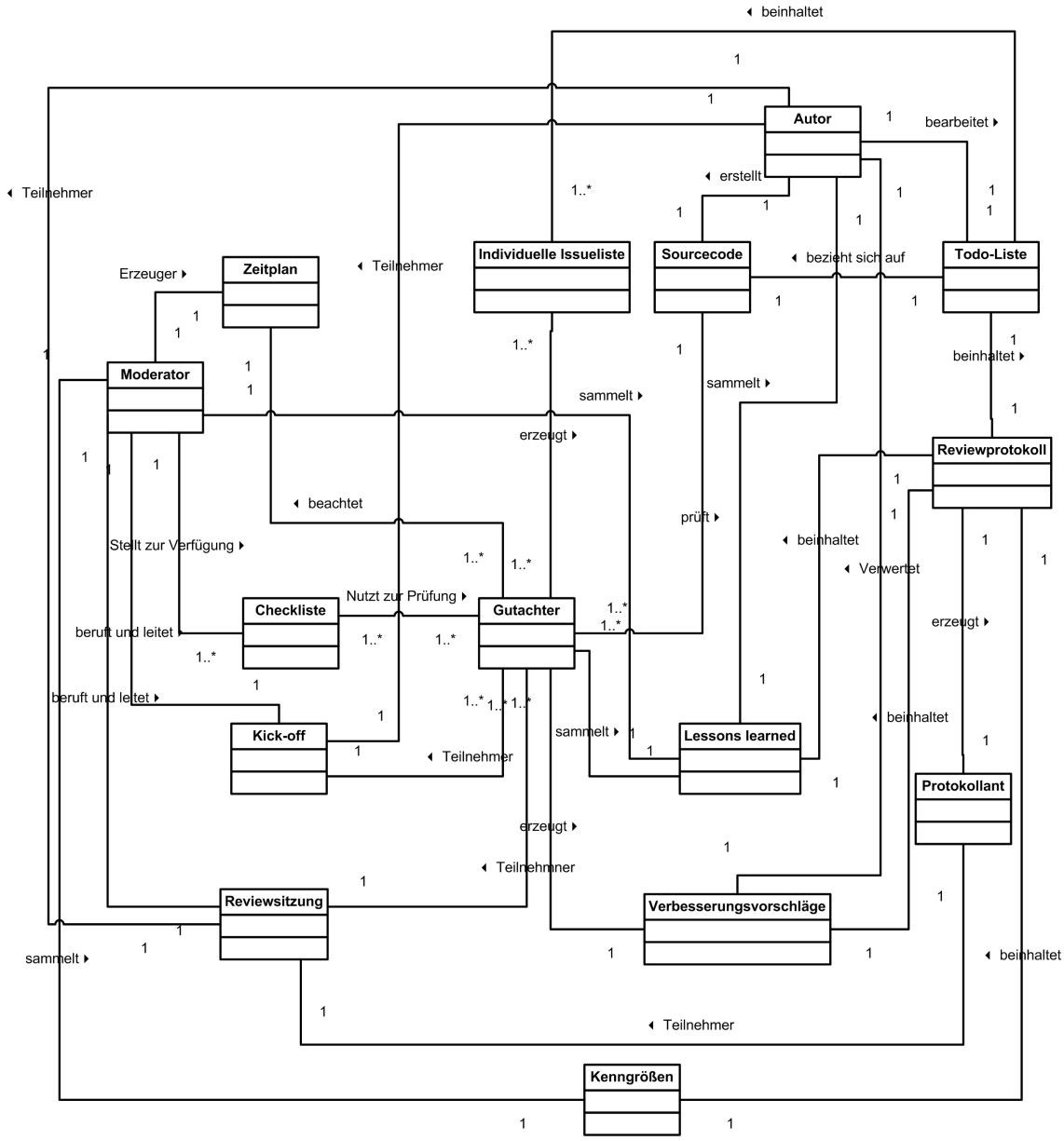



**Abbildung 3.9: Statisches Modell des formellen Reviews mit individueller Prüfung**

### 3.2.2 Formelles Codereview ohne individuelle Prüfung

Das formelle Codereview ohne individuelle Prüfung entspricht prinzipiell dem formellen Codereview mit individueller Prüfung, mit dem Unterschied, dass die Prüfung des Sourcecodes ausschließlich in der gemeinsamen Reviewsitzung stattfindet. Von der Zielsetzung und der Formalität unterschieden sich die beiden Reviewprozesse nicht.

Abbildung 3.10 zeigt den Ablauf des formellen Reviews ohne individuelle Prüfung. Hier entfallen nun das Kick-off Meeting und die individuelle Prüfung.

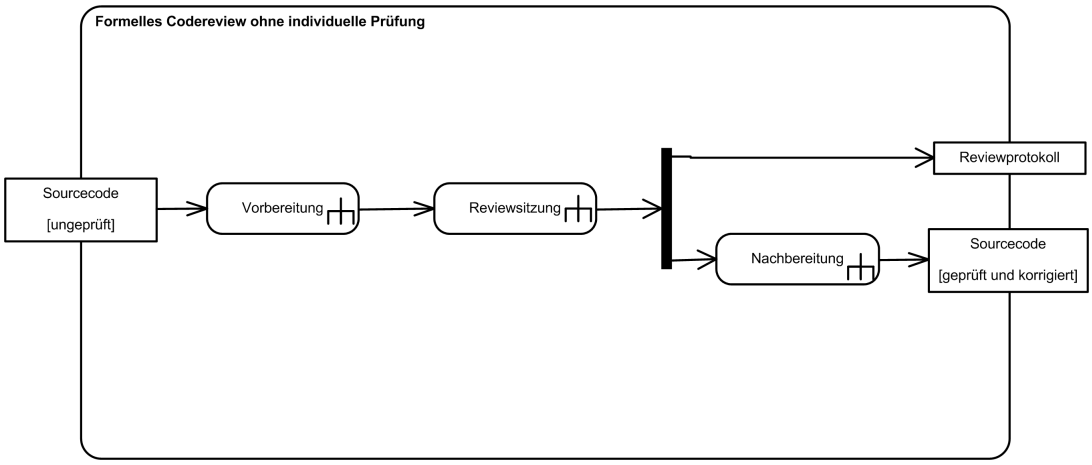



**Abbildung 3.10: Ablauf des formellen Reviews ohne individuelle Prüfung**

Die Vorbereitung entspricht vollkommen der Vorbereitung des formellen Reviews mit individueller Prüfung.

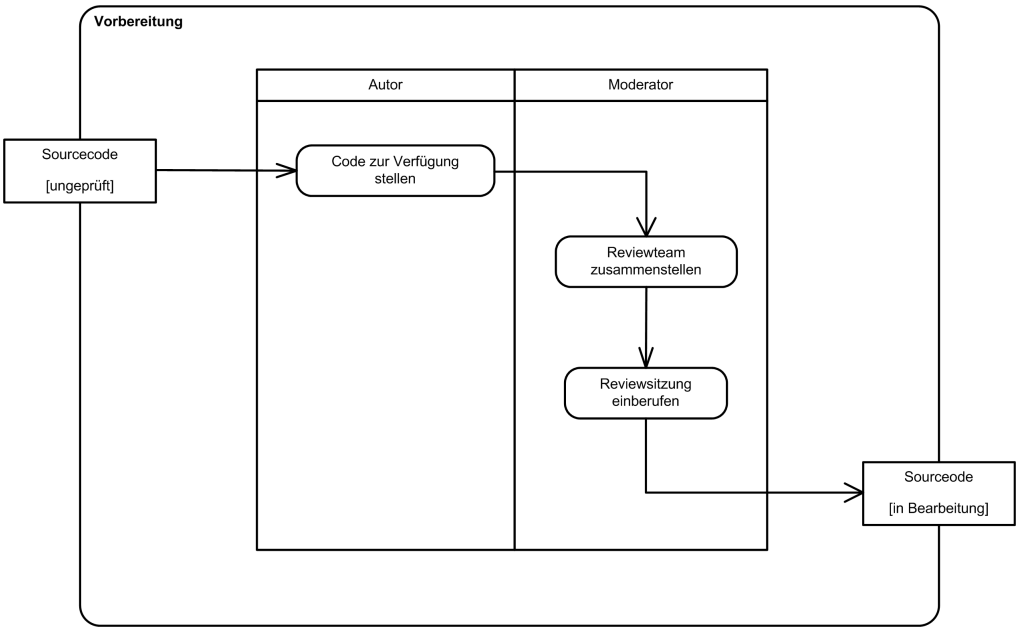



**Abbildung 3.11: Vorbereitung des formellen Reviews ohne individuelle Prüfung**

Die Reviewsitzung unterscheidet sich hingegen stark vom formellen Review mit individueller Prüfung, da ohne eine individuelle Prüfung des Sourcecodes, der Sourcecode während der eigentlichen Reviewsitzung geprüft wird.

Die Reviewsitzung wird vom Moderator geleitet. Er stellt den Gutachtern alle wichtigen Dokumente zur Verfügung. Zu Beginn stellt der Autor den zu überprüfenden Sourcecode kurz vor. Danach arbeitet er den Sourcecode strukturiert durch. Während dessen können die Gutachter Fragen zum Sourcecode stellen und Anmerkungen und Verbesserungsvorschläge machen. Zum Ende der Reviewsitzung ist dann der komplette zu überprüfende Sourcecode durchgearbeitet. Zum Abschluss der Reviewsitzung werden auch bei diesem Reviewprozess allgemeine Anmerkungen zum Ablauf des Codereviews gesammelt, die „lessons learned“. Der Moderator weist dann im Anschluss

noch die Verantwortlichkeiten zur Anomalienbeseitigung zu. Wie auch im formellen Codereview mit individueller Prüfung erstellt der Protokollant ein Protokoll der gesamten Reviewsitzung.

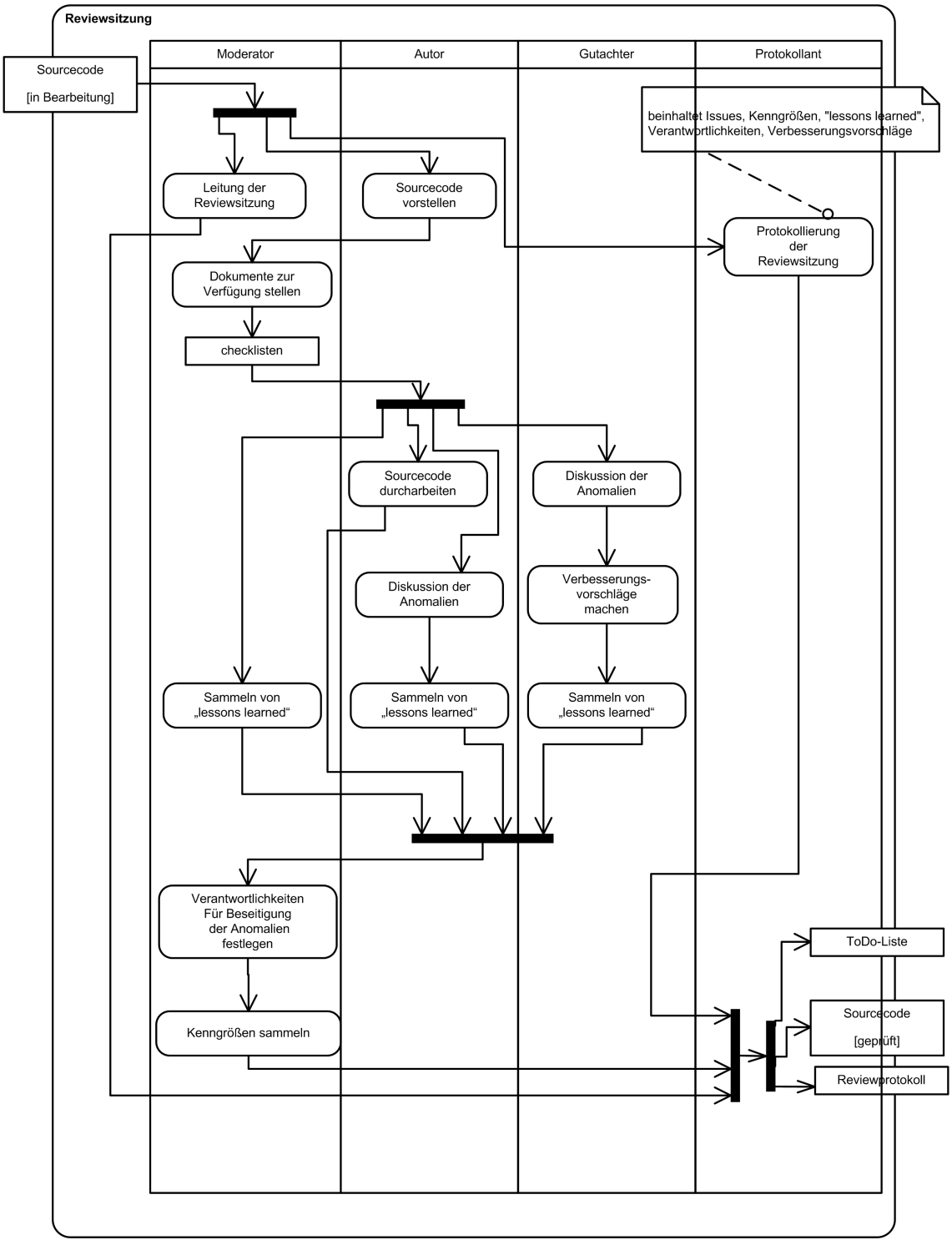



**Abbildung 3.12: Reviewsitzung des formellen Codereviews ohne ind. Prüfung**

Die letzte Phase des formellen Codereviews ist erneut die „Nachbereitung“. Sie ist wieder identisch mit der Nachbereitung des formellen Codereviews mit individueller Prüfung.

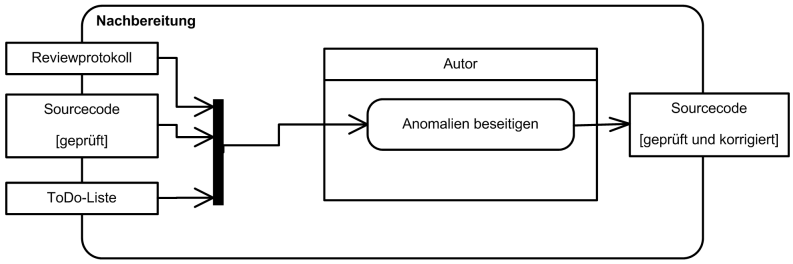



**Abbildung 3.13: Nachbereitung des formellen Codereviews ohne ind. Prüfung**

Um die Zusammenhänge der einzelnen Rollen und Dokumente zu verdeutlichen, erfolgt auch an dieser Stelle eine Darstellung als statisches Modell.

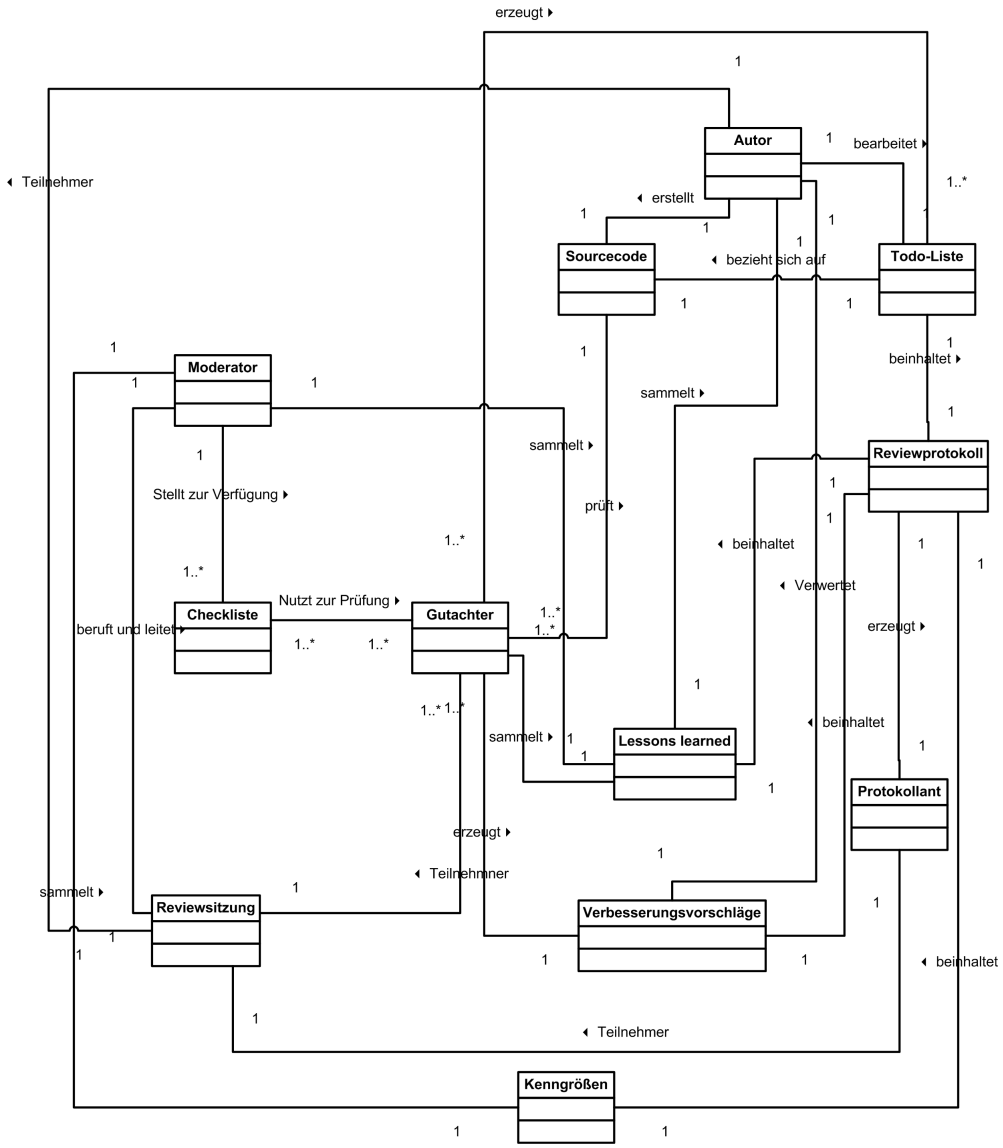



**Abbildung 3.14: Statisches Modell des formellen Codereviews ohne ind. Prüfung**

### 3.2.3 Informelles Codereview

Bei einem informellen Codereview führt der Autor durch seinen Sourcecode. Ziel ist besseres Verständnis für den Sourcecode bei den Teilnehmern des Codereviews zu erzielen. Des Weiteren sollen auch Anomalien im Sourcecode aufgedeckt werden und Verbesserungsvorschläge gemacht werden. Abbildung 3.15 zeigt die beteiligten Rollen und Dokumente als UML-Klassendiagramm.

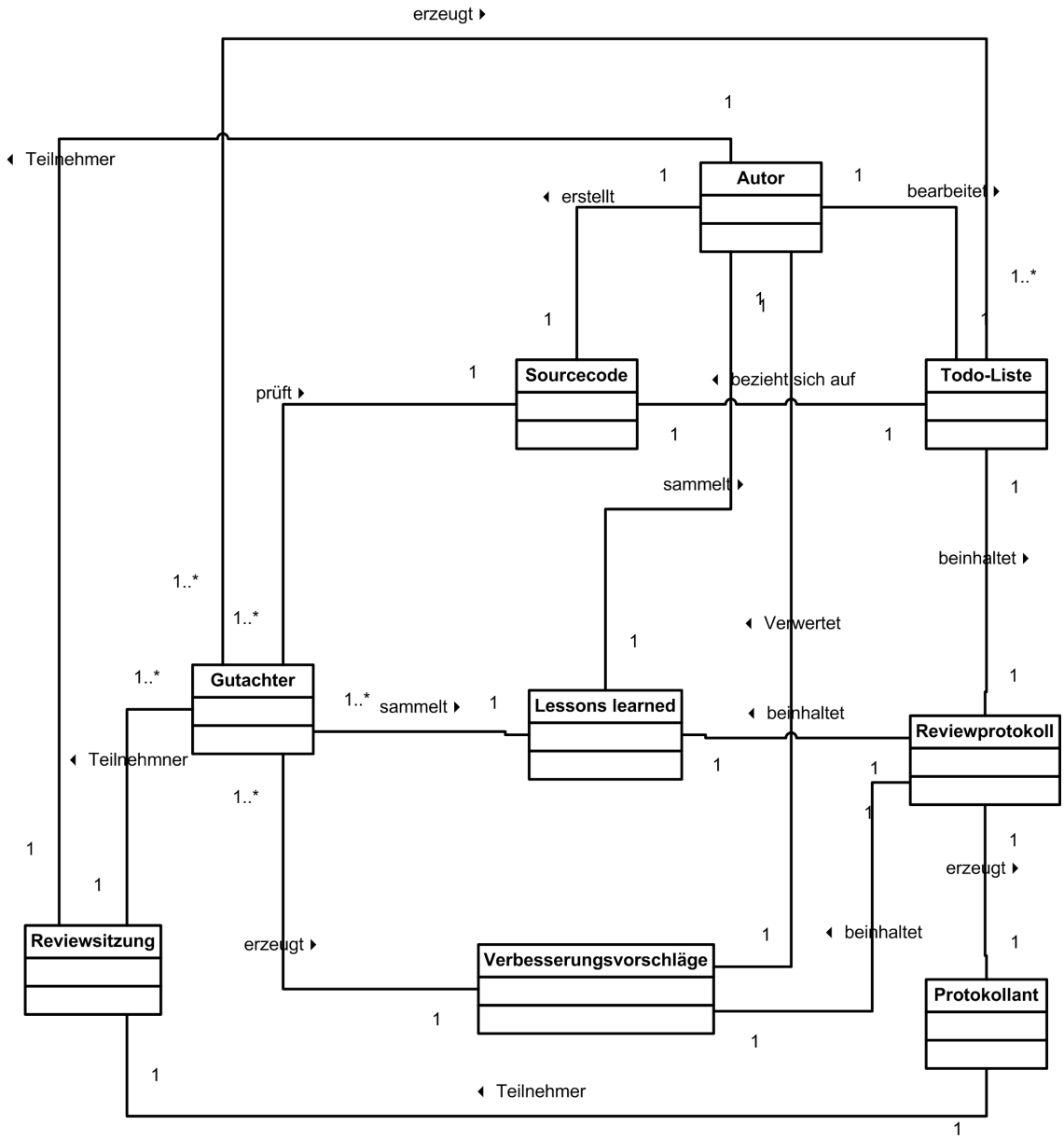



**Abbildung 3.15: Statisches Modell des informellen Codereviews**

Das informelle Codereview unterteilt sich in die Phasen „Vorbereitung“, „Reviewsitzung“ und „Nachbereitung“.

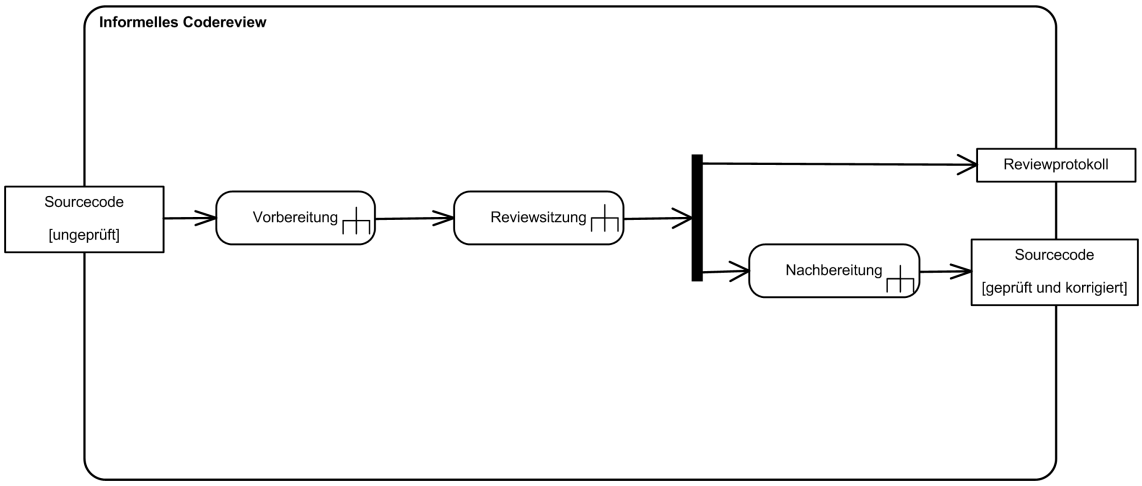



**Abbildung 3.16: Ablauf des informellen Codereviews**

In der Vorbereitungsphase stellt der Autor oder ein Projektleiter ein Reviewteam zusammen und beruft eine Reviewsitzung ein.

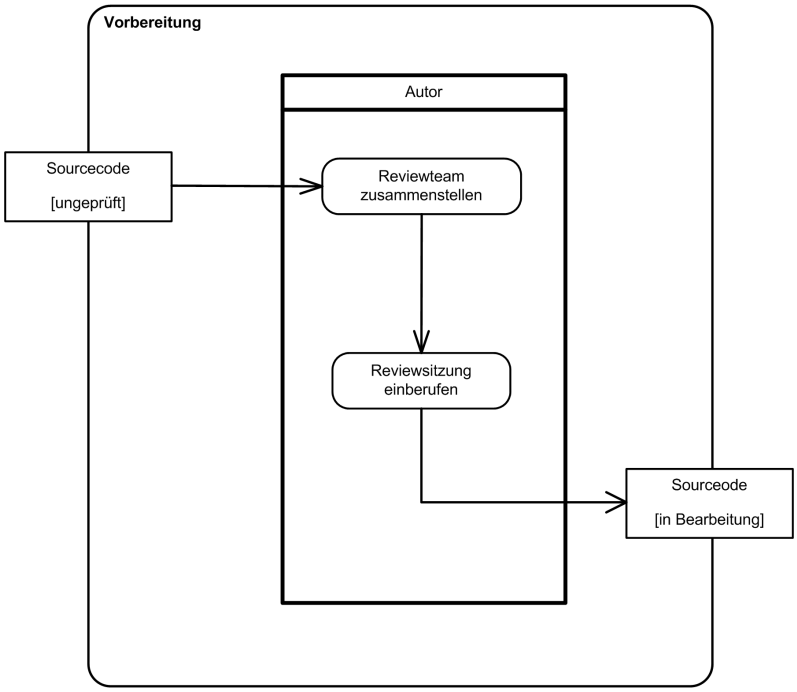



**Abbildung 3.17: Vorbereitung des informellen Codereviews**

In der Reviewsitzung stellt der Autor dann seinen Sourcecode vor und arbeitet ihn zusammen mit den Gutachtern durch. Dabei werden Anomalien diskutiert und Verbesserungsvorschläge gemacht. Zum Ende der Reviewsitzung werden, wie auch in den anderen zwei Reviewprozessen die „lessons learned“ gesammelt. Der Protokollant hält die Ergebnisse des Codereviews in einem Reviewprotokoll fest.

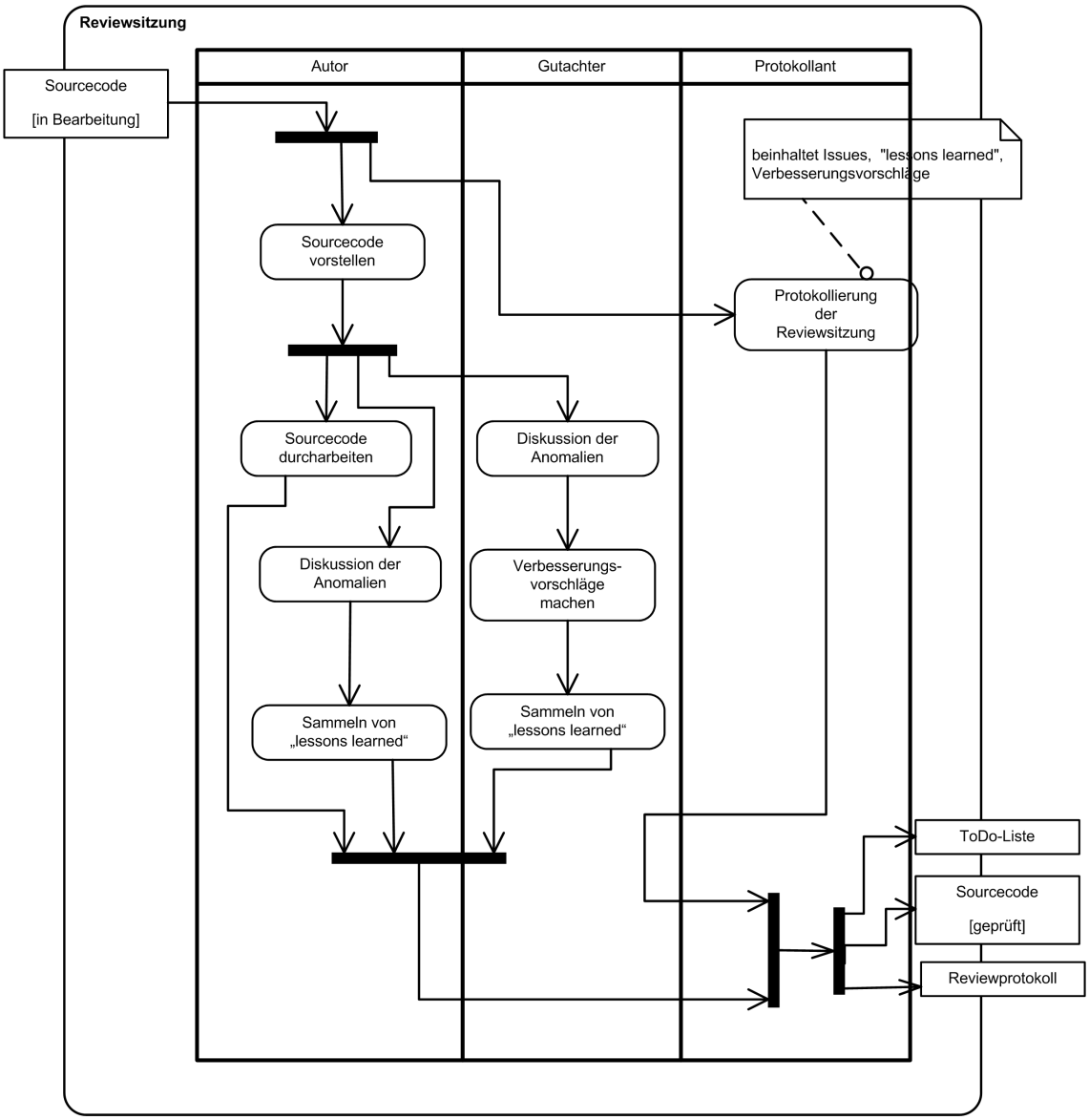



**Abbildung 3.18: Reviewsitzung des informellen Codereviews**

Den Abschluss des Codereviews bildet auch in diesem Reviewprozess die Phase „Nachbereitung“. Diese ist identisch mit den bereits vorgestellten Nachbereitungsphasen der anderen beiden Reviewprozesse.

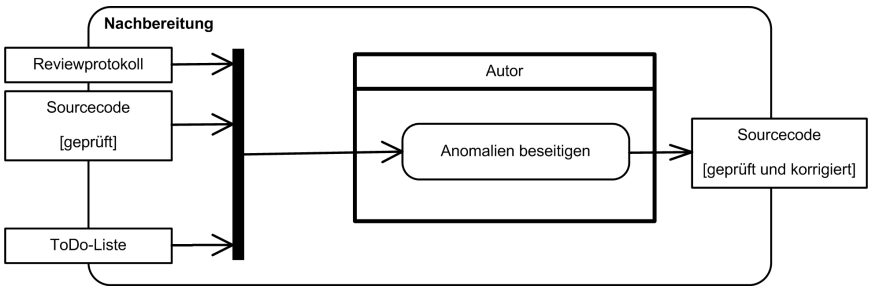



**Abbildung 3.19: Nachbereitung des informellen Codereviews**

# 4 Werkzeuggestützte Codereviews

Dieses Kapitel setzt sich mit werkzeuggestützten Codereviews auseinander. Dabei wird eine Einführung in Reviewtools gegeben und es werden exemplarisch zwei kommerzielle Reviewtools kurz vorgestellt.

## 4.1 Einfache Reviewtools

Das einfachste Reviewtool ist der Editor, mit dem der Sourcecode erstellt und betrachtet wird. Mit ihm kann man einfach Kommentare an den vakanten Stellen im Sourcecode einfügen. Dies ist keine sehr komfortable Methode, aber durchaus eine einfache, unkomplizierte Methode um Anmerkungen zum Sourcecode auch an den entsprechenden Stellen zu dokumentieren. Der Autor kann nach dem Review den entsprechenden Sourcecode durcharbeiten und die Änderungen an den kommentierten Stellen durchführen.

Etwas komfortabler gestalten sich Codereviews bei der Nutzung von Eclipse als Entwicklungsumgebung. Dort bietet Eclipse die Möglichkeit, Tasks an beliebigen Stellen im Sourcecode zu setzen. Am Rand dieser Zeile erscheint anschließend ein bestimmtes Symbol, das auf diesen Task hinweist. Diese Tasks werden dann mit der entsprechenden Datei, in der sie gemacht wurden und der dazugehörigen Zeilennummer in einer Tabelle gespeichert. Von dort aus kann dann wieder an die entsprechende Stelle im Sourcecode zurückgesprungen werden. Diese Funktionalität kommt schon sehr nah an die eines komplexes Reviewtools heran. Tasks bieten jedoch z. B. nicht die Möglichkeit, Fehler zu klassifizieren. Abbildung 4.1 zeigt die Nutzung von Tasks in Eclipse.

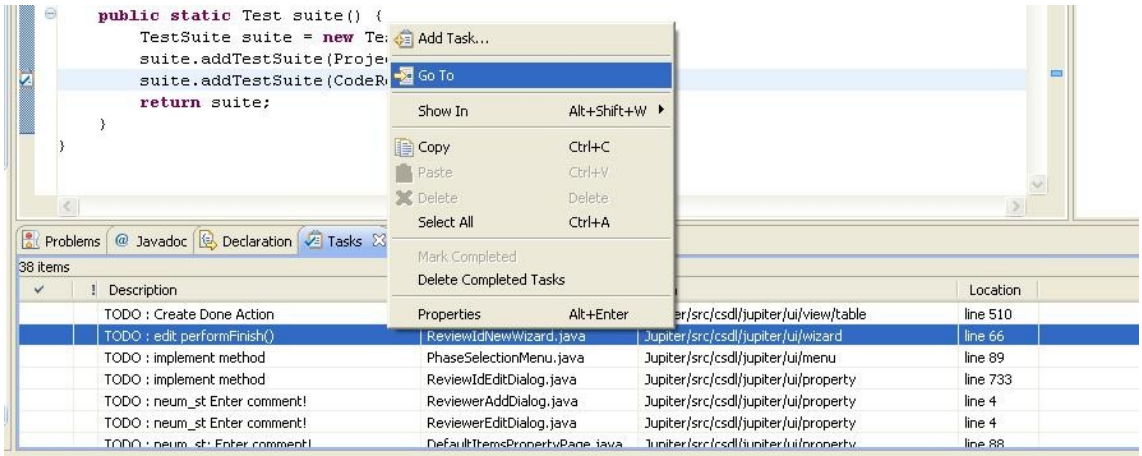



**Abbildung 4.1: Tasks in Eclipse**

## 4.2 Komplexe Reviewtools

Komplexe Reviewtools bieten weit mehr Möglichkeiten, als nur Anmerkungen im Sourcecode zu machen. Ein ganz wichtiges Feature ist die Klassifizierung von Fehlern, zumindest in „major“ und „minor“, also schwere und leichte Fehler. Dies ist wichtig, um bei der Abarbeitung der Fehler Prioritäten setzten zu können. Des Weiteren sollte ein komplexes Reviewtool alle zum Review gehörigen Daten mit verwalten. Dazu gehören die Reviewteilnehmer, die Auflistung der zu prüfenden Files, Datum und Uhr-

zeit des Reviews usw.. Im Idealfall kann mit den Reviewtools so automatisch ein Reviewreport generiert werden. Beispiele für komplexe Reviewtools werden im nächsten Abschnitt gegeben.

## 4.3 Kommerzielle Reviewtools

Es gibt am Markt verschiedene Ansätze, wie Reviewtools realisiert wurden. Populär sind dabei Tools mit einem Webinterface, sodass Codereviews online durchgeführt werden können. Hier können Reviewer dann sogar online über den Code diskutieren. Ein anderer Ansatz ist die Integration des Reviewtools in die Entwicklungsumgebung.

Beispielhaft werden dazu an dieser Stelle das webbasierte Reviewtool Code Collaborator und das Eclipse-Reviewtool EclipsePro Audit vorgestellt.

Neben den kommerziellen Reviewtools gibt es natürlich auch einige Open-Source-Produkte wie bspw. „Codestriker“ [URL:CS] oder auch „Jupiter“ [URL:Jupiter]. Da Jupiter das zurzeit im DLR eingesetzte Tool ist, wird es in Kapitel 6 noch genauer betrachtet.

### 4.3.1 Code Collaborator

Code Collaborator [URL:CC] ist ein webbasiertes Peer-Codereview-System der Firma Smart Bear Software. Es bietet die Möglichkeit Codereviews online durchzuführen.

Dabei wird zuerst ein Review erzeugt und die entsprechenden Dateien mittels Version Control Integration zum Review hinzugefügt. Den Reviewteilnehmern werden ihre Rollen zugewiesen und sie erhalten automatisch eine E-Mail, wenn ein neues Codereview für sie bereitsteht. Das Review kann nun online durchgeführt werden und die Reviewteilnehmer können per integrierter Chatfunktion miteinander kommunizieren. Anomalien werden kategorisiert und automatisch abgespeichert. Um ein Review abschließen zu können, müssen die Reviewer die vom Autor durchgeführten Korrekturen der Anomalien bestätigen. Abschließend erzeugt Code Collaborator dann einen Reviewreport mit allen Reviewissues und den Reviewmetriken. Des Weiteren bietet Code Collaborator eine Integration verschiedener Versionsverwaltungssysteme, Bugtrackingsysteme und verschiedener Entwicklungsumgebungen.

Code Collaborator ist gerade für Projekte, bei denen die Mitarbeiter weit entfernt voneinander arbeiten eine gute Lösung. Auch die Integration verschiedenster anderer Softwareentwicklungstools ist besonders hervorzuheben.

### 4.3.2 EclipsePro Audit

EclipsePro Audit [URL:EclipseProAudit] ist ein Eclipse-basiertes Reviewtool der Firma Instantiations. Es arbeitet dynamisch und erkennt Abweichungen von Standards. Dabei lassen sich neben den voreingestellten Regeln auch weitere Regeln für bestimmte Anwendungen selbst definieren. EclipsePro Audit schlägt dann für die Anomalien direkt eine Verbesserungsmöglichkeit vor. Ebenso lassen sich automatisch ein Reviewreport und Reviewmetriken erstellen. Zudem bietet EclipsePro Audit noch weitere vielfältige Möglichkeiten der Codeanalyse, deren Erläuterung den Rahmen dieser Arbeit sprengen würde.

Als Tool für einfache Codereviews erscheint es jedoch zu komplex und gerade die Dynamik dieses Tools bei der Erkennung von Anomalien lenkt die Reviewer zu sehr von eventuellen Fehlern ab, die das Tool nicht selbstständig erfasst.

# 5 Die Eclipse-Plattform

Da sowohl das bestehende Codereviewtool Jupiter als Eclipse-Plugin implementiert wurde, als auch das neue Codereviewtool in die Eclipse-Plattform eingebettet werden soll, wird in diesem Kapitel Eclipse näher betrachtet. Dabei wird nach einer kurzen Vorstellung der Eclipse-Plattform, deren Architektur detailliert betrachtet.

## 5.1 Einführung in Eclipse

Eclipse [URL:Eclipse] ist eine Open-Source-Software, die im November 2001 veröffentlicht wurde. 2003/2004 führte IBM mit der Eclipse Foundation ein Konsortium ein, das die Weiterentwicklung bestimmt. Diesem Konsortium gehören unter anderem die Mitglieder BEA, Borland, Computer Associates, Intel, HP, SAP, und Sybase an [JAVAINSEL6]. Laut www.eclipse.org ist Eclipse eine Plattform für „alles Mögliche und nichts im Besonderen“ [DAUM05]. Den Meisten wird Eclipse jedoch als Entwicklungsumgebung für Java-Programme bekannt sein. Doch das ist nur ein spezielles Anwendungsfeld von Eclipse. Das eigentlich Anwendungsfeld geht weit darüber hinaus.

Die Plugin-Architektur von Eclipse macht es dem Nutzer möglich eigene Entwicklungen in die Eclipse-Plattform zu integrieren. So stehen heute neben der Entwicklungsumgebung für Java, Plugins für die Entwicklung von C/C++, Python oder anderen Programmiersprachen zur Verfügung.

## 5.2 Die Architektur der Eclipse-Plattform

Eclipse besteht aus einer kleinen Kernapplikation, die lediglich für die Ausführung von Plugins zuständig ist. Alles, was man von Eclipse als IDE (Integrated Development Inironment) kennt, (Editoren, Menüeinträge usw.) läuft als Plugin auf dieser Kernapplikation.

### 5.2.1 Plugin-Manifest

Ein Plugin besteht zumeist aus einem Java Archiv und diversen anderen Dateien wie Icons oder Hilfetexten. Das Plugin wird über das Plugin-Manifest plugin.xml definiert. Das Manifest beschreibt die Konfiguration des Plugins und seine Integration in die Eclipse-Plattform.

### 5.2.2 Extension Points

Grundlegend für die Plugin-Architektur sind Extension Points. Diese Extension Points beschreiben, an welchen Stellen das Plugin an die restliche Plattform „angestöpselt“ wird. Dabei können Plugins auch eigene Extension Points definieren, sodass andere Plugins auf deren Funktionalität zugreifen können. Die Extension Points werden im Plugin-Manifest definiert.

### 5.2.3 OSGi

Seit Eclipse 3.0 entsprechen die Plugin-Formate dem Open Service Gateway Initiative (OSGi) Standard [URL:OSGI]. Dabei erfüllt der Ablaufkern der Eclipse-Plattform die Rolle eines OSGi Servers. Ziel des OSGi Standards ist es Dienste zu standardisieren. Dazu müssen OSGi Dienste das Interface BundleActivator implementieren und eine

OSGi-Manifest- Datei bereitstellen. Hierzu meldet der BundleActivator den Dienst beim OSGi Server, also dem Eclipse-Ablaufkern, an und auch wieder ab. In Eclipse ist dafür die abstrakte Klasse „Plugin“ zuständig, die einen BundleActivator implementiert.

### 5.2.4 Ressourcenverwaltung

Ressourcen sind in Eclipse: Projekte, Verzeichnisse und Dateien. Die Ressourcenverwaltung ist ebenfalls ein Plugin, welches auf dem Ablaufkern ausgeführt wird. Dieses Plugin implementiert ein von dem jeweiligen Betriebssystem unabhängiges Dateisystem. Interessant ist hierbei der dazugehörige Mechanismus zur Markerverwaltung, also z. B. des aus der Eclipse-IDE bekannten Tasks. Ein ebenfalls wichtiges Feature der Ressourcenverwaltung ist die Ereignisverarbeitung von Ressourcenänderungen. So können z. B. Editoren auf die Änderungen an einer geöffneten Datei reagieren. Typisch ist bei diesem Beispiel die Meldung *„The file has been changed on the file system. Do you want to load the changes?“*, wenn die Datei außerhalb des Editors editiert wurde.

### 5.2.5 Benutzeroberflächen

Für die Gestaltung von Benutzeroberflächen stehen mehrere Plugins zur Verfügung. Zum einen SWT und Jface zur Gestaltung eigener GUI-Komponenten (Graphical User Interface) und Dialoge, zum anderen Komponenten wie Views, Wizards, Editoren usw., die schon fertige Komponenten zur Verfügung stellen. Die fertigen Komponenten bilden einen Rahmen, der dann mit SWT und Jface nach den eigenen Bedürfnissen gestaltet werden kann.

Da auch die Eclipse-Workbench mit diesen Plugins realisiert wurde, können so Benutzeroberflächen erstellt werden, die sich perfekt in die Eclipse-Workbench integrieren.

### 5.2.6 Weitere Extension Points

Eclipse bietet noch eine Vielzahl weitere Extension Points, mit deren Hilfe eigene Plugins erstellt werden können. Dazu zählen bspw. Hilfesysteme, CheatSheets, PreferencePages, PropertyPages usw.

### 5.2.7 Die Kernklassen der Eclipse-Plattform

Die wichtigsten Klassen bei der Entwicklung eines Plugins sind die Klassen „Platform“, „Plugin“ und „Preferences“.

Die Klasse „Platform“ verwaltet alle installierten Plugins und regelt den Zugriff auf die Eclipse-Ressourcen.

Die Klasse „Plugin“, bzw. „AbstractUIPlugin“ bei Plugins mit Benutzeroberfläche, ist die Basis für alle Plugins. Alle Plugins sind von dieser abstrakten Klasse abgeleitet. Über eine Instanz der von „Plugin“ abgeleiteten Klasse kann dann auf alle Ressourcen der Plattform zugegriffen werden, Einstellungen des Plugins abgefragt oder gesetzt werden, sowie auf im Plugin gespeicherte Bilder zugegriffen werden.

Die Klasse „Preferences“ verwaltet die Speicherung für Einstellungen des Plugins. Dabei besteht jede Präferenz aus einem Wertepaar. Diese können dann abgefragt und gesetzt werden.

## 5.3 Entwicklung eigener Plugins

Die Architektur der Eclipse-Plattform bietet vielfältige Möglichkeiten, eigene Plugins zu entwickeln. Dabei werden im Plugin-Manifest die Extension Points definiert, die beschreiben, an welchen Stellen das Plugin auf die Eclipse-Plattform zugreift. Dabei hat man über die Ressourcenverwaltung die Möglichkeit auf Projekte, Verzeichnisse und Dateien zuzugreifen. Des Weiteren kann man mit Hilfe von SWT, Jface und den Extension Points View, Wizard usw. eigene Benutzeroberflächen erstellen. In Eclipse werden mithilfe des „New Plugin Project“-Wizards die Kernklassen automatisch generiert und man kann mit einer komfortablen Benutzeroberfläche das Gerüst des neuen Plugins leicht konfigurieren.

Abbildung 5.1 zeigt noch einmal den Zusammenhang der einzelnen Plattform-Komponenten.

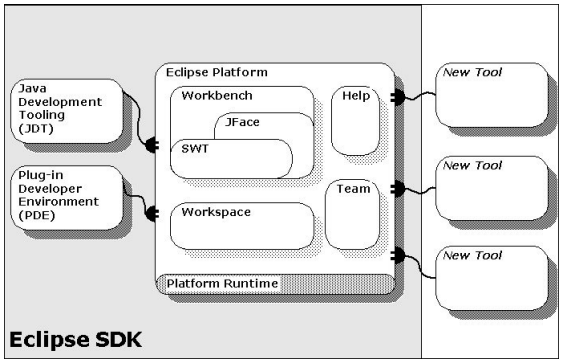



**Abbildung 5.1: Eclipse-Architektur**

Abbildung 5.2 zeigt exemplarisch einige Extension Points.

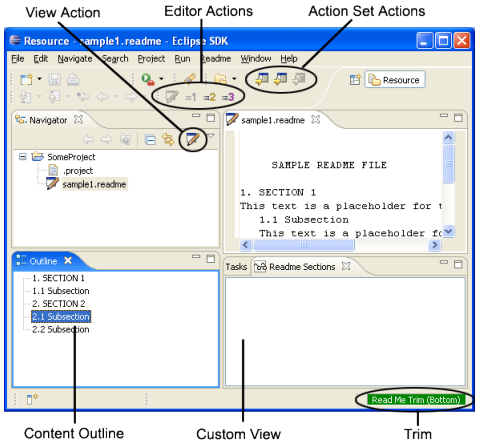



**Abbildung 5.2: Eclipse Extension Points**

# 6 Das Eclipse-Reviewtool Jupiter

In diesem Kapitel wird das zurzeit im DLR eingesetzte Codereviewtool Jupiter vorgestellt. Nach einem kurzen Abschnitt zur Entstehung von Jupiter wird näher auf die Funktionalität und den Reviewprozess von Jupiter eingegangen. Abschließend werden die Vor- und Nachteile von Jupiter gegenübergestellt.

## 6.1 Die Entstehung von Jupiter

Das Tool Jupiter ist eine Entwicklung des Collaborative Software Development Laboratory (CSDL) des Departments of Information and Computer Science der University of Hawaii. Es ist das Resultat von über zehn Jahren Forschung an Reviewtools und -techniken. Bereits 1990 entwickelte das CDSL das erste Reviewtool mit dem Namen Collaborative Software Review System (CSRS), das zwar reich an Funktionalität war, jedoch zu kompliziert in der Installation und in der Nutzung. Nach der Entwicklung eines weiteren Reviewtools namens „Leap“ ist Jupiter die dritte Generation von Reviewtools. Jupiter ist ein Plugin für das Eclipse-IDE-Framework und somit einfach zu installieren und einfach zu nutzen. [URL:JupiterUG]

## 6.2 Jupiter-Features

Laut [Jupiter06] ist das Hauptdesignkriterium des Jupiter-Reviewtools, dass verschiedene Programmiersprachen bzw. einfache Textfiles, sowie verschiedene Reviewprozesse unterstützen werden. Es soll somit in verschiedensten Organisationen einsetzbar sein. Um die oben genannten Kriterien zu erfüllen und um neben der Verbesserung der Softwarequalität auch den Reviewprozess immer weiter zu verbessern, besitzt Jupiter die folgenden Features:

- *Open Source* – Jupiter basiert auf der CPL-Lizenz
- *Free* – Jupiter ist frei und kostenlos im Internet verfügbar
- *IDE Integration* – Jupiter basiert auf der Eclipse-Plugin-Architektur
- *Cross-platform* – Jupiter ist auf allen von Eclipse unterstützen Plattformen verfügbar
- *XML data storage* – Jupiter speichert die Reviewdaten in XML-Dateien
- *CM Repository* – Jupiter-Benutzer können die Reviewdaten genau wie ihre sonstigen Files in jedem beliebigen Repository verwalten
- *Sorting and Filtering* – Jupiter bietet Sortier- und Filtermechanismen, um in einem Review leichter navigieren zu können
- *File Integration* – Jupiter bietet die Möglichkeit, einfach zwischen Review und dem entsprechenden Sourcecode hin und her zu springen

## 6.3 Codereviews mit Jupiter

Der Jupiter-Reviewprozess beinhaltet vier Phasen:

- Configuration Phase
- Individual Review Phase
- Team Review Phase
- Rework Phase

Diese vier Phasen bilden gleichzeitig auch den Jupiter-Reviewprozess ab. Er basiert also auf dem Audit oder dem technischen Review, bei denen vor der eigentlichen Reviewsitzung die Dokumente in Einzelarbeit überprüft werden. Die Vor- und Nachteile, die daraus resultieren, werden im letzten Abschnitt dieses Kapitels noch erläutert. Im Folgenden werden nun die einzelnen Phasen näher betrachtet.

### 6.3.1 Configuration Phase

Der Zweck dieser Phase ist es, ein neues Review anzulegen. Dazu wird unter dem Eintrag „Review“ unter den Projekteigenschaften in Eclipse ein Wizard zum Erzeugen eines neuen Reviews gestartet. In diesem Wizard müssen eine Review-Id, eine kurze Beschreibung zum Review, die zu überprüfenden Dateien, eine Liste von Reviewern, der Autor, der Speicherort des Reviews, die Item-Entries, die Default-Items und die Filtereinstellungen angegeben bzw. eingestellt werden.

Abbildung 6.1 zeigt die „Item Entries Wizard Page“. Dort können die Kategorien Type, Severity, Resolution und Status Item-Entries hinzugefügt, gelöscht oder editiert, sowie deren Reihenfolgen geändert werden. Diese Item-Entries können im Review den einzelnen Review-Issues zugewiesen werden.

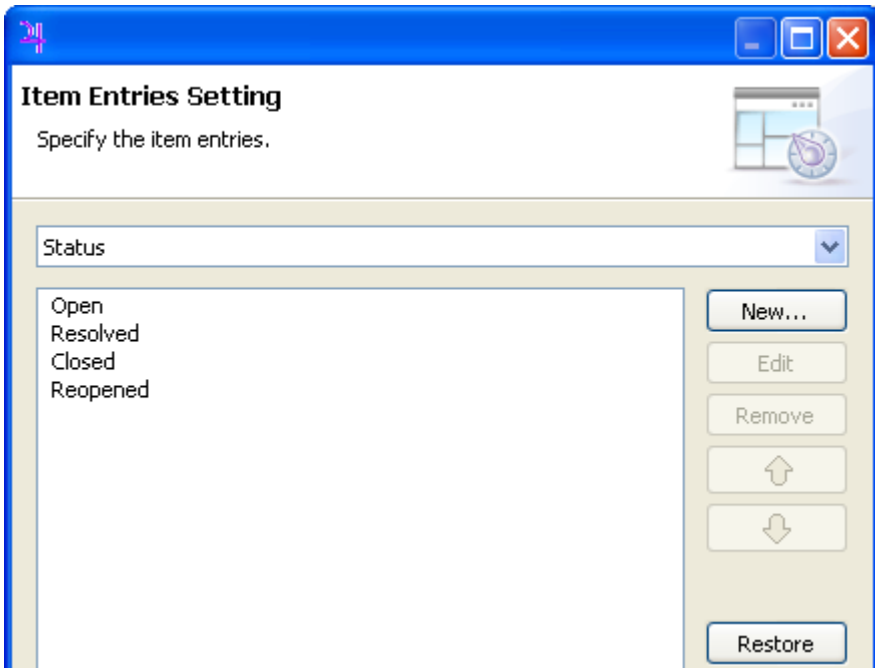



**Abbildung 6.1: Item Entries Wizard Page**

Auf der folgenden WizardPage, der „Default Items Page“, können nun den einzelnen Kategorien ihre Defaultwerte zugewiesen werden. Diese Werte sind dann beim Erzeugen eines neuen Reviews voreingestellt.

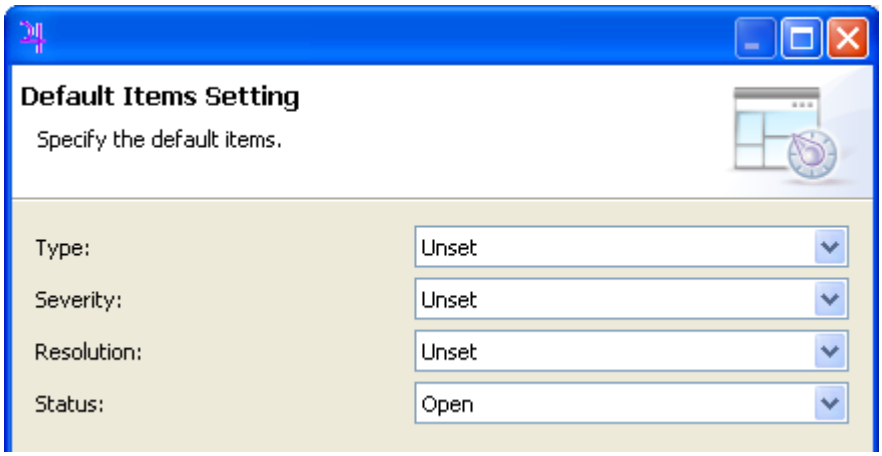



**Abbildung 6.2: Default Items Wizard Page**

Auf der „Filter Wizard Page“ können diverse Filtereinstellungen für die einzelnen Review-Phasen durchgeführt werden. Abbildung 6.3 zeigt hierzu exemplarisch die Filtereinstellungen für die „Rework Phase“.

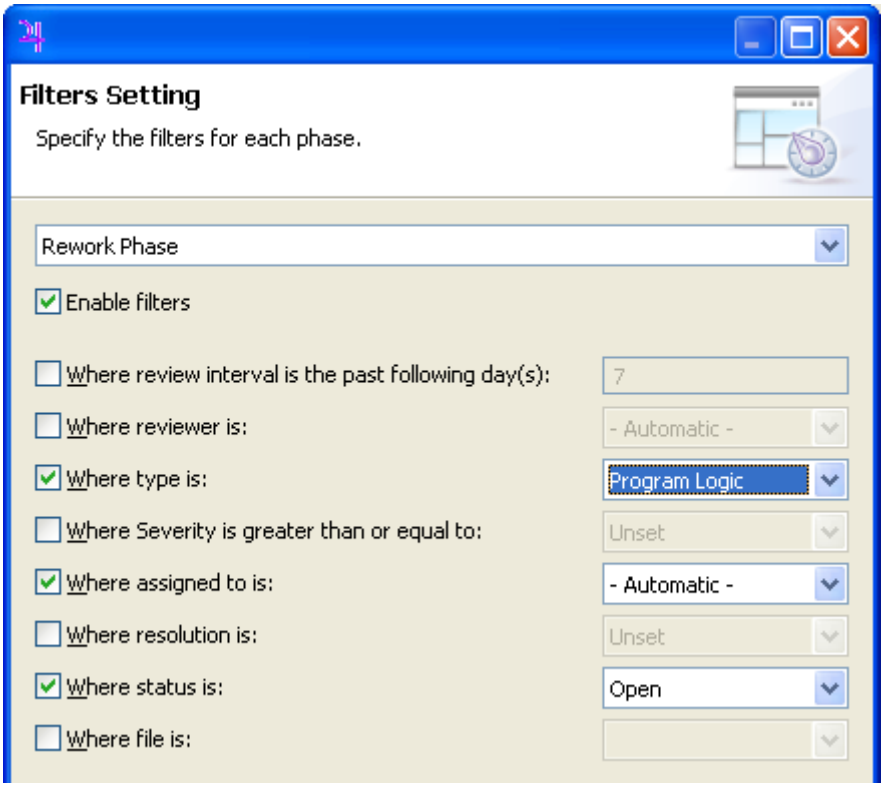



**Abbildung 6.3: Filter Wizard Page**

Bei dieser Filtereinstellung werden z. B. in der „Rework Phase“ nur die Review-Issues aufgelistet, bei denen der Issue-Typ vom Typ „Program Logic“ ist.

Unter dem Eintrag „Review“ unter den Projekteigenschaften von Eclipse können auch bestehende Reviews editiert oder gelöscht werden. Die Reviewtabelle in Abbildung 6.4 zeigt ein ebenfalls erwähnenswertes Feature: die „Default Review Configuration“. Einträge, die in der „Default Review Id“ gemacht wurden, dienen als Default-Einträge beim Erzeugen eines neuen Reviews.

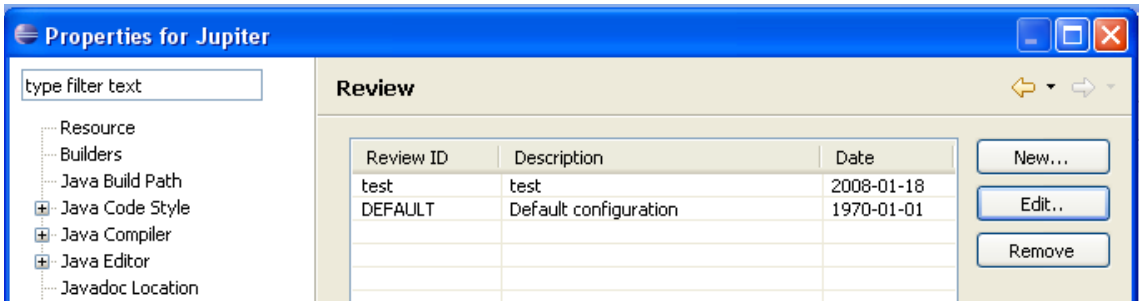



**Abbildung 6.4: Anlegen eines neuen Reviews**

Nachdem ein Review angelegt wurde, kann ein Review mit der Auswahl der „Individual Review Phase“ gestartet werden.

### 6.3.2 Individual Review Phase

Die „Individual Review Phase“ wird über das Pulldown-Menü des Jupiter-Buttons in der Toolbar der Eclipse-IDE gestartet. Dort muss der Eintrag „Individual Phase“ ausgewählt werden.

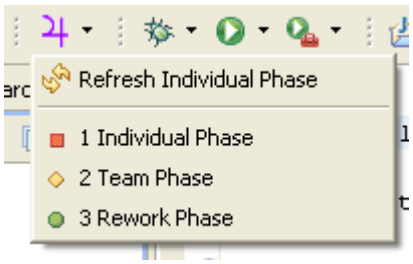



**Abbildung 6.5: Auswahl der Reviewphase**

In einem Dialogfenster müssen dann noch das entsprechende Projekt, die Review-Id und der Reviewer ausgewählt werden.

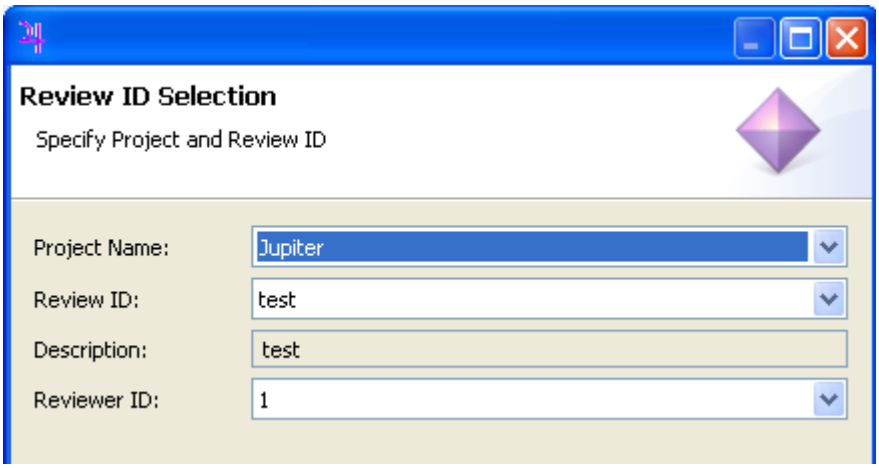



**Abbildung 6.6: Auswahl des Projekts, der Review Id und des Reviewers**

Existiert noch kein Review, so erscheint ein Hinweis, dass noch keine Review-Id existiert und man gelangt automatisch zum Wizard zum Erzeugen eines neuen Reviews.

Wurde der in Abbildung 6.6 gezeigte Dialog bestätigt, kann das eigentlich Review beginnen. Dazu können einfach Stellen im Sourcecode markiert werden und mit einen Rechtsklick und der Auswahl von „Add Jupiter Issue“ ein Review-Issue zum Review hinzugefügt werden.

Der markierte Codeabschnitt erscheint dann im „Review Editor“. Dort können dann noch Type und Severity ausgewählt werden. Der markierte Codeabschnitt steht im Description-Textfeld des „Review Editors“. Im Textfeld Summary kann dann noch eine kurze Zusammenfassung geschrieben werden.

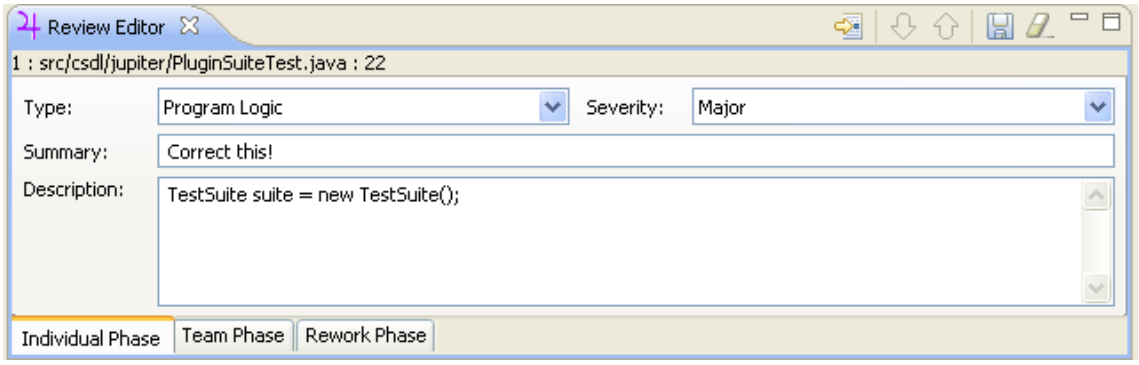



**Abbildung 6.7: Review Editor in der Individual Phase**

Im Editor des Sourcecodes wird der Review-Issue durch eine kleine violette Fahne markiert.

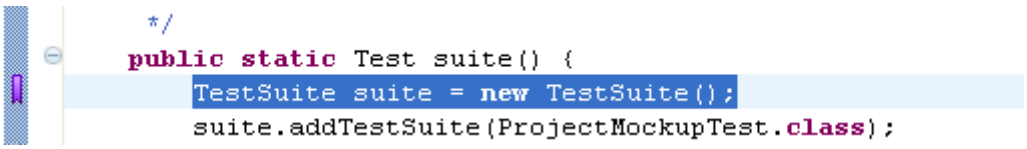



**Abbildung 6.8: Kennzeichnung des Review Issues**

Alle zum Review gehörigen Issues werden in einer weiteren View, dem „Review Table" aufgelistet.

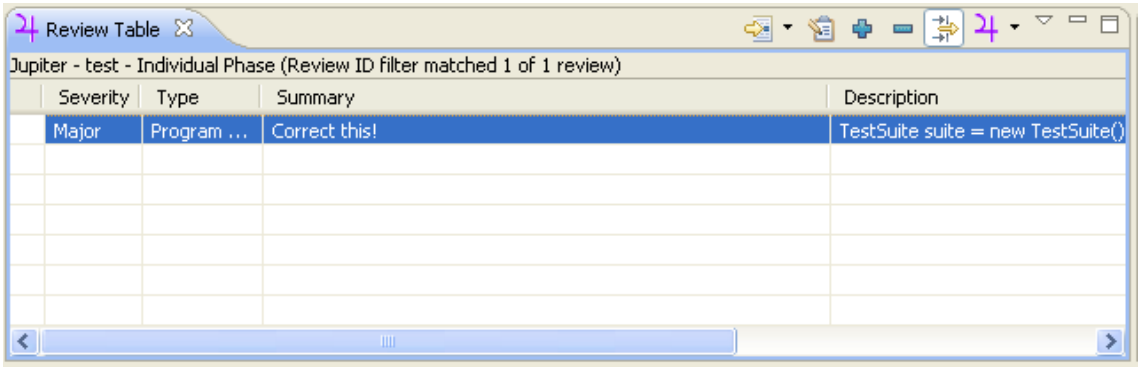



**Abbildung 6.9: Review Table**

Von dort aus kann dann direkt an die entsprechende Stelle im Sourcecode gesprungen werden. Die einzelnen Review-Issues können auch editiert und gelöscht werden. Des Weiteren besteht die Möglichkeit, die Tabelle nach einzelnen Spalten zu sortieren. Dies ist besonders in der „Team Review Phase" und der „Rework Phase" von großem Nutzen.

Die Reviews werden automatisch gespeichert und es bedarf keiner besonderen Aktion, um die „Individual Review Phase" abzuschließen.

### 6.3.3 Team Review Phase

Zum Starten der „Team Review Phase" muss im Pulldown-Menü des Jupiter-Buttons der Eintrag „Team Phase" ausgewählt werden. Dort wird, wie schon im vorherigen Abschnitt bereits beschrieben, die Auswahl von Projekt, Review-Id und Reviewer verlangt. In dieser „Team Review Phase" bleibt der „Review Table" in seiner bisherigen Form bestehen, der „Review Editor" wechselt jedoch in die „Team Review" Ansicht.

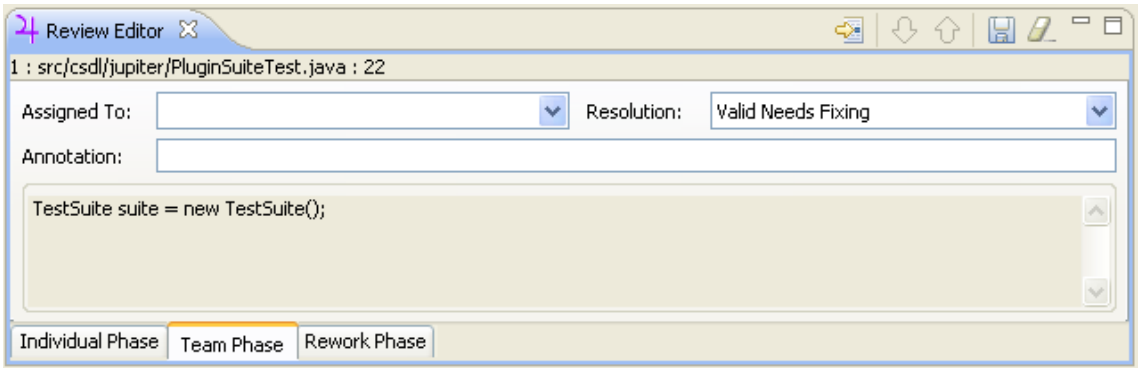



**Abbildung 6.10: Review Editor in der Team Phase**

In der „Team Review Phase" werden nun die in der „Individual Review Phase" gemachten Anmerkungen diskutiert. Es können weitere Anmerkungen im Textfeld Annotation gemacht werden und der Review-Issue kann einem Reviewer zugewiesen werden. Außerdem kann im Feld Resolution ausgewählt werden, wie mit dem Issue verfahren werden soll.

So können auch mithilfe des „Review Tables“ alle Issues nacheinander abgearbeitet werden.

### 6.3.4 Rework Phase

Die „Rework Phase“ bildet den Abschluss eines jeden Reviews. Hierzu wird wie bereits in den vorherigen beiden Review-Phasen, die Phase über den Jupiter-Button ausgewählt. Im „Review Table“ erscheinen nun, je nach Filtereinstellung, nur die Review-Issues, die dem entsprechenden Reviewer in der „Team Review Phase“ zugewiesen wurden. Von dort kann der Reviewer bequem an die entsprechenden Stellen im Sourcecode springen. Der „Review Editor“ wechselt in dieser Phase in die „Rework Phase“-Ansicht. Dort kann der Status des Issues ausgewählt werden und es kann noch ein Kommentar zur Auflösung des Issues hinzugefügt werden.

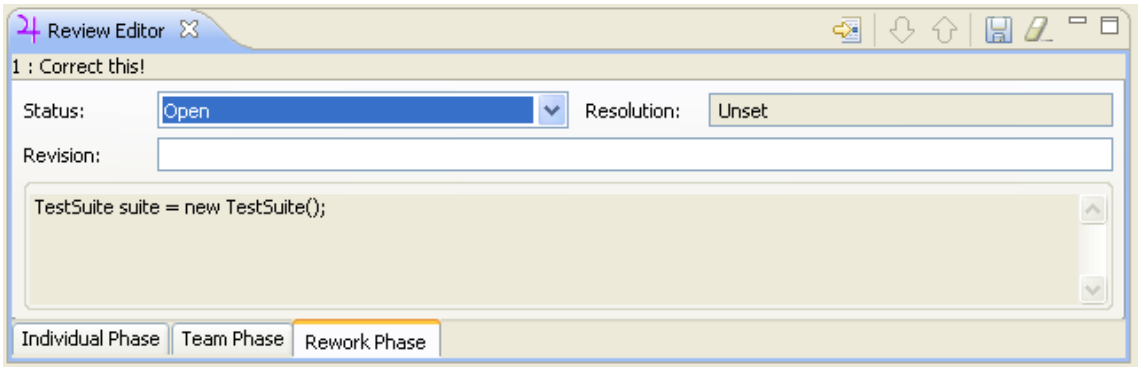



**Abbildung 6.11: Rework Phase**

### 6.3.5 Datenverwaltung

Jupiter speichert alle Review-Informationen in XML-Dateien ab. Dabei erstellt Jupiter eine „.jupiter“-Datei pro Projekt, in der die allgemeinen Informationen zu den zum Projekt gehörigen Reviews enthalten sind.

Diese Datei beinhaltet die folgenden Informationen zu jedem Review:

- Review-Id
- Description
- Author
- Creation-Date
- Directory
- Reviewers
- Field-Items
- Filtereinstellungen

Zudem erzeugt Jupiter einen Ordner in dem pro Review und Reviewer eine „.review“-Datei erzeugt wird, in dem alle vom Reviewer erzeugten Review-Issues gespeichert werden.

Diese Datei beinhaltet die folgenden Informationen zu jedem Issue:

- Review-Id
- Review-Issue-Id (automatisch generiert)
- Creation-Date
- Last-Modification-Date
- Reviewer-Id
- Assigned to
- File + line
- Type, Severity, Resolution, Status
- Summary, Description, Annotation, Revision

## 6.4 Bewertung von Jupiter

Jupiter bietet sehr viele gute Ansätze, hat aber auch Schwächen. An dieser Stelle wird eine abschließende Beurteilung von Jupiter vollzogen.

### 6.4.1 Eclipse-Integration

Die Eclipse-Integration ist eine der großen Stärken von Jupiter, immer vorausgesetzt, dass in den entsprechenden Einrichtungen auch mit Eclipse entwickelt wird. Leider wurde die Integration in die Eclipse-IDE nicht konsequent durchgeführt. So ist bspw. das Erzeugen eines Reviews unter den Projekteigenschaften keine Lösung, die bei anderen Eclipse-Plugins angewandt wird. Dort wäre eine engere Einhaltung der Eclipse-Standards wünschenswert gewesen. Jupiter ist so leider nicht intuitiv bedienbar.

### 6.4.2 Phasenkonzept

Das Phasenkonzept ist für Einrichtungen, die genau diesen Reviewprozess abbilden sicher ideal. Dort wird der Prozess dann eins zu eins im Reviewtool abgebildet. Wird jedoch vom Jupiter-Reviewprozess abgewichen, z. B. bei der Durchführung von Walkthroughs, so ist Jupiter hierfür kaum anwendbar. Problematisch ist dabei, dass nur in der „Individual Review Phase“ Anmerkungen im Sourcecode gemacht werden können. Diese Phase kann also nicht übersprungen werden. Man könnte in einem Walkthrough zwar diese Phase nutzen, es entsteht so jedoch ein Bruch zwischen dem durchgeführten Reviewprozess und dem Jupiter-Reviewprozess.

### 6.4.3 Handhabbarkeit

In der Handhabbarkeit hat Jupiter teilweise Schwächen, gerade dann, wenn man wie oben bereits erwähnt vom Jupiter-Reviewprozess abweicht. Das Erstellen eines neuen Reviews unter den Projekteigenschaften ist dabei ein Punkt, der von jeglichen Standards abweicht.

Die Übersicht der Review-Issues im „Review Table“ und das leichte Rückspringen an die entsprechende Stelle im Sourcecode ist hingegeben gut gelöst. Es erleichtert die Arbeit gerade in der „Team Phase“ und in der „Rework Phase“ erheblich.

Beim Hinzufügen neuer Review-Issues ist der Sprung in den „Review Editor“ von der Handhabung her etwas umständlich. Man muss so bei vielen Anmerkungen sehr oft zwischen dem Sourcecode und der „Review Editor“ hin und her wechseln.

### 6.4.4 Datenverwaltung

Die Datenverwaltung mittels XML-Dateien ist eine gute Lösung, da die Dateien einfach mit dem Sourcecode in das Repository eingecheckt werden können.

### 6.4.5 Schnittstellen

Jupiter fehlen Schnittstellen zu Bugtrackingsystemen oder Fehlerdatenbanken. Dadurch, dass Jupiter jedoch Open-Source ist, können Institutionen, die Jupiter nutzen wollen diese Schnittstellen selber implementieren. Da die Review-Informationen in XML-Dateien vorliegen, können diese leicht weiterverarbeitet werden.

### 6.4.6 Funktionalitäten

Jupiter bringt fast alle Features mit, die ein gutes Codereviewtool braucht. Einzig die Erstellung eines Reviewreports wird nicht unterstützt. Da jedoch, wie oben bereits erwähnt, die Review-Informationen in XML-Dateien vorliegen, kann daraus leicht ein Reviewreport generiert werden.

### 6.4.7 Fehler

Bei der Verknüpfung der Review-Issues mit dem Sourcecode haben sich die Jupiter Entwickler an den Tasks in Eclipse orientiert. Der Review-Issue ist nur an das entsprechende File und die Zeile geknüpft. Wird nun die entsprechende Zeile gelöscht, so bleibt die Markierung mittels des Review-Flags bestehen, verweist nun aber auf den falschen Inhalt.

Es ist ferner möglich auch Anmerkungen in Dateien zu machen, die nicht Teil des gerade ausgewählten Reviews sind.

### 6.4.8 Fazit

Jupiter bietet sehr gute Ansätze. Die vorhandene Funktionalität deckt bis auf die fehlende Anbindung anderer Tools und den fehlenden Reviewreport alle Anforderungen an ein Reviewtool ab. Der starre Reviewprozess ist jedoch ein großer Nachteil, der eine weiter verbreitete Nutzung von Jupiter behindert. Durch den Open-Source-Ansatz bietet sich aber die Möglichkeit, Jupiter flexibler zu gestalten und die Abweichungen von Eclipse-Standards zu beheben.

Eine Betrachtung, in, wie weit Jupiter die Anforderungen an ein Reviewtool in der Einrichtung Simulations- und Softwaretechnik abdeckt, wird am Anfang des Kapitels 8 durchgeführt.

# 7 Anforderungsermittlung

Dieses Kapitel setzt sich mit der Anforderungsermittlung des neuen Reviewtools auseinander. Dabei werden zunächst die Zielbestimmung und die Zielgruppen definiert. Darauf aufbauend werden die funktionalen sowie nicht-funktionalen Anforderungen definiert und ein Ablauf für ein werkzeuggestütztes Codereview festgelegt. Anschließend werden die möglichen Szenarien dokumentiert.

## 7.1 Zielbestimmung und Zielgruppen

In diesem Abschnitt werden die Produktperspektive und der Einsatzkontext des Reviewtools definiert.

### 7.1.1 Produktperspektive

Es ist ein Reviewtool zu erstellen, das die Durchführung von Codereviews unterstützen soll. Es soll in den allgemeinen Softwareentwicklungsprozess und in die allgemeine Werkzeugpalette integriert werden.

### 7.1.2 Einsatzkontext

Das Reviewtool wird zunächst in der Einrichtung Simulations- und Softwaretechnik eingesetzt und erprobt und soll dann DLR-weit eingesetzt werden. Zielgruppe des Produktes sind Softwareentwickler und Projektleiter im DLR.

## 7.2 Funktionale Anforderungen

In diesem Abschnitt werden die funktionalen Anforderungen an das Reviewtool definiert. Die funktionalen Anforderungen unterteilen sich in die Bereiche Produktfunktionen, Produktdaten und Produktschnittstellen.

### 7.2.1 Produktfunktionen

Die Produktfunktionen beschreiben die Funktionalität des Reviewtools.

| Nr. | Anforderung |
|---|---|
| F10 | Ein Review muss neu angelegt werden können. |
| F20 | Ein vorhandenes Review muss ausgewählt werden können. |
| F30 | Die Reviewteilnehmer müssen ausgewählt werden können. |
| F40 | Die Reviewdateien müssen ausgewählt werden können. |
| F50 | Bei der Auswahl der Reviewdateien müssen auch ganze Packages ausgewählt werden können. |
| F60 | Es müssen Anmerkungen zum Sourcecode gemacht werden können. |
| F70 | Anmerkungen zum Sourcecode müssen editiert werden können. |
| F80 | Anmerkungen zum Sourcecode müssen entfernt werden können. |

| | |
|---|---|
| F90 | Reviews müssen editiert werden können. |
| F100 | Reviews müssen gelöscht werden können. |
| F110 | Reviewer müssen editiert werden können. |
| F120 | Reviewer müssen gelöscht werden können. |
| F130 | Reviewer müssen hinzugefügt werden können. |
| F140 | Default-Reviewer müssen gesetzt werden können. |
| F150 | Eine Default-Checkliste muss geöffnet werden können. |
| F160 | Item-Entries müssen editiert werden können. |
| F170 | Default-Item-Entries müssen gesetzt werden können. |
| F180 | Das Review-Storage-Directory muss ausgewählt werden können. |
| F190 | Anmerkungen dürfen nur in einem zum Review gehörigen File gemacht werden können. |
| F200 | Review-Issues müssen nach Mantis exportiert werden können. |
| F210 | Beim Exportieren der Review-Issues müssen diese gruppiert werden können. |
| F220 | Das Generieren eines Reviewreports muss per Knopfdruck möglich sein. |
| F230 | Das Dateiformat des Reviewreports muss PDF sein. |
| F240 | Das Dateiformat des Reviewreports sollte HTML sein. |
| F250 | Falls für den Reviewreport mehrere Dateiformate zur Auswahl stehen, so muss beim Generieren des Reviewreports das Dateiformat auswählbar sein. |
| F260 | Die Dauer des Reviews sollte gemessen werden können. |
| F270 | Die Anzahl der Fehler sollte gemessen werden können. |
| F280 | Bei gefundenen Fehlern muss zwischen den verschiedenen Fehlerschweren unterschieden werden können. |
| F290 | Eine Reviewüberdeckungsmessung sollte per Knopfdruck durchführbar sein. |
| F300 | Bei einer Reviewüberdeckungsmessung muss gemessen werden, zu wie viel Prozent der Sourcecode bereits überprüft wurde. |

| | |
|---|---|
| F310 | Bei einer Reviewüberdeckungsmessung muss zu jeder zu überprüfenden Datei ein Statusbalken angezeigt werden, welcher die prozentuale Reviewabdeckung der Datei anzeigt. |

**Tabelle 7.1: Produktfunktionen**

### 7.2.2 Produktdaten

| Nr. | Anforderung |
|---|---|
| D10 | Zu einem Review müssen ein Name, eine Kurzbeschreibung, die zu überprüfenden Dateien, sowie die Reviewteilnehmer angegeben werden. |
| D20 | Zu jedem Reviewer müssen Reviewer-Id und Name angegeben werden. |
| D30 | Ein Review Issue beinhaltet die folgenden Daten: Issue-Id, Review-Id, Reviewer, File, Zeilennummer, Creation-Date, Last-Modification-Date, Summary, Description, Status, Severity, Type, Mantis-Id. |
| D40 | Ein Reviewreport enthält folgende Daten: Alle Daten des Reviews (siehe D10), das Eclipseprojekt, das aktuelle Tagesdatum, zu jeder Issue-Datei, Zeilennummer, Summary, Severity und Type + optionale weitere Anmerkungen. |

**Tabelle 7.2: Produktdaten**

### 7.2.3 Produktschnittstellen

| Nr. | Anforderung |
|---|---|
| S10 | Das Reviewtool muss in die Eclipse-IDE integriert sein. |
| S20 | Das Reviewtool muss an das Bugtrackingsystem Mantis angebunden sein. |

**Tabelle 7.3: Produktschnittstellen**

## 7.3 Nicht-funktionale Anforderungen

In diesem Abschnitt werden die nicht-funktionalen Anforderungen an das Reviewtool definiert. Die nicht-funktionalen Anforderungen unterteilen sich dabei in die Bereiche Qualitätsanforderungen sowie technische Anforderungen.

### 7.3.1 Qualitätsanforderungen

**Funktionalität**

Es sind alle Vorgänge besonders kritisch, die Änderungen von Daten verursachen. Alle Produktfunktionen müssen fehlerfrei durchführbar sein. Des Weiteren muss das Re-

viewtool so flexibel gestaltet sein, dass die Durchführung der für die Einrichtung neu erstellten Reviewprozesse möglich ist.

**Zuverlässigkeit**

Höchste Priorität hat die Sicherheit der Daten.

**Benutzbarkeit**

Das Reviewtool muss nach dem Eclipse/ Java Styleguide benutzbar sein.

**Effizienz**

Die Dauer der Erstellung eines neuen Review-Issues soll gegenüber dem jetzigen System deutlich gesteigert werden.

**Änderbarkeit**

Spätere Systemerweiterungen um Fehlerdatenbanken und andere externe Tools sind einzuplanen.

### 7.3.2 Technische Anforderungen

Das Reviewtool muss in der Eclipse-IDE Version 3.3 lauffähig sein.

## 7.4 Anwendungsszenarien

Die Anwendungsszenarien beschreiben typische Arbeitsprozesse beim Durchführen eines Codereviews mit dem zu entwickelnden Reviewtool.

### 7.4.1 Erstellen eines neuen Reviews

Der Benutzer startet Eclipse und möchte ein neues Review durchführen. Dazu erzeugt er ein neues Review zu dem entsprechenden Projekt. Dort gibt er die unter D10 geforderten Daten ein. Nach Erstellen des neuen Reviews kann das Review durchgeführt werden.

### 7.4.2 Öffnen eines vorhandenen Reviews

Der Benutzer startet Eclipse und möchte ein bestehendes Review durchführen. Dazu wählt er das entsprechende Review aus einer Liste aus. Anschließend kann das Review durchgeführt werden.

### 7.4.3 Erstellen einer Anmerkung zum Sourcecode

Der Benutzer hat Eclipse bereits gestartet und ein neues Review erstellt oder ein vorhandenes Review ausgewählt. Nun wird dem Benutzer das aktuelle Review angezeigt und er kann Anmerkungen in der zum Review gehörigen Datei machen.

### 7.4.4 Exportieren der Review-Issues in das Bugtrackingsystem Mantis

Der Benutzer hat sein Review abgeschlossen und möchte die Review-Issues in das Bugtrackingsystem Mantis exportieren. Dazu wählt er das entsprechende Review aus. Die Review-Issues können nun gruppiert werden und anschließend nach Mantis ex-

portiert werden. Dabei werden automatisch neue Mantis-Ids erzeugt. Nach dem Exportieren der Issues werden diese als „exportiert“ markiert.

### 7.4.5 Erstellen eines Reviewreports

Der Benutzer hat sein Review abgeschlossen und möchte einen Reviewreport erstellen. Dazu wählt er das entsprechende Review aus. Der Benutzer kann nun das Zielformat des Reviewreports auswählen und die zum Review gehörigen Daten um weitere Kommentare erweitern. Nach Auswahl des Zielverzeichnisses und des Namens des Reviewreports wird dieser erstellt.

### 7.4.6 Durchführung einer Reviewüberdeckungsmessung

Der Benutzer hat sein Review abgeschlossen und möchte nun wissen, zu wie viel Prozent der Sourcecode bereits überprüft wurde. Dazu wählt er das entsprechende Review aus und startet die Reviewüberdeckungsmessung. Er erhält dann eine Auflistung der zum Review gehörenden Dateien samt ihrer Reviewabdeckung. Die Angabe der Reviewabdeckung erfolgt als Balken und als numerische Prozentangabe.

### 7.4.7 Review-Issues bearbeiten

Der Benutzer hat sein Review abgeschlossen und möchte nun die im Review gefundenen Fehler beheben. Dazu wählt er das entsprechende Review aus. Anschließend kann er Issue für Issue abarbeiten. Dazu wählt er den Issue aus einer Liste aus und kann von dort direkt in den Sourcecode springen. Dort führt er die Änderung durch und kann anschließend den Issue als bearbeitet markieren.

## 7.5 Ablauf eines Codereviews

Abbildung 7.1 zeigt den geforderten Ablauf eines Codereviews mit dem neuen Reviewtool. Dieser Ablauf kann in allen Phasen der in Kapitel 3.2 definierten Reviewprozesse durchlaufen werden. Das Tool ist somit nicht an einen speziellen Reviewprozess geknüpft.

Wichtig bei diesem Codereviewprozess ist die Tatsache, dass somit in jeder Phase des Reviews Anmerkungen gemacht werden können. Dadurch können mit dem Tool sowohl Reviews mit als auch ohne individuelle Prüfung durchgeführt werden. Somit ist die größte Schwäche von Jupiter, der zu starre Reviewprozess, beseitigt.

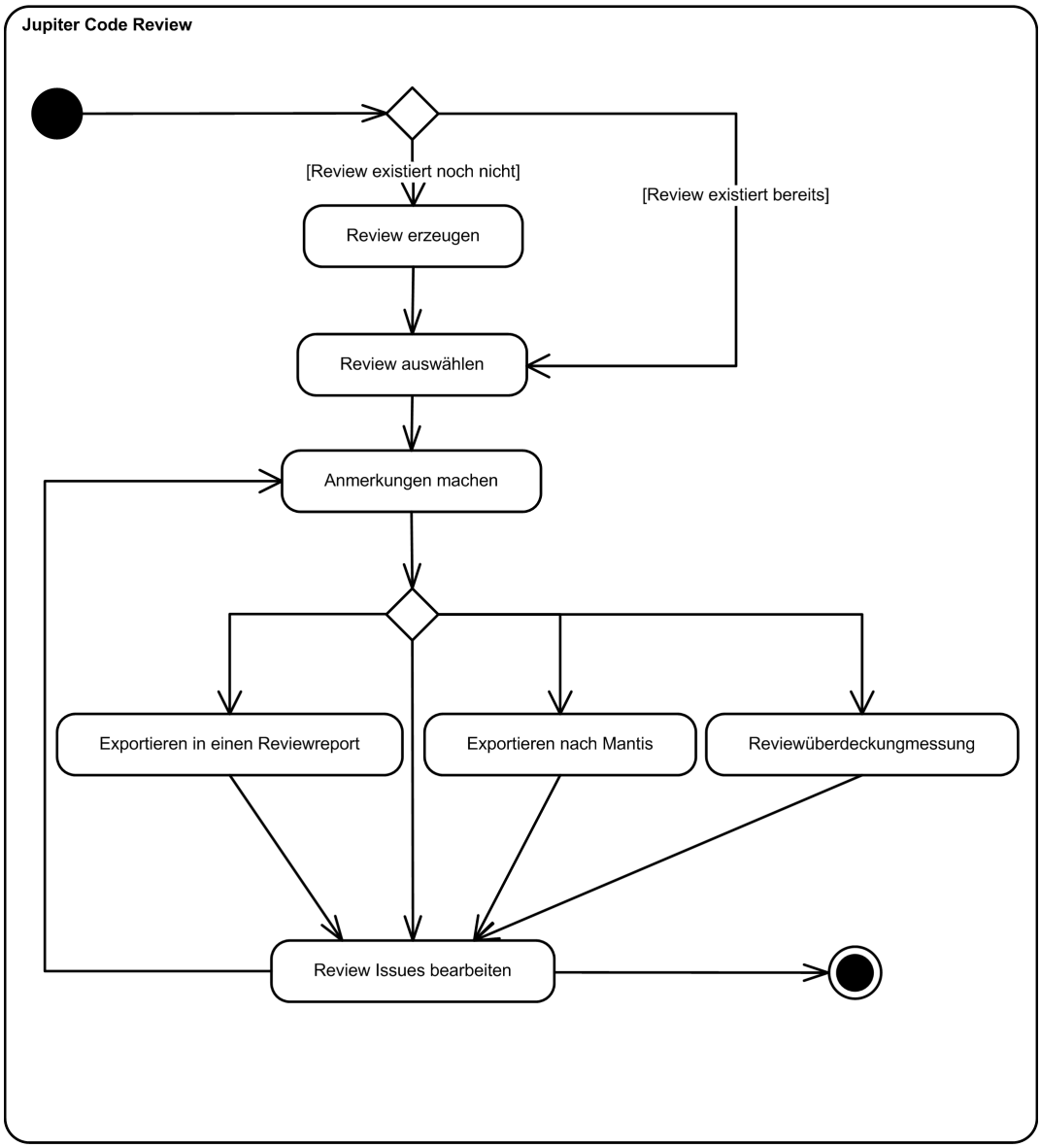



**Abbildung 7.1: Ablauf eines Codereviews**

# 8 Konzeption

In diesem Kapitel folgt die Konzeption des neuen Reviewtools. Dazu erfolgt zu Beginn ein Abgleich der Anforderungen mit dem Tool Jupiter, um zu der Entscheidung zu gelangen, ob Jupiter angepasst und erweitert werden soll, oder ob ein komplett neues Reviewtool erstellt werden soll.

## 8.1 Abgleich der Anforderung und des Reviewprozesses mit Jupiter

Die in Kapitel 3 definierten Reviewprozesse lassen sich mit Jupiter nicht abbilden, da Jupiter immer von einer individuellen Prüfung im Vorfeld der Reviewsitzung ausgeht. Daher können in Jupiter Anmerkungen auch nur in der „Individual Review Phase“ gemacht werden. Dies ist ein großer Nachteil, der gegen eine Erweiterung und Anpassung von Jupiter spricht. Jedoch bietet Jupiter schon viele Funktionen, die in den Anforderungen an ein Reviewtool aus Kapitel 7 definiert sind. So lassen sich beispielsweise bereits Review-Issues mit dem Sourcecode verknüpfen. Auch das Erstellen eines neuen Reviews erfüllt bereits, bis auf kleine Details, die Anforderungen. Ebenso entsprechen die Reviewdaten zu großen Teilen den Anforderungen. Die Speicherung der Review-Issues als XML-Dateien ist ebenso bereits gut gelöst. Ein weiterer großer Vorteil ist auch die Verwaltung der Item-Entries, die so für ein neues Reviewtool nur leicht verändert werden muss.

Um Jupiter als Reviewtool nutzen zu können, müssen folgende Änderungen vorgenommen werden:

- Änderung des Phasenkonzeptes von Jupiter
- Anpassung der Benutzeroberfläche an den Eclipse Standard
- Anpassung der Reviewdaten

Des Weiteren müssen folgende Funktionalitäten hinzugefügt werden:

- Erstellen eines Reviewreports
- Exportieren der Review-Issues nach Mantis
- Reviewüberdeckungsmessung

Trotz einiger Änderungen, die an Jupiter vorgenommen werden müssen, bildet Jupiter eine gute Grundlage für ein neues Reviewtool. Deshalb wird das neue Reviewtool auf Jupiter aufgebaut.

## 8.2 Erweiterung und Anpassung von Jupiter

Da das Reviewtool als Eclipse-Plugin realisiert wurde, unterliegt es auch der Eclipse-Architektur. Die folgenden Konzepte orientieren sich daher an denen in Kapitel 6 vorgestellten Extension Points, mit deren Hilfe das neue Tool an die Eclipse-Workbench angeschlossen wird. Jede neue Funktionalität gehört zu einem Extension Point. Dieser Extension Point bildet dann den Startpunkt der neuen Funktionalität. Die genaue Implementierung wir dann im Kapitel 9 Implementierung beschrieben.

### 8.2.1 Änderung des Phasenkonzeptes

Die wichtigste Änderung an Jupiter ist die Änderung des Phasenkonzeptes. Das neue Tool hat keine speziellen Phasen mehr und ist somit flexibel für verschiedene Reviewprozesse einsetzbar.

### 8.2.2 Entfernen der Filter

Durch das Entfernen der Phasen und durch die Integration des Bugtrackingsystems Mantis werden die Filter nicht mehr benötigt. Diese werden vollständig aus Jupiter entfernt. Die Filterung der Aufgaben für die Nachbereitung erfolgt dann über die Mantis-Id. Näheres dazu folgt im folgenden Kapitel.

### 8.2.3 Anpassung der Reviewdaten

Die allgemeinen Daten zu einem Review werden, wie bereits erwähnt, in der „.jupiter"-Datei gespeichert. Da die Filter entfernt werden, entfallen hier die Angaben zu den Filtereinstellungen. Auch die Angabe eines Autors ist für das Review nicht von großer Bedeutung. Der Name des oder der Autoren ist bereits im entsprechenden Sourcecode vermerkt.

Daraus ergeben sich die folgenden Änderungen an den Review-Id-Daten:

| Daten | vorher | nachher |
|---|---|---|
| Review-Id | x | x |
| Description | x | x |
| Author | x | |
| Creation-Date | x | x |
| Directory | x | x |
| Reviewers | x | x |
| Field-Items | x | x |
| Filtereinstellungen | x | |

**Tabelle 8.1: Review-Id-Daten**

Durch die Änderung des Phasenkonzeptes und besonders durch die Anbindung von Mantis ergeben sich auch einige Änderungen für die Review-Issue-Daten. So entfallen aus der „Team Phase" die „Assigned to"-, „Resolution"- und „Annotation"-Daten. Aus der „Rework Phase" entfallen die „Revision"-Daten.

Die Zuweisung der Verantwortlichkeit mittels „Assigned to" und der Lösungsvorschlag mittels „Resolution" werden durch Mantis geregelt. Zusätzliche Anmerkungen zum Review-Issue benötigen keinen gesonderten Eintrag im Review-Issue und können direkt mit das Summary- oder Description-Feld eingetragen werden.

| Daten | vorher | nachher |
|---|---|---|
| Review-Id | x | x |
| Review-Issue-Id | x | x |

| | | |
|---|---|---|
| Creation-Date | x | x |
| Last-Modification-Date | x | x |
| Reviewer-Id | x | x |
| Assigned to | x | |
| File + line | x | x |
| Summary | x | x |
| Description | x | x |
| Annotation | x | |
| Revision | x | |
| Type | x | x |
| Severity | x | x |
| Resolution | x | |
| Status | x | x |
| Mantis-Id | | x |

**Tabelle 8.2: Review-Issue-Daten**

Ebenfalls geändert werden die Plugin-Preferences. Diese XML-Datei liegt im Workspace-Ordner von Eclipse und konfiguriert somit die Einstellungen des Jupiter-Plugins. Aus dieser Datei werden die View- und Filtereigenschaften entfernt. Hinzugefügt wird jedoch eine Auflistung der verschiedenen Mantis-Verbindungen.

Des Weiteren werden die Daten der Reviewer zukünftig in einer separaten XML-Datei, der „.reviewer"-Datei abgespeichert. Jedes Eclipse-Project enthält dann neben der „.jupiter"-Datei auch diese Datei. Sie enthält eine Auflistung aller potenziellen Reviewer. Beim Erstellen eines neuen Reviews kann dann auf diese Daten zugegriffen werden.

### 8.2.4 Views

Jupiter beinhaltet zwei Views, den „Review Table" und den „Review Editor". Der „Review Table" listet alle Review-Issues des aktuellen Reviews auf. „Der Review Editor" zeigt eine detailliertere Ansicht eines Review-Issues und beinhaltet verschiedene Ansichten für jede der drei Reviewphasen. Diese verschiedenen Ansichten sind für das neue Tool, dass keine Phasen mehr beinhaltet, überflüssig. Ebenso stellte sich bei Codereviews mit Jupiter heraus, dass beim Erstellen eines neuen Issues im Sourcecode der Sprung in der „Review Editor" zeitaufwendig und umständlich ist.

Der „Review Editor" wird somit entfernt. Die Funktionalität des Hinzufügens und Editierens neuer Review-Issues wird von PopUp-Dialogen übernommen. Dort können alle in Abschnitt 8.2.3 aufgelisteten Review-Issue-Daten gesetzt werden.

Abbildung 8.1 zeigt den „Add Review Issue Dialog". Der Fokus liegt direkt im Summary-Feld, sodass dort direkt ein Text eingegeben werden kann. Beim „Edit Review Dialog" werden dann statt der Defaultwerte für Type, Severity und Status die entsprechenden Werte des selektierten Review-Issues eingetragen.

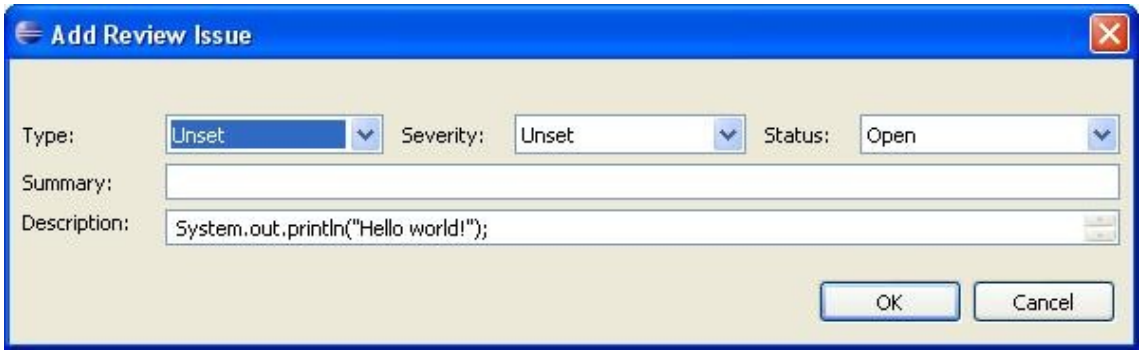



**Abbildung 8.1: Add Review Issue Dialog**

Der „Review Table“ wird in „Review Issue Table“ umbenannt, da er keine Reviews, sonder Review-Issues auflistet.

Alle relevanten Review-Issue-Daten werden im „Review Issue Table“ aufgelistet. Des Weiteren wird dem PopUp-Menü des „Review Issue Table“ ein Eintrag „Done“ hinzugefügt, der den Status des Review-Issues direkt auf „Closed“ setzt. Dies stellt gerade für die Nachbereitung eine große Erleichterung dar.

### 8.2.5 PopUpMenus

Bisher aktiviert der Jupiter-PopUp-Menüeintrag „Add Jupiter Issue“ des Editors den „Review Editor“-View und trägt den markierten Text in das Description-Feld ein.

Da der „Review Editor View“ jedoch entfernt wird, öffnet sich stattdessen der bereits im vorherigen Abschnitt vorgestellte PopUp-Dialog.

Dieser öffnet sich jedoch nur, wenn die Datei Teil des aktiven Reviews ist. Anderenfalls wird ein PopUp-Dialog geöffnet, der den Nutzer darauf hinweist, dass die entsprechende Datei nicht Teil des aktiven Reviews ist.

### 8.2.6 PreferencePages

Durch das Entfernen der Filter in der Jupitererweiterung wird auch die Preference Page „Filter“ entfernt. Hinzugefügt wird jedoch die PreferencePage „Mantis Connections“ in der die verschiedenen Mantis-Verbindungen konfiguriert werden können. Abbildung 8.2 zeigt die „Mantis Connections PreferencePage“. Dort können Verbindungen erstellt, editiert und gelöscht werden. Zu einer Mantis-Verbindung müssen ein Name, eine URL (Uniform Resource Locator), ein Benutzername und ein Passwort angegeben werden. Mit diesen Daten kann sich Jupiter dann beim Bugtrackingsystem Mantis anmelden. Einen detaillierteren Einblick in die Mantis-Anbindung bietet Kapitel 9.6.

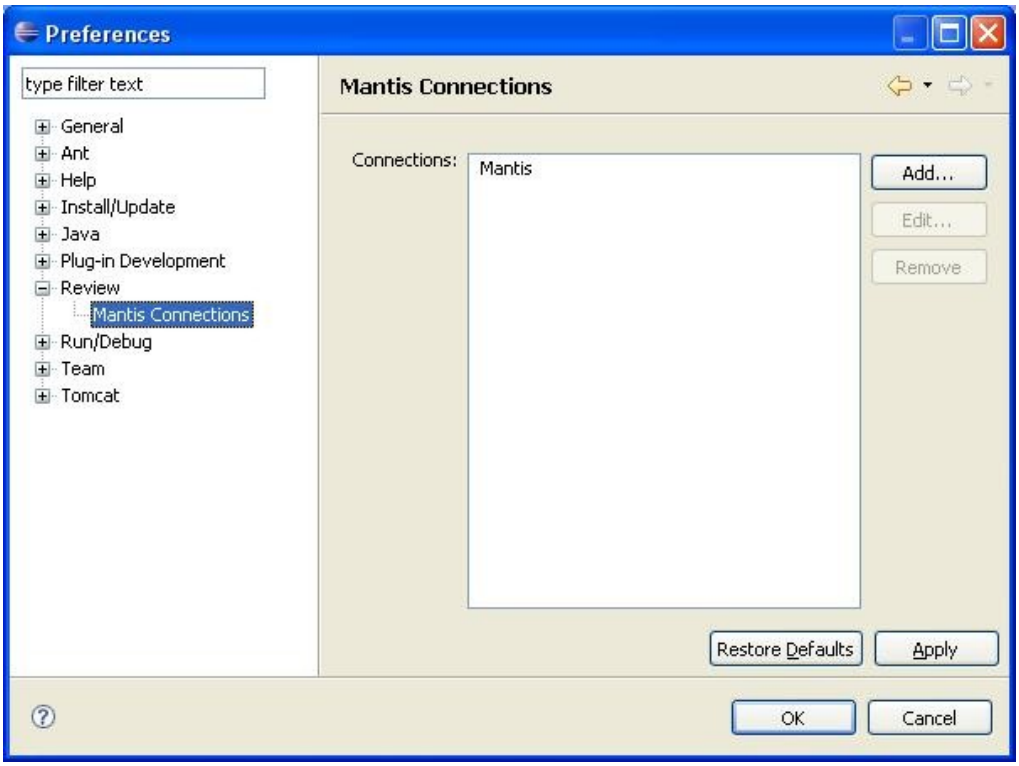



**Abbildung 8.2: Mantis Connections Preference Page**

### 8.2.7 ActionSets

Ein ActionSet stellt einen Menüeintrag und einen Button in der Eclipse-Workbench bereit. Jupiter implementiert einen Pulldown-Button. Dort kann die Phase ausgewählt werden und das Review aktualisiert werden.

Dieser Pulldown-Button wird nach dem Entfernen der Phasen dazu genutzt, schnell auf die wichtigsten Funktionen zugreifen zu können. So kann von dort aus ein neues Review erzeugt oder ein Vorhandenes geöffnet werden. Ebenso kann eine Übersicht der Reviewmetriken angezeigt werden. Abbildung 8.3 zeigt den geänderten Pulldown-Button.

Die Aktualisierung des Reviews ist nicht mehr notwendig, da das Tool selbstständig auf Änderungen reagieren soll.

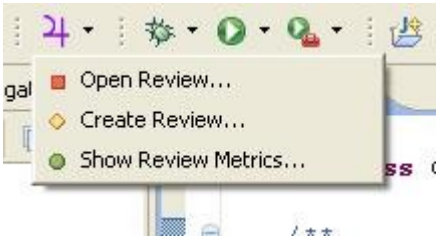



**Abbildung 8.3: Jupiter-Pulldown-Button**

### 8.2.8 PropertyPages

Jupiter bietet eine PropertyPage an, in der Reviews erzeugt, editiert und gelöscht werden können. Das Erzeugen von Reviews an dieser Stelle wird entfernt, da dies nicht den Eclipse-Standards entspricht. Stattdessen wird wie in Abschnitt 8.2.11 noch beschrieben wird, ein neues Review über File -> New erzeugt. Das Editieren eines Reviews wird dann an die veränderten Reviewdaten angepasst. D.h. die Angabe eines Autors und die Konfiguration der Filter wird entfernt.

### 8.2.9 Perspective

Durch den Wegfall des „Review Editor“-Views muss die Perspektive überarbeitet werden. Der „Review Issue Table“ nimmt nun den gesamten unteren Bereich der Eclipse-Workbench ein, sodass in der Tabelle eine Vielzahl von Review-Issue-Daten angezeigt werden kann.

### 8.2.10 CheatSheetsContent

CheatSheets sind kleine Hilfe-Guides, in denen bestimmte Problemstellungen thematisiert werden. Sie werden als View in der Eclipse-Workbench angezeigt. So gibt es bspw. ein CheatSheet, in dem das Erstellen eines neuen Plugins beschrieben wird. Die Jupiter-Erweiterung nutzt die CheatSheets, um dem Nutzer eine Checkliste zur Verfügung zu stellen. Diese soll ihm dann eine Hilfestellung zur bestmöglichen Durchführung eines Reviews geben. Weitere Checklisten können von den Nutzern selbst erstellt werden und dann aus einer externen Quelle in Eclipse geladen werden.

### 8.2.11 NewWizard

Der NewWizard ist ein wichtiger Extension Point, der das Tool an den Eclipse-Standard anpassen soll. Mit Hilfe dieses Extension Points kann dem Eclipse „New Menue“ ein neuer Eintrag hinzugefügt werden. Unter dem Menüpunkt Others - > Review kann nun von dort aus ein Review erzeugt werden. Die Auswahl dieses Eintrags startet dann den Wizard zur Erstellung eines neuen Reviews. Der NewWizard hilft somit das Tool für Eclipse-Nutzer intuitiver bedienbar zu machen.

### 8.2.12 ExportWizard

Dieser Extension Point nimmt eine zentrale Bedeutung bei der Erweiterung von Jupiter ein. Über den ExportWizard werden die Review-Issues nach Mantis exportiert, sowie ein Reviewreport erzeugt. Er stellt also die komplette Anbindung an andere Tools bereit.

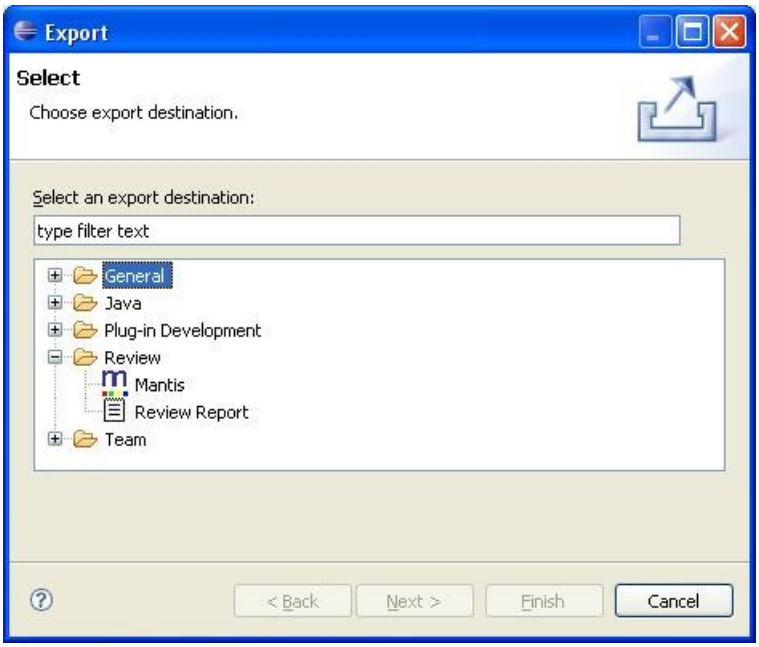



**Abbildung 8.4: Export Wizard**

### 8.2.13 Weitere Extension Points

Jupiter implementiert noch eine Anzahl weiterer Extension Points, die die Verwaltung der Marker und die Definition der Dateien „.jupiter“ und „.review“ übernehmen. Diese Extension Point sind:

- org.eclipse.core.resources.markers
- org.eclipse.ui.ide.markerImageProviders
- org.eclipse.ui.editors.annotationTypes
- org.eclipse.ui.editors.markerAnnotationSpecification
- org.eclipse.ui.ide.markerResolution
- org.eclispse.team.core.fileType

Auf diese Extension Points wird an dieser Stelle jedoch nicht genauer eingegangen, da sie für die Erweiterung von Jupiter nicht verändert werden müssen.

## 8.3 Vorgehensweise bei der Jupitererweiterung

Nach der Anforderungsanalyse und der Konzeption werden zuerst die GUI-Elemente der Extension Points implementiert. So erhält man einen Rahmen, in den die Funktionalität dann integriert werden kann. Die detaillierte Beschreibung der Implementierung folgt im nächsten Kapitel.

# 9 Implementierung

Dieses Kapitel setzt sich mit der konkreten Umsetzung der Konzepte aus Kapitel 8 auseinander. Dabei ist das Kapitel in die durchzuführenden Arbeitsblöcke unterteilt.

## 9.1 Änderung der Reviewdaten

Die Änderung der Reviewdaten betrifft zum einen die allgemeinen Reviewdaten und zum anderen die Daten eines Review Issues.

### 9.1.1 Allgemeine Reviewdaten

Zur Änderung der Review-Id-Daten, also den allgemeinen Daten zum Review, müssen Änderungen in den Paketen „csdl.jupiter.file“ und „csdl.jupiter.model.review“ durchgeführt werden. Des Weiteren müssen Änderungen der XML-Datei „property.xml“ durchgeführt werden.

Das Paket „csdl.jupiter.file“ ist hauptsächlich für die Speicherung der Reviewdaten in die entsprechenden XML-Dateien verantwortlich. Abbildung 9.1 zeigt die Klassen, die zur Verwaltung der allgemeinen Reviewinformationen benötigt werden. Dabei werden die Attribute und Methoden der Übersicht halber jedoch nur andeutungsweise dargestellt.

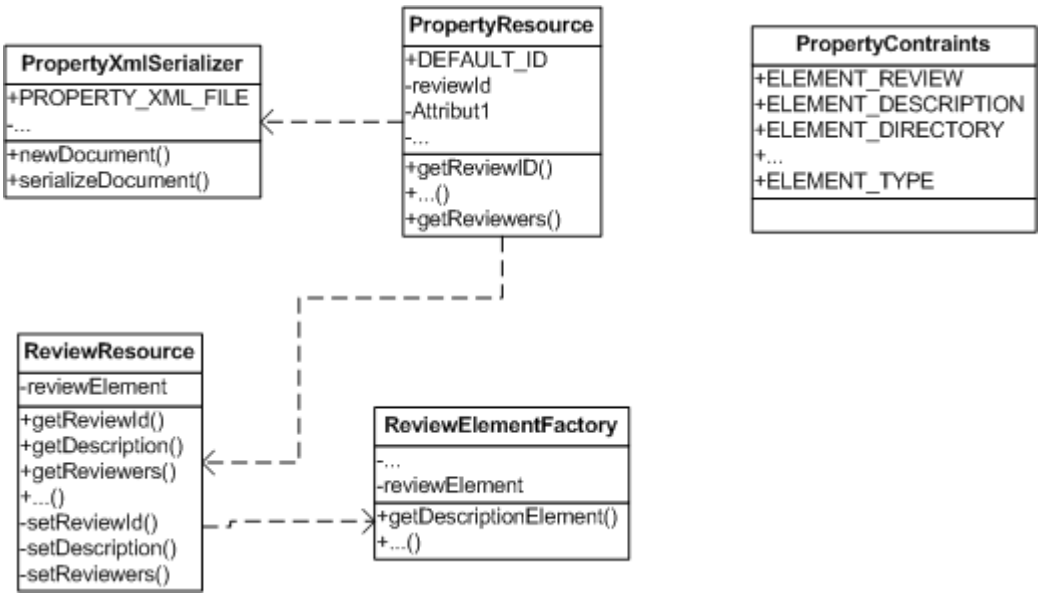



**Abbildung 9.1: Klassen zur Speicherung der allgemeinen Review-Informationen**

Die Klasse „PropertyConstraints“enthält alle Elemente, die für das Erzeugen der „.jupiter“-Datei benötigt werden. Die Klasse „PropertyResource“ ist für das Speichern und Laden der Reviewdaten verantwortlich. Die Klasse „PropertyXmlSerializer“ erzeugt die XML-Datei. Die Klasse „ReviewElementFactory“ regelt den Zugriff auf die XML-Elemente. Die Klasse „ReviewResource“ ist für die Abfrage und das Setzen der Review-Elemente zuständig.

Die für die allgemeinen Review-Daten wichtigen Klassen des Pakets „csdl.jupiter.model.review“ sind „ReviewId“ und „ReviewIdManager“.

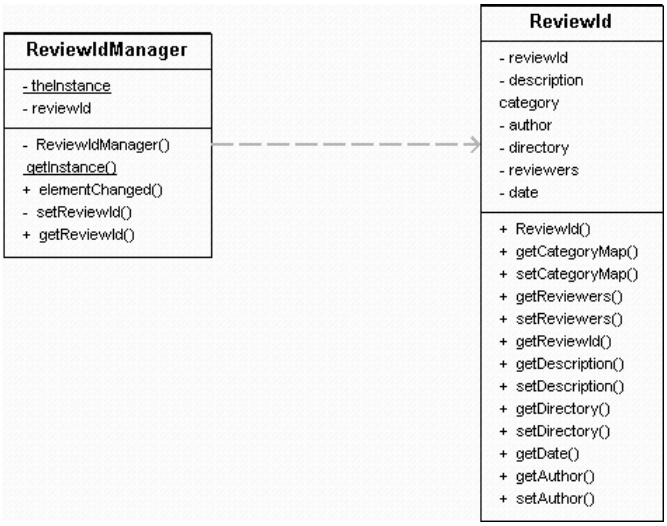



**Abbildung 9.2: ReviewId Klassen**

Die Methoden zum Auslesen und Speichern des Autors sind zwar noch in der Klasse „ReviewId“ vorhanden, werden aber nicht mehr genutzt. Die Klasse „ReviewIdManager“ verwaltet die Reviews und überprüft, ob Änderungen an den Reviewdaten vorgenommen wurden.

In der XML-Datei „property.xml“ sind die Defaultwerte gespeichert.

```
<?xml version="1.0" encoding="UTF-8"?>
<Property>
  <Review id="DEFAULT">
    <Description>property.default.description</Description>
    <Author/>
    <CreationDate format="yyyy-MM-dd :: HH:mm:ss:SSS z">1970-01-01 ::
00:00:00:000 GMT-10:00</CreationDate>
    <Directory>review</Directory>
    <Reviewers/>
    <Files/>
    <FieldItems>
      <FieldItem id="Type" default="item.label.unset">
        <Entry name="item.label.unset" />
        <Entry name="item.type.label.codingStandards" />
            ...
      </FieldItem>
      <FieldItem id="Severity" default="item.label.unset">
        <Entry name="item.label.unset" />
        <Entry name="item.severity.label.critical" />
              ...
        <Entry name="item.severity.label.trivial" />
      </FieldItem>
      <FieldItem id="Status" default="item.status.label.open">
        <Entry name="item.status.label.open" />
        <Entry name="item.status.label.rejected" />
        <Entry name="item.status.label.closed" />
      </FieldItem>
    </FieldItems>
  </Review>
</Property>
```

**Abbildung 9.3: Auszug aus property.xml**

Diese Datei wird zu Beginn in die „.jupiter“-Datei des Projektes kopiert. Im Vergleich zu der ursprünglichen Version werden hier die Filtereigenschaften nicht mehr aufgeführt. Die restlichen Werte werden übernommen. Das „Author“-Element ist zwar noch vorhanden, wird von der neuen Version jedoch nicht mehr genutzt.

### 9.1.2 Review-Issue-Daten

Die Review-Issue-Daten erfahren im Vergleich zu den allgemeinen Reviewdaten eine größere Änderung. Hier werden, wie bereits im vorherigen Kapitel erläutert, die Elemente „Assigned to“, „Annotation“, „Revision“ und „Resolution“ nicht mehr benötigt. Hinzu kommt das Element „Mantis Id“, dass nach dem Exportieren der Review-Issues nach Mantis automatisch belegt wird.

Für die Review-Issue-Daten sind vor allem das Paket „csdl.jupiter.file“, sowie das Paket „csdl.jupiter.model.reviewIssue“ von Bedeutung.

Im Paket „csdl.jupiter.model.reviewIssue“ sind besonders die Klassen „ReviewIssue“, „ReviewIssueModel“ und „ReviewIssueModelManager“ wichtig. Neben diesen drei Klassen gibt es noch diverse weitere Klassen, z. B. zur Verwaltung von Type, Severity und Status. Abbildung 9.4 zeigt jedoch nur diese drei wichtigsten Klassen.

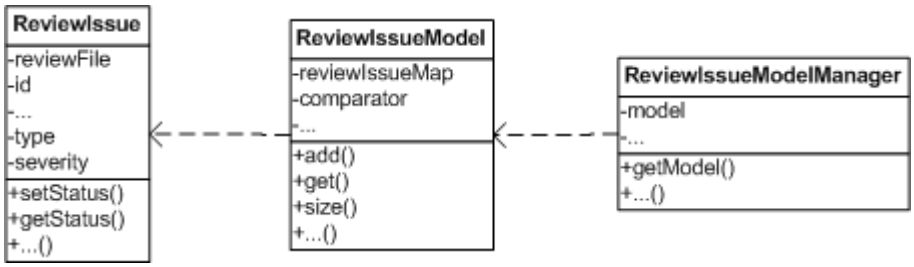



**Abbildung 9.4: Review-Issue-Klassen**

Die Instanzen der Klasse „ReviewIssue“ bilden die einzelnen Review-Issues mit ihren Eigenschaften ab. Die Klassen „ReviewIssueModel“ und „ReviewIssueModelManager“ verwalten die Review-Issues.

Die Klasse „ReviewIssueXmlSerializer“ im Paket „csdl.jupiter.file“ ist für die Speicherung der einzelnen Review-Issues in der „.review“-Datei verantwortlich. Dabei greift diese Klasse unter anderem auf die in Abbildung 9.4 abgebildeten Klassen, sowie die Klasse „ReviewId“ zu.

Pro Review und Reviewer existiert eine „.review“-Datei. Es ist eine XML-Datei mit einer Auflistung aller Review-Issues, die durch diesen Reviewer erzeugt wurden. Abbildung 9.5 zeigt exemplarisch eine „.review“-Datei.

```
<?xml version="1.0" encoding="UTF-8"?>
<Review id="4711">
  <ReviewIssue id="FEPPUZG2">
    <ReviewIssueMeta>
      <CreationDate format="yyyy-MM-dd :: HH:mm:ss:SSS z">2008-04-06 ::
16:35:04:562 CEST</CreationDate>
      <LastModificationDate format="yyyy-MM-dd :: HH:mm:ss:SSS z">2008-04-06 ::
16:35:04:562 CEST</LastModificationDate>
    </ReviewIssueMeta>
    <ReviewerId>neum_st</ReviewerId>
    <File line="4">src/SampleClass.java</File>
    <Type>item.type.label.codingStandards</Type>
    <Severity>item.severity.label.critical</Severity>
    <Summary>Test</Summary>
    <Description>System.out.println("Hello world!");</Description>
    <MantisId />
    <Status>item.status.label.open</Status>
  </ReviewIssue>
  <ReviewIssue id="FEPPV5FC">
    <ReviewIssueMeta>
      <CreationDate format="yyyy-MM-dd :: HH:mm:ss:SSS z">2008-04-06 ::
16:35:12:312 CEST</CreationDate>
      <LastModificationDate format="yyyy-MM-dd :: HH:mm:ss:SSS z">2008-04-06 ::
16:35:12:312 CEST</LastModificationDate>
    </ReviewIssueMeta>
    <ReviewerId>neum_st</ReviewerId>
    <File line="2">src/SampleClass.java</File>
    <Type>item.label.unset</Type>
    <Severity>item.label.unset</Severity>
    <Summary>Test2</Summary>
    <Description>public class SampleClass {</Description>
    <MantisId />
    <Status>item.status.label.open</Status>
  </ReviewIssue>
</Review>
```

**Abbildung 9.5: Reviewdatei „.review“**

## 9.2 Änderung der Reviewerverwaltung

Die Reviewerverwaltung erfährt zur ursprünglichen Version eine gravierende Änderung. Die Reviewer werden, wie bereits in der Konzeption beschrieben, in einer separaten Datei, der „.reviewer“-Datei gespeichert. Diese XML-Datei ist für das gesamte Projekt gültig. Dabei wird zu jedem Reviewer eine Reviewer-Id und der Name des Reviewers angegeben. Abbildung 9.6 zeigt ein Beispiel dieser Datei.

```
<?xml version="1.0" encoding="UTF-8"?>
<user>
  <reviewer id="neum_st">
    <name>Stefan Neumann</name>
  </reviewer>
  <reviewer id="tmesch">
    <name>Thijs Metsch</name>
  </reviewer>
</user>
```

**Abbildung 9.6: Reviewerdatei „.reviewer“**

Die Reviewerdatei wird über die Default-Review-Id, in den Projekteigenschaften gefüllt. Die dort eingegebenen Reviewer stehen dann für neue Reviews zur Auswahl zur Verfügung. Es besteht jedoch auch die Möglichkeit, während des Anlegens eines neuen Reviews neue Reviewer hinzuzufügen.

In der ursprünglichen Version konnten zwar auch Default-Reviewer in der Default-Review-Id definiert werden, diese waren dann aber für ein neues Review voreingestellt und mussten, falls sie nicht benötigt wurden, extra wieder aus der Liste der Reviewer entfernt werden. In der neuen Version können die Reviewer nun je nach Bedarf ausgewählt werden.

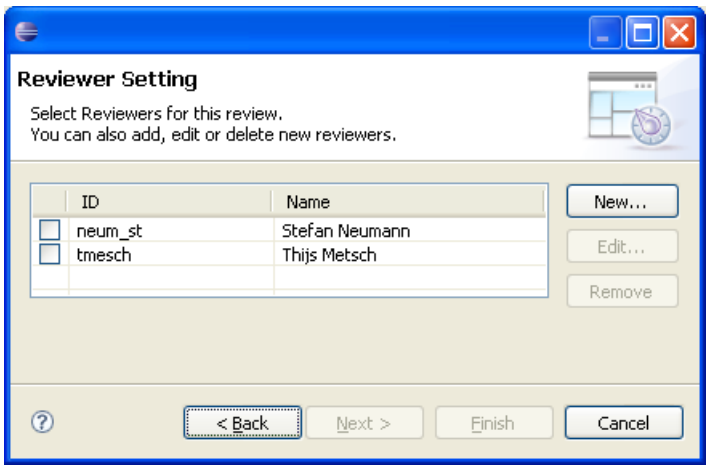



**Abbildung 9.7: Auswahl der Reviewer**

Werden die Reviewer eines Projektes auch für ein anderes Projekt benötigt, so kann diese Reviewerdatei einfach in den Projektordner des neuen Projektes kopiert werden. So müssen nicht für jedes neue Projekt die Reviewer neu angelegt werden.

## 9.3 Entfernen der Filter und der Phasen

Um Jupiter flexibler zu gestalten, wurden die drei Phasen „Individual Phase“, „Team Phase“ und „Rework Phase“ entfernt. In der neuen Version existiert nur eine Phase, die „Review Phase“. Aus diesem Grund, und da die neue Jupiterversion an Mantis angebunden ist, werden auch die Filter nicht mehr benötigt. Die Filterung für die Nachbereitung kann nun mithilfe der Mantis-Id durchgeführt werden.

Als Konsequenz des Entfernens der Filter und Phasen werden Änderungen an „preference.xml“ und „property.xml“ durchgeführt. Dort werden die Phasen durch die einige Phase „Review Phase“ ersetzt und die Filter vollständig entfernt.

```
<?xml version="1.0" encoding="UTF-8"?>
<Preference>
  <View default="phase.review">
    <Phase name="phase.review">
      <ColumnEntry name="columnHeader.label.linkIcon" width="19"
resizable="false" enable="true" />
      <ColumnEntry name="columnHeader.label.status" width="77" resizable="true"
enable="true" />
      <ColumnEntry name="columnHeader.label.type" width="68" resizable="true"
enable="true" />
      <ColumnEntry name="columnHeader.label.severity" width="55"
resizable="true" enable="true" />
      <ColumnEntry name="columnHeader.label.summary" width="250"
resizable="true" enable="true" />
      <ColumnEntry name="columnHeader.label.description" width="420"
resizable="true" enable="true" />
      <ColumnEntry name="columnHeader.label.modificationDate" width="70"
resizable="true" enable="false" />
      <ColumnEntry name="columnHeader.label.file" width="179" resizable="true"
enable="true" />
      <ColumnEntry name="columnHeader.label.line" width="40" resizable="true"
enable="true" />
      <ColumnEntry name="columnHeader.label.reviewer" width="70"
resizable="true" enable="true" />
      <ColumnEntry name="columnHeader.label.mantisId" width="70"
resizable="true" enable="true" />
      <ColumnEntry name="columnHeader.label.creationDate" width="70"
resizable="true" enable="false" />
      <ColumnEntry name="columnHeader.label.id" width="70" resizable="true"
enable="false" />
    </Phase>
  </View>
   ...
   ...
</Preference>
```

**Abbildung 9.8: Auszug aus „preference.xml“**

Des Weiteren werden an einer Vielzahl von Klassen kleinere Änderungen vorgenommen. Da die Filter und Phasen in den ursprünglichen Jupiterversion jedoch sehr tief und breit in den Sourcecode integriert sind, kann an dieser Stelle nicht auf alle Änderungen eingegangen werden. Für detailliertere Betrachtungen kann nur der Vergleich der Sourcecodes der beiden Jupiterversionen genutzt werden.

## 9.4 Anpassen der Wizards und Dialoge

Die Änderung der Reviewdaten und die Änderung der Reviewerverwaltung, sowie das Entfernen der Filter und Phasen zieht eine Anpassung einiger Wizards und Dialoge nach sich. Dies betrifft vor allem die Wizards und Dialoge zum Erzeugen und Editieren von Reviews. Neu hinzugefügt sind die Dialoge zum Anlegen und Editieren von Reviewern und zum Anlegen und Editieren von Review-Issues.

### 9.4.1 Review-Wizards und Dialoge

Im Wizard zum Erzeugen eines neuen Reviews, sowie im Dialog zum Editieren eines bereits bestehenden Reviews werden diverse Änderungen vorgenommen. Da beide äquivalent zueinander sind, beziehen sich die folgenden Ausführungen sowohl auf den Wizard, als auch auf den Dialog.

Wie bereits im vorherigen Kapitel beschrieben, werden die Angabe eines Autors entfernt. Somit wird auch diese Seite aus dem Wizard entfernt. Ebenso wird die Seite zur Konfiguration der Filter aus dem Wizard entfernt. Die Angabe eines Speicherortes für die Review-Issues ist in der neuen Jupiterversion nur noch in der Default-Review-Id unter den Projekteigenschaften möglich, damit nicht jeder Benutzer, der ein neues Review anlegt, den Speicherort frei wählt.

Alle anderen Seiten werden von der alten Jupiterversion übernommen.

### 9.4.2 Reviewer-Dialoge

Durch die neue Reviewerverwaltung werden zwei neue Dialoge benötigt. Zum einen ein Dialog zum Anlegen eines neuen Reviewers und ein Dialog zum Editieren eines bereits bestehenden Reviewers.

Beim Anlegen eines neuen Reviewers müssen eine Reviewer-Id, sowie der Name des Reviewers angegeben werden. Keines der Felder darf leer bleiben. Beim Editieren eines Reviewers kann nur der Name editiert werden. Somit ist sichergestellt, dass keine anderen Reviewer überschrieben werden.

### 9.4.3 Review-Issue-Dialoge

Durch das Entfernen des „Review Editor“-Views muss diese Funktionalität durch Dialoge übernommen werden. Hierzu werden zwei analoge Dialoge hinzugefügt, der „ReviewIssueAddDialog“ und der „ReviewIssueEditDialog“. Beide beinhalten drei Combo-Boxen, zum Einstellen von Type, Severity und Status und zwei Textfelder für Summary und Description. Der Aufruf des „ReviewIssueAddDialog“ erfolgt über das Kontextmenü des Editors („Add Review Issue“) oder über das Menü des „Review Issue Tables“. Der Aufruf des „ReviewIssueEditDialog“ erfolgt nach Markierung des Review-Issues im „Review Issue Table“ und anschließender Auswahl des Menüpunktes „Edit“.

### 9.4.4 Weitere Wizards und Dialoge

Neben den oben erwähnten Wizards und Dialogen werden auch für die Mantis-Anbindung, für das Erzeugen eines Reviewreports sowie für die Reviewmetriken Wizards und Dialoge erstellt. Diese werden jedoch in den entsprechenden Abschnitten noch näher betrachtet.

## 9.5 Anpassen des Review Issue Table

Der „Review Issue Table“ ist eines der zentralen GUI-Elemente der Jupitererweiterung. Dort werden alle Review-Issues aufgelistet. Im Vergleich zu der ursprünglichen Jupiterversion werden dort einige Änderungen vorgenommen.

Da sich der „Review Issue Table“ in der Jupitererweiterung über den gesamten unteren Bereich erstreckt, können mehr Review-Issue-Informationen aufgelistet werden. Diese sind: Status, Type, Severity, Summary, Description, File, Line, Reviewer und Mantis-Id.

Durch das Entfernen der Filter entfällt der Menüeintrag zur Konfiguration der Filter. Ebenso entfällt der Menüeintrag zur Auswahl der Phasen. Hinzugefügt wird der Menüeintrag „Done“, der den Status des Review-Issues automatisch auf „Closed“ setzt.

In der Jupitererweiterung werden nun in der Titelzeile statt der Phase, das Projekt, das aktive Review und der Reviewer angezeigt, damit der Reviewer zu jeder Zeit sehen kann, welches Review gerade aktiv ist.

Die entsprechenden Klassen befinden sich im Paket „csdl.jupiter.ui.view.table“.

Abbildung 9.9 zeigt einen Auszug dieses Pakets. Dabei sind hier nur die drei wichtigsten Klassen aufgelistet. Die Klasse „ReviewTableView“ zum Erstellen des eigentlichen Views, sowie die Klasse „TableViewPart“ zur Konfiguration der Toolbar und der Menüeinträge. In der Klasse „ReviewTableViewAction“ werden die Aktionen definiert, die bei Auswahl der entsprechenden Einträge in der Toolbar oder dem Kontextmenü des „Review Issue Table“ ausgeführt werden.

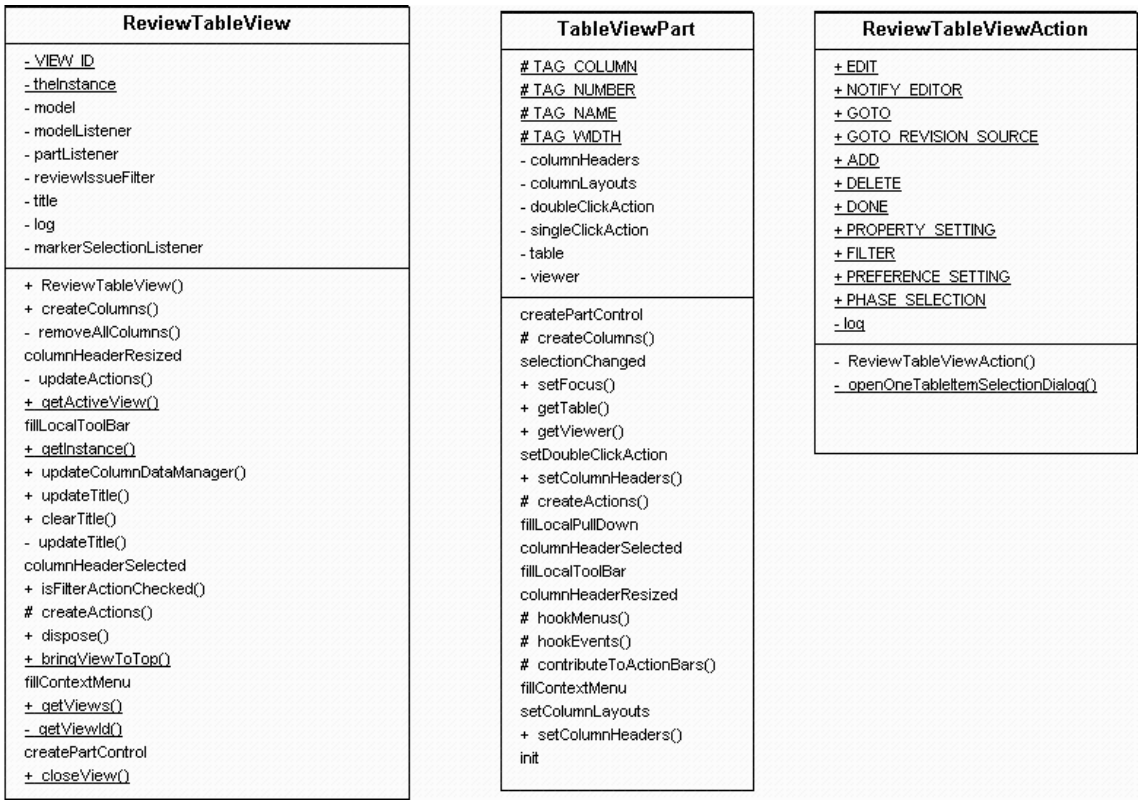



**Abbildung 9.9: Auszug aus Paket „csdl.jupiter.ui.view.table“**

## 9.6 Hinzufügen der Mantis-Anbindung

Die Mantis-Anbindung ist eine der Kernerweiterungen. Sie erstreckt sich auf drei Bereiche. Die Erweiterung der Klasse „ReviewIssue“ um das Attribut „mantisId“ und die

entsprechenden Methoden, das Anlegen einer PreferencePage zur Konfiguration der Mantis-Verbindungen und das Erstellen eines Export-Wizards.

Bis auf die Erweiterung der Klasse „ReviewIssue“ um die Mantis-Id greift die Mantis-Anbindung nicht in das bestehende Programm ein. D. h. auch die ursprüngliche Jupiter Version kann so einfach um diese Funktionalität erweitert werden.

### 9.6.1 Erweiterung der Klasse ReviewIssue

Die Klasse „ReviewIssue“ wird nur um das Attribut „mantisId“ vom Typ String und die Methode „getMantisId()“ mit dem Rückgabewert String erweitert. Der Konstruktor der Klasse wird entsprechend angepasst.

### 9.6.2 Anlegen einer Mantis-PreferencePage

Da das Jupiter-Plugin seine Konfiguration in der Datei „preference.xml“ in den Metadaten des Eclipse-Workspaces abspeichert, muss diese bereits in Abbildung 9.8 in Auszügen gezeigte Datei entsprechend erweitert werden. Abbildung 9.10 zeigt diese Erweiterung.

```
<?xml version="1.0" encoding="UTF-8"?>
<Preference>
  ...
  ...
  <Mantis>
    <Connection name="Mantis Demo"
url="http://www.futureware.biz/mantisdemo/api/soap/mantisconnect.php" user=""
password="" save="true" />
  </Mantis>
</Preference>
```

**Abbildung 9.10: Auszug aus „preference.xml“**

Jede Mantis-Verbindung besteht aus einem Namen, einer URL, einem Benutzernamen, einem Passwort und der Information, ob das Passwort gespeichert wurde.

Diese Informationen müssen nun in einer PreferencePage konfiguriert werden können. Die PreferencePage wird mittels der Klasse „MantisPreferencePage“ im Paket „csdl.jupiter.ui.preference“ erstellt. Diese PreferencePage listet alle vorhandenen Verbindungen auf und ermöglicht das Hinzufügen, Löschen und Editieren von Verbindungen. Zum Hinzufügen und Editieren der Verbindungen werden die Dialoge „MantisConnectionAddDialog“ und „MantisConnectionEditDialog“ aus dem Paket „csdl.jupiter.ui.dialogs“ genutzt.

Des Weiteren wird eine neue Klasse „MantisConnection“ erstellt. Diese Klasse beinhaltet die oben bereits erwähnten Attribute, sowie die entsprechenden „getter“ und „setter“ Methoden.

Zur Speicherung der Verbindungen im der Datei „preference.xml“ werden, wie bereits bei den Review-Issues und den Properties, Klassen aus dem Paket „csdl.jupiter.file“ genutzt.

### 9.6.3 Zugriff auf Mantis

Der Zugriff auf Mantis erfolgt über die „MantisConnect-Client-API“. Diese API (Application Programming Interface) stammt aus dem Eclipse-Plugin „MantisConnect“ [URL:MantisConnect]. Sie ist frei verfügbar und wurde aus der WSDL-Datei (Web Services Description Language)der Mantis Version 1.1.1 generiert. Diese WSDL-Datei beschreibt die Schnittstelle von Mantis.

Die „MantisConnect-Client-API“ besitzt eine Reihe von Methoden zum Zugriff auf Mantis. Zum einen zum Herstellen einer Verbindung und zum anderen zum Zugriff auf Projekte und Mantis-Issues. Dabei können bestehende Mantis-Issues bearbeitet, sowie Neue erstellt werden.

Für diese Anwendung werden nur die folgenden Methoden benötigt:

- Herstellen einer Verbindung zur Mantis-Installation und deren Projekte
- Abfrage der „FieldItems“, der Einträge für Kategorie, Priorität usw.
- Hinzufügen eines Mantis-Issues

Der Zugriff auf Mantis und die Abfrage alle Projekte dieser Mantis-Installation zeigt das folgende Beispiel:

```
mantisSession = new MCSession(url, mantisConnection.getUser(),
mantisConnection.getPwd());
                mantisProjects =
mantisSession.getAccessibleProjects();
```

**Abbildung 9.11: Zugriff auf Mantis und Abfrage der Projekte**

Eine „MantisConnect Session“ wird über die Angabe einer URL, eines Benutzernamens und der Angabe eines Passwortes hergestellt. Mithilfe der Methode „getAccessibleProjects()“ werden alle Projekte der entsprechenden Mantis-Verbindung zurückgegeben.

Die Abfrage der „FieldItems“, zeigt das folgende Beispiel:

```
category =
mantisSession.getCategories(selectedMantisProject.getId());
...
IMCAttribute[] priorities =
mantisSession.getEnum(Enumeration.PRIORITIES);
```

**Abbildung 9.12: Abfrage der FieldItems von Mantis**

Über die Angabe eines Mantisprojekts, bzw. dessen Id werden ein Teil der „FieldItems“ abgefragt. Die vom Projekt unabhängigen „FieldItems“ können ohne eine Projekt-Id abgefragt werden.

Das Hinzufügen eines Mantis-Issues zeigt das folgende Beispiel:

```
IIssue issue = (IIssue) iterator.next();
long id = session.addIssue(issue);
```

**Abbildung 9.13: Hinzufügen eines Mantis-Issues**

Dieser Ausschnitt zeigt einen Teil einer Iteration, bei der eine Reihe von Issues erzeugt werden. Beim Erzeugen eines Mantis-Issues wird dessen Id zurückgegeben.

Die „MantisConnect-API“ bietet noch weitere Möglichkeiten, die im Rahmen der Anwendung jedoch nicht benötigt werden.

### 9.6.4 Erstellen eines Mantis-Export-Wizards

Der Mantis-Export-Wizard ist das Herzstück der Mantis-Anbindung. Er beinhaltet zwei WizardPages. Auf der ersten Seite werden die Mantisverbindung, das Mantisprojekt, das Eclipseprojekt, sowie das Review für den Export ausgewählt. Auf der zweiten Seite werden die Review-Issues gruppiert und zu Mantis-Issues zusammengefasst.

Die erste WizardPage dient dabei nur zur Herstellung der Verbindung und zur Abfrage der „ItemEntries“. Dieser Vorgang nimmt einige Zeit in Anspruch, muss aber nur einmal zu Beginn des Exportierens durchgeführt werden. Alle weiteren Transaktionen zwischen Jupiter und Mantis können im Anschluss ohne merkliche Zeitverzögerung durchgeführt werden.

Auf der zweiten WizardPage werden die Review-Issues durch Mehrfachauswahl und Gruppierung zu Mantis-Issues zusammengefasst. Dabei werden die Review-Issue-Informationen in das Description-Feld des Mantis-Issues übertragen. Alle anderen Einstellungen wie „Category“, „Reproducibility“, „Severity“, „Priority“, „Summary“ und „Additional Information“ können anschließend frei konfiguriert werden.

Zur Erzeugung der Mantis-Issues wird eine Hilfsklasse „MantisIssue“ benutzt. Die Mantis-Issues vom Typ „MantisIssue“ werden dann bei Fertigstellung des Wizards in Issues vom Typ „IIssue“ umgewandelt. Diese Hilfsklasse ist notwendig, da das Instantiieren von „IIssue“ direkt einen neuen Mantis-Issue im Bugtrackingsystem erzeugt. Ferner soll die Möglichkeit bestehen Mantis-Issues vor Beenden des Wizards auch wieder zurück in Review-Issues zu wandeln. Dazu hat die Klasse „MantisIssue“ als eines ihrer Attribute eine Liste von Review-Issues. Abbildung 9.14 zeigt diese Hilfsklasse. Dabei sind die Methoden nur exemplarisch angedeutet.

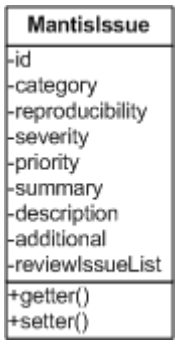



**Abbildung 9.14: Hilfsklasse MantisIssue**

Beim Löschen eines Mantis-Issues können so die zu einem Mantis-Issue zusammengefassten Review-Issues wieder hergestellt werden. Ebenso soll es möglich sein, Mantis-Issues noch vor ihrem eigentlichen Erzeugen im Bugtrackingsystem zu editieren.

Zum Abschluss des Exports wird eine Liste von Mantis-Issues des Typs „MantisIssue“ über das in Abbildung 9.13 gezeigte Verfahren in echte Mantis-Issues umgewandelt und in das Bugtrackingsystem eingetragen. Durch die Rückgabe der Mantis-Id und der Liste der Review-Issues als Attribute der Klasse „MantisIssue“ kann so auch den Review-Issues eine Mantis-Id zugewiesen werden.

## 9.7 Hinzufügen des Reviewreport

Der Reviewreport enthält die in Anforderung D40 geforderten Daten. Tabelle 9.1 listet diese Daten hier noch einmal auf.

| Nr. | Anforderung |
|---|---|
| D40 | Ein Reviewreport enthält folgende Daten: Alle Daten des Reviews (Name, Beschreibung, Reviewer), das Eclipseprojekt, das aktuelle Tagesdatum, zu jedem Issue File, Zeilennummer, Summary, Severity und Type + optionale weitere Anmerkungen. |

**Tabelle 9.1: Daten des Reviewreports**

Im Reviewreport-Export-Wizard hat der Benutzer dann die Möglichkeit diese Daten einzugeben. Die Daten der Review-Issues werden über die Auswahl des Eclipseprojektes und des entsprechenden Reviews ausgewählt. Es werden alle Issues dieses Reviews tabellarisch im Reviewreport aufgelistet. Der Benutzer hat anschließend die Auswahl zwischen den Formaten PDF und HTML. Ist diese Auswahl getroffen, kann über einen Dialog der Pfad ausgewählt werden.

Das fertige Dokument besteht aus 2 Tabellen (den allgemeinen Reviewinformationen und der Auflistung der Review-Issues) und den zusätzlich eingegebenen Informationen zum Review. Als zusätzliche Informationen können dort z. B. allgemeine Anmerkungen zum Ablauf des Reviews gemacht werden.

### 9.7.1 Reviewreport im PDF-Format

Das Generieren eines PDF-Dokuments erfolgt mithilfe der Java-Bibliothek „iText-2.0.8.jar“, des freien I-Text-Projekts des Belgiers Bruno Lowagie [URL:Itext]. Diese Bibliothek erlaubt eine Erstellung von PDF-Dokumenten direkt aus Java heraus.

Der folgende Codeausschnitt zeigt das Erzeugen eines PDF-Dokuments mit Hilfe von I-Text:

```
Document document = new Document(PageSize.A4);
        try {
            PdfWriter.getInstance(document, new
              FileOutputStream(destinationText.getText()));
            document.open();
            ...
              ...
            document.close();
        } catch (FileNotFoundException e) {
            log.error(e);
        } catch (DocumentException e) {
            log.error(e);
        }
```

**Abbildung 9.15: Erzeugen eines PDF-Dokuments mit I-Text**

Dazu muss eine Instanz des „PdfWriters“ erzeugt werden. Diese wird dann mit dem „FileOutputStream“ unter der Angabe eines Pfades verknüpft. Anschließend wird das Dokument geöffnet und kann nun erstellt werden. Zur Fertigstellung wird das Dokument geschlossen. Dem Dokument können Kopf- und Fußzeilen, Paragrafen, Tabellen, Grafiken und vieles mehr hinzugefügt werden.

Im Review-Report-Export-Wizard werden nach Auswahl des Eclipseprojektes und des Reviews alle in Tabelle 9.1 aufgeführten Informationen zum PDF-Dokument hinzugefügt.

Über die angegebene Review-Id kann auf die allgemeinen Informationen des Reviews zugegriffen werden. Über die Klasse „ReviewIssueModel“ kann mithilfe des Eclipseprojektes und der Review-Id auf alle Review-Issues dieses Projektes zugegriffen werden.

Abbildung 9.16 zeigt an dieser Stelle einmal beispielhaft den Zugriff auf die Review Issues. Die Methode gilt auch für das Erzeugen eine Reviewreports im HTML-Format.

```
/**
 * Fills the review issue list.
 */
private void fillReviewIssueList() {
      ReviewIssueModelManager reviewIssueModelManager =
      ReviewIssueModelManager.getInstance();
      ReviewIssueModel reviewIssueModel =
            reviewIssueModelManager.getModel(this.project, reviewId);
      Object[] list = reviewIssueModel.getElements(this.project);
      reviewIssueList = new ArrayList<ReviewIssue>();
      for (int i = 0; i < list.length; i++) {
          reviewIssueList.add((ReviewIssue) list[i]);
      }
}
```

**Abbildung 9.16: Zugriff auf alle Issues eines Reviews**

Die folgende Abbildung zeigt ein Beispiel eines PDF-Reviewreports.

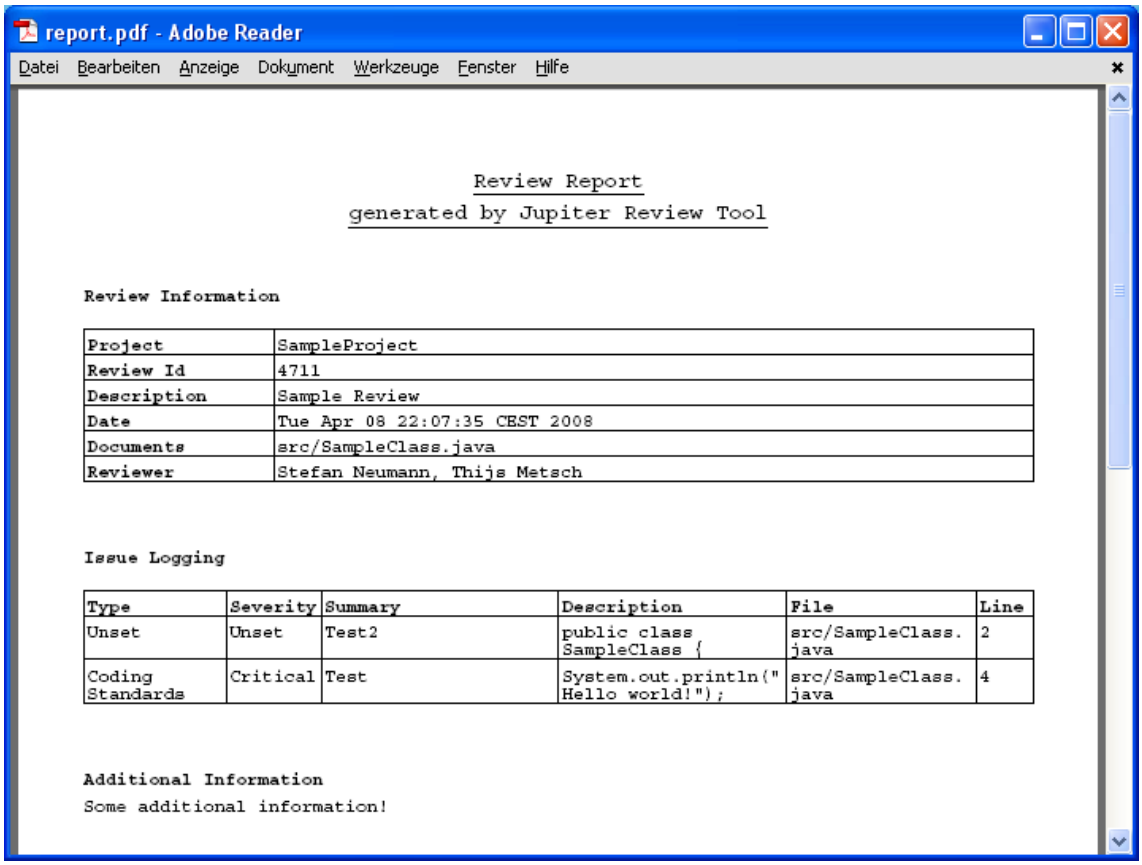



**Abbildung 9.17: Reviewreport im PDF-Format**

### 9.7.2 Reviewreport im HTML-Format

Das Erstellen eines Reviewreports im HTML-Format benötigt keine zusätzliche Java-Bibliotheken. Das Erzeugen des HTML-Dokuments erfolgt mittels eines Java-"FileWriters". Unter Angabe der entsprechenden HTML-Tags und der entsprechenden Dateiendung erstellt der „FileWriter" das HTML-Dokument. Der Aufbau ist analog zum PDF-Report.

Die folgende Abbildung zeigt ein Beispiel eines HTML-Reviewreports.

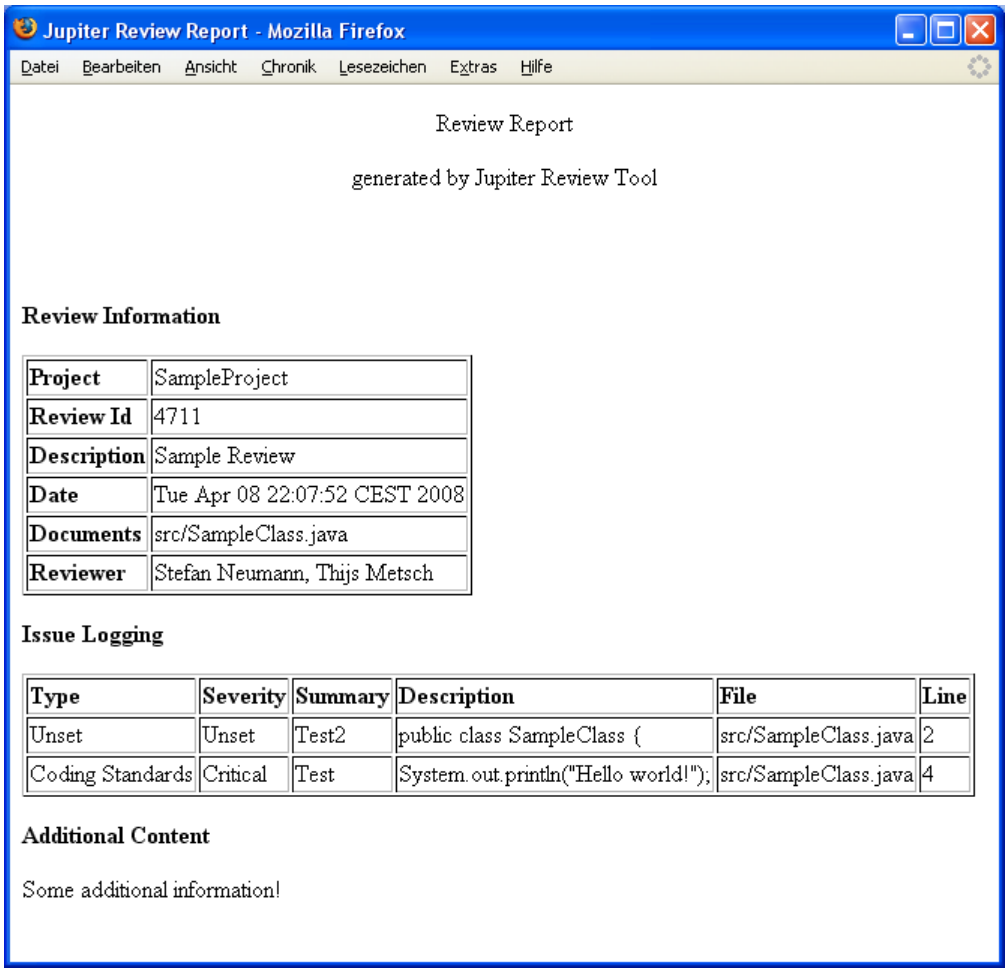



**Abbildung 9.18: Reviewreport im HTML-Format**

## 9.8 Hinzufügen der Reviewmetriken

Das Hinzufügen von Reviewmetriken ist eine Funktionalität, die anstelle einer echten Reviewüberdeckungsmessung implementiert wird. Die Reviewmetriken werden als Dialog angezeigt. Gestartet wird dieser Dialog über den Jupiter-Pulldown-Button. Die Reviewmetriken bilden die Verzeichnisstruktur eines Projektes ab und listen dort Informationen zu eventuellen Reviews auf. Dabei wird zwischen Ordnern und Dateien unterschieden. Bei Ordnern wird angegeben, wie viele der in ihm befindlichen Dateien bereits einem Review unterzogen wurden. Bei Dateien wird die Anzahl der Review-Issues und die Anzahl der Zeilen angegeben. Wurde eine Datei noch keinem Review unterzogen, so wird die Information „not reviewed“ angezeigt.

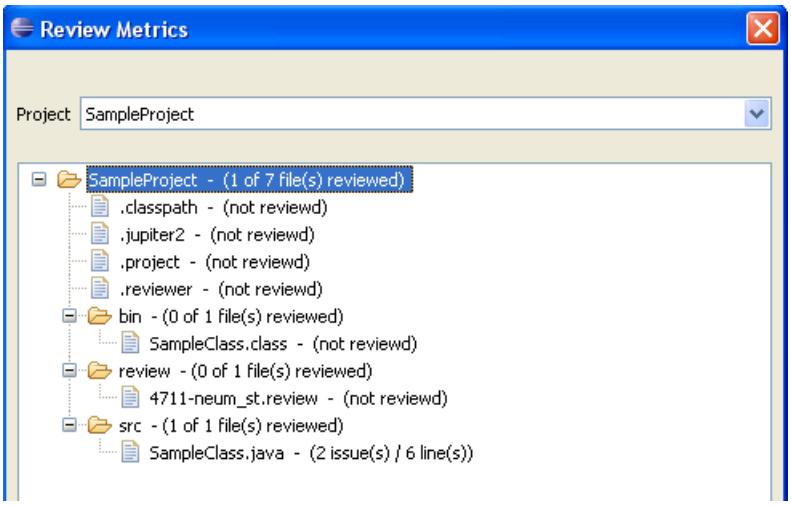



**Abbildung 9.19: Review Metriken Dialog**

Die zuständige Klasse „ReviewMetricsDialog“ baut den Verzeichnisbaum nach Auswahl eines Projektes rekursiv auf. Dabei greift die Klasse auf die in Kapitel 5.2.4 vorgestellten Ressourcen zu. Der Baum wird als „Java-Tree“ implementiert. Die folgende Abbildung zeigt die Klasse „ReviewMetricsDialog“.

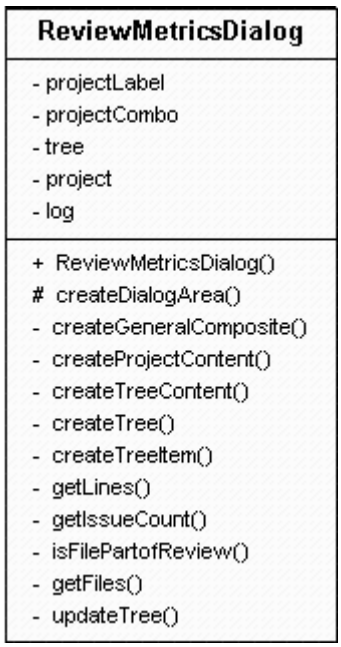



**Abbildung 9.20: Klasse ReviewMetricsDialog**

Dabei implementiert diese Klasse neben den Methoden zum Erstellen der Benutzeroberfläche diverse Methoden zum Zugriff auf die Ressourcen und zum Erzeugen der Baumstruktur. Die Methode „getLines()“ gibt die Anzahl der Zeilen einer Datei zurück, die Methode „getIssueCount()“ liefert die Anzahl der Issues in einer Datei. Diese Informationen werden dann zu einer Datei angezeigt. Die Methode „isFilePartofReview()“ gibt zurück, ob eine Datei bereits einem Review unterzogen wurde. Die Ressourcen

des Projekts werden rekursiv durchgearbeitet und am Ende wird die Baumansicht erzeugt. Dieser Vorgang nimmt bei sehr großen Projekten einen längeren Zeitraum ein.

## 9.9 Allgemeiner Adapter zur Anbindung externer Tools

Da der Zugriff auf die Reviewdaten und die Review-Issue-Daten in der ursprünglichen Jupiterversion über verschiedene Klassen geregelt wird, erhält die Jupitererweiterung eine neue Klasse „JupiterResource“ als allgemeinen Adapter zur Anbindung externer Tools, wie anderen Bugtrackingsystemen oder Fehlerdatenbanken. Über die neue Klasse erfolgt der Zugriff auf alle relevanten Reviewdaten.

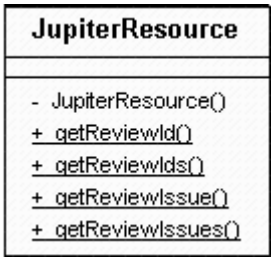



**Abbildung 9.21: Die Klasse JupiterResource als allgemeiner Adapter**

„JupiterResource“ bietet zwei Methoden zum Zugriff auf Reviews und zwei Methoden zum Zugriff auf Review-Issues. Unter Angabe des Reviewnamens und des Projekts erhält man mit der Methode „getReviewId()“ Zugriff auf ein bestimmtes Review. Unter der Angabe des Projekts wird mit „getReviewIds()“ eine Liste aller Reviews eines Projektes zurückgegeben.

Analog verhält es sich bei den Methoden zum Zugriff auf Review-Issues. Dort kann unter Angabe des Projekts, des Reviews und der Review-Issue-Id mit der Methode „getReviewIssue()“ auf einen bestimmten Review-Issue zugegriffen werden. Eine Liste aller Review-Issues eines Reviews erhält man mit der Methode „getReviewIssues()“. Hierzu müssen als Parameter das Projekt und das Review angegeben werden. Der Zugriff auf alle relevanten Informationen erfolgt so über diese vier Methoden, ohne tiefer in die Struktur des Tools einsteigen zu müssen.

## 9.10 Hinzufügen einer Default-Checkliste

Die Jupitererweiterung stellt dem Benutzer eine Default-Checkliste als CheatSheet zur Verfügung. Dazu wird der Extension Point „org.eclipse.ui.cheatsheets“ implementiert. Dort wird eine Kategorie angegeben unter der das Cheatsheet unter Help -> Cheat Sheets erreichbar sein soll. Diese Kategorie lautet wie in allen anderen Jupiter-Menüpunkten „Review“. Für das eigentliche Cheatsheet muss lediglich eine XML-Datei referenziert werden, in der die Elemente des CheatSheets definiert sind.

# 10 Resümee

Im Rahmen dieser Arbeit wurde ein Reviewprozess für die Einrichtung Simulations- und Softwaretechnik im Deutschen Zentrum für Luft- und Raumfahrt entwickelt, der den verschiedenen Anforderungen an Reviews in der Einrichtung gerecht wird. Er beinhaltet drei Reviewprozesse, die von sehr formell bis informell reichen. Je nach Kritikalität des zu prüfenden Sourcecodes oder anderer Dokumente kann einer der drei Prozesse zur Durchführung des Reviews gewählt werden. Die bestehenden Spezifikationen in der Einrichtung SISTEC beschränkten sich auf den IEEE Std. 1028 – 1997, der jedoch für die Durchführung von Reviews nicht genutzt wurde, da er keine praktische Anleitung für die Durchführung von Reviews bildet. Somit musste die Spezifikation des neuen Reviewprozesses hauptsächlich durch Gespräche mit Mitarbeitern und weiterer Literatur erarbeitet werden.

Das Tool Jupiter wurde an den neuen Reviewprozess angepasst und in die bestehende QS-Werkzeugpalette integriert. Dazu wurde Jupiter um eine Mantis-Anbindung erweitert, die es ermöglicht, die Reviewinformationen in das bestehende Bugtrackingsystem Mantis zu exportieren. Dazu können die Review-Issues beliebig gruppiert werden. Des Weiteren wurde Jupiter eine Reviewreport-Exportfunktion hinzugefügt, mit der alle Reviewinformationen im PDF- oder HTML-Format abgespeichert werden können. Abschließend wurde Jupiter um einen Dialog zur Darstellung der Reviewmetriken erweitert. Diese Metriken erlauben dem Benutzer eine Übersicht, welche der Dateien eines Projektes bereits einem Review unterzogen wurden und wie viele Anmerkungen pro Zeile dort gemacht wurden.

Die Umsetzung der Aufgabenstellung ist insgesamt zufriedenstellend gelaufen. Problematisch war zu Beginn die Abschätzung, wie lange die Arbeiten an bestimmten Problemstellungen benötigen würden. Dabei gestalteten sich die Einarbeitung in die Plugin-Entwicklung mit Eclipse und die Einarbeitung in Jupiter als besonders zeitaufwendig und schwierig.

Auch die Anforderungsanalyse gestaltete sich schwierig, da es immer wieder neue Ideen zur Umsetzung der Jupitererweiterung gab. Zum Schluss musste ich mich dann auf die wesentlichen Dinge konzentrieren, um im gesetzten Zeitrahmen die geforderten Features zu implementieren.

Nach der schwierigen Einarbeitung konnte ich dann die Anforderungen insbesondere die Anbindung an das Bugtrackingsystem Mantis gut umsetzen. Dort ist besonders die Gruppierung der einzelnen Review-Issues zu Mantis-Issues hervorzuheben. Da jedem Review-Issue so auch eine Mantis-Id zugewiesen wird, wird die Zuweisung der Verantwortlichkeiten und die anschließende Behebung der Anomalien stark vereinfacht.

Da am Ende die Zeit nicht mehr ausreichte, konnten einige weitere Ideen nicht realisiert werden. Eine detaillierte Zusammenfassung der durchgeführten und nicht-durchgeführten Pflicht- und Küraufgaben bietet der nächste Abschnitt. Einen daraus resultierenden Ausblick liefert Kapitel 11.

## 10.1 Zusammenfassung

Die folgenden Pflichtaufgaben wurden erfüllt:

1. Es wurde drei Reviewprozesse für die Einrichtung Simulations- und Softwaretechnik definiert, die von informell bis sehr formell reichen.
2. Jupiter wurde so angepasst, dass alle drei Reviewprozesse von ihm abgebildet werden können.
3. Jupiter wurde so erweitert, dass die Review-Issues in das Bugtrackingsystem Mantis exportiert werden können. Dabei können die Review-Issues beliebig gruppiert werden. Nach dem Export nach Mantis erhält jeder Review-Issue eine Mantis-Id, mit der die Nachbereitung des Reviews erheblich vereinfacht wird.

Zusätzlich wurden die folgenden Küraufgaben erfüllt:

1. Jupiter wurde um eine Reviewreport-Exportfunktion erweitert. Nach Abschluss eines Reviews kann so ein Report erstellt werden, der alle Reviewinformationen enthält. Der Report kann wahlweise als PDF- oder HTML-Dokument erzeugt werden.
2. Jupiter kann nun Reviewmetriken anzeigen. D. h., zu jedem Projekt und jedem Verzeichnis eines Projektes wird angezeigt, wie viele der beinhalteten Dateien bereits einem Review unterzogen wurden. War eine Datei bereits Teil eines Reviews, so wird die Anzahl der Review-Issues pro Zeile angezeigt.
3. Es wurde ein allgemeiner Adapter erstellt, der einfache Methoden bereitstellt, um auf Reviewinformationen zuzugreifen. So wird bspw. die Erweiterung von Jupiter um eine Fehlerdatenbank erleichtert.

Folgende Küraufgaben konnten nicht umgesetzt werden:

1. Die Vorschaltung von Metriken zur Auswahl des Reviewprozesses
2. Die Reviewüberdeckungsmessung

# 11 Ausblick

Aus dem vorherigen Abschnitt ergeben sich auch die Folgearbeiten an Jupiter. Die Vorschaltung von Metriken zur Auswahl des Reviewprozesses und die Reviewüberdeckungsmessung sind Features, die in der bisherigen Jupitererweiterung aus Zeitmangel noch nicht umgesetzt werden konnten.

Besonders die Reviewüberdeckungsmessung ist ein Feature, dass so bisher noch in keinem Reviewtool implementiert wurde. Dabei stellt sich vor allem das Problem, wie man dieses Feature handhabbar macht. Codeabschnitte müssen als überprüft markiert werden können, um dann daraus eine Abdeckung berechnen zu können. Diese Funktionalität kann dem Projektleiter einen guten Überblick über den Umfang der Reviews geben.

Die Vorschaltung von Reviewmetriken könnte den Jupiternutzern eine Hilfestellung zur Auswahl des Reviewprozesses geben. Diese Metriken könnten bspw. die strukturelle Komplexität des zu überprüfenden Sourcecodes nach der McCabe-Metrik, die sich aus den Kanten und Knoten des Kontrollflussgraphen des Programms berechnet, beinhalten. Ferner könnten die vor geschalteten Metriken Aussagen zur Verbundenheit von Klassen und Paketen (Kopplung) oder zur Vererbungshierarchie des Programms machen.

Neben dem Vorschalten von Metriken zur Auswahl des Reviewprozesses könnte Jupiter auch um detaillierte Metriken zu durchgeführten Reviews erweitert werden. Diese Metriken sollten Informationen über den ausgewählten Reviewprozess, die Anzahl der Gutachter, die Dauer des Reviews, die Anzahl überprüfter Zeilen Sourcecode und die Anzahl und Schwere der gefundenen Anomalien beinhalten. Durch diese Daten können Aufwandsschätzungen für zukünftige Reviews erstellt werden und die Güte des durchgeführten Reviews beurteilt werden.

Ebenfalls zu überprüfen ist die Nutzung verschlüsselter Verbindungen zu Mantis im DLR. Dieses Feature funktioniert in der Jupitererweiterung bisher noch nicht. Es steht zwar ein Zertifikat zur Verfügung, das auch bei der Nutzung des Mantis-Plugins „MantisConnect“ genutzt wird. Nach der Umstellung zur Mantis Version 1.1.1 kann jedoch keine Verbindung über HTTPS hergestellt werden. Weder über die Jupitererweiterung noch über das „MantisConnect“.

## Literaturverzeichnis


[DAUM05] Daum, Berthold; Java Entwicklung mit Eclipse 3.1; 2005

[FREEDMAN90] Freedman, Daniel P.; Weinberg Gerald M.; Walkthroughs, Inspections, and Technical Reviews; Third Edition; 1990

[GILB93] Gilb, Tom; Graham, Dorothy; Software Inspection; Addison-Wesley; 1993

[IEEE1028-1997] IEEE; IEEE Standard for Software Reviews; 1998

[ISO9126] o.V.; Software Engineering - Product Quality; 2001

[ISTQB05] o.V.; Certified Tester Foundation Level SyllabusGerman Testing Board e. V. & Swiss Testing Board; 2005

[JAVAINSEL6] Ullenboom, Christian; Java ist auch eine Insel; 2006

[Jupiter06] Takuya Yamashita; Evaluation of Jupiter:A lightweight Code Review Framework; 2006

[SESIS07] Metsch, Thijs; Anwendung von Software Engineering bei der SESIS Entwicklung; 2007

[URL:CC] http://www.smartbearsoftware.com/codecollab-overview.php?gclid=CJDYsMyZuZECFSgfaAod1ANgDQ; Aufruf: 10.02.2008

[URL:CS] http://codestriker.sourceforge.net; Aufruf: 09.02.2008

[URL:DLR] http://www.dlr.de/desktopdefault.aspx/tabid-636//1065_read-1465; Aufruf: 09.02.2008

[URL:Eclipse] http://www.eclipse.org; Aufruf: 16.04.2008

[URL:EclipseProAudit] http://www.instantiations.com/eclipsepro/audit.html; Aufruf: 21.03.2008

[URL:Itext] http://www.lowagie.com/iText; Aufruf: 08.04.2008

[URL:Jupiter] http://csdl.ics.hawaii.edu/Tools/Jupiter; Aufruf: 09.02.2008

[URL:JupiterUG] http://csdl.ics.hawaii.edu/Tools/Jupiter/Core/doc/UsersGuide.html; Aufruf: 10.02.2008

[URL:MantisConnect] http://www.futureware.biz/mantisconnect; Aufruf: 08.04.2008

[URL:Mariner] http://www.t-online.ch/c/96/13/20/9613204,si=0.html; Aufruf: 06.02.2008

[URL:OSGI] http://www.osgi.org/Main/HomePage; Aufruf: 16.04.2008

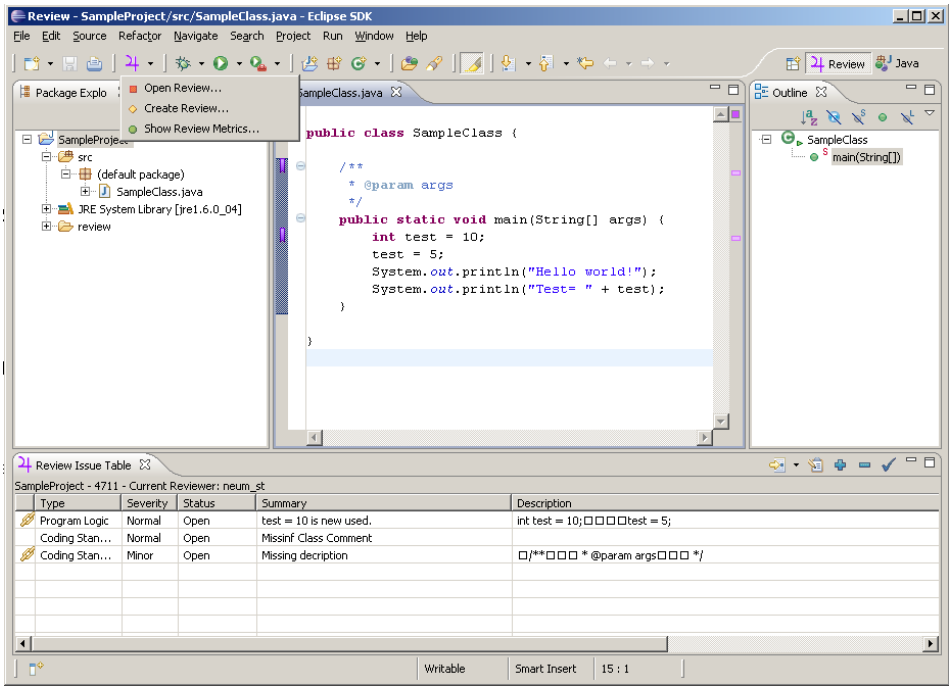
[URL:SC] http://www.dlr.de/sc; Aufruf: 09.02.2008

# 12 Anhang

## 12.1 Jupiter-Anleitung

### 12.1.1 Review-Perspektive

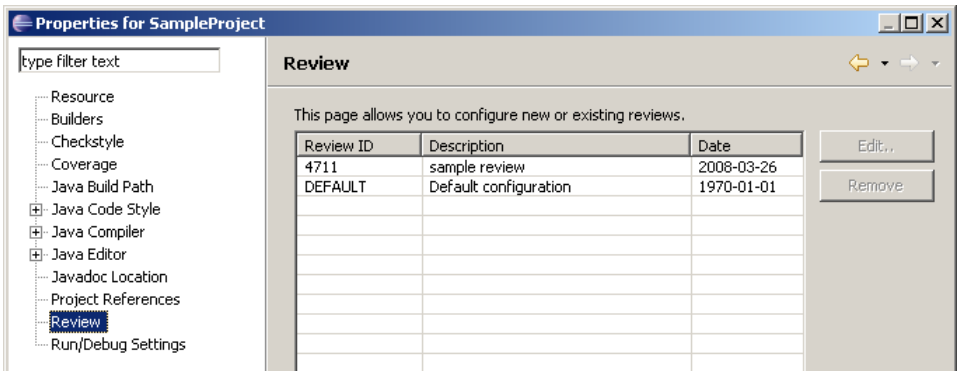
Die Review-Perspektive wird über Window -> Open Perspective -> Other -> Review geöffnet.



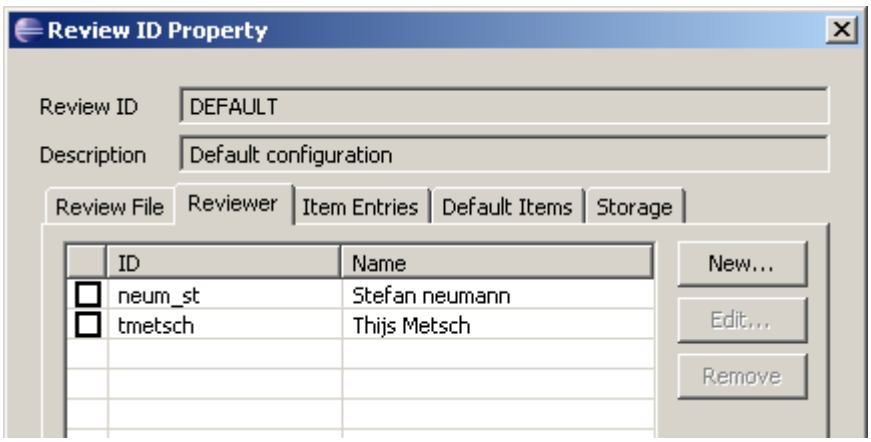

Dabei stellt die Review-Perspektive den Jupiter-Pulldown-Button, sowie den „Review Issue Table“ zur Verfügung. Review-Issues werden im Editor mit einem violetten Fähnchen gekennzeichnet.

### 12.1.2 Default-Konfiguration

Die Default-Konfiguration erfolgt über die Projekteigenschaften. Dort werden unter dem Eintrag "Review" alle Reviews des Projektes aufgelistet. Über die Default-Konfiguration (DEFAULT ID) können nun Einstellungen gemacht werden, die eine Voreinstellung für alle neuen Reviews sind.

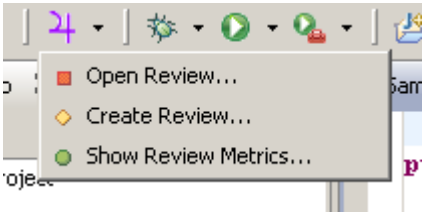



In der Default-Konfiguration können dann auch die Default-Reviewer eingestellt werden. Diese werden im Projektordner in der Datei „.reviewer“ gespeichert.



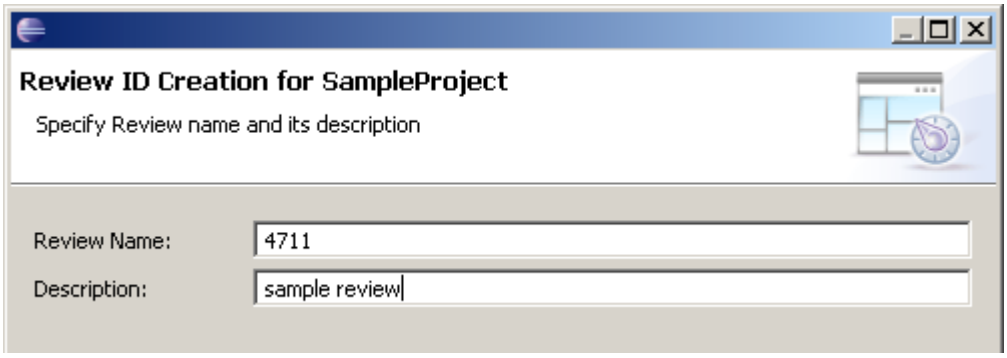

### 12.1.3 Review anlegen

Ein Review kann entweder über den Jupiter Pulldown Button ("Create Review...") oder über File -> New -> Other -> Review erstellt werden.

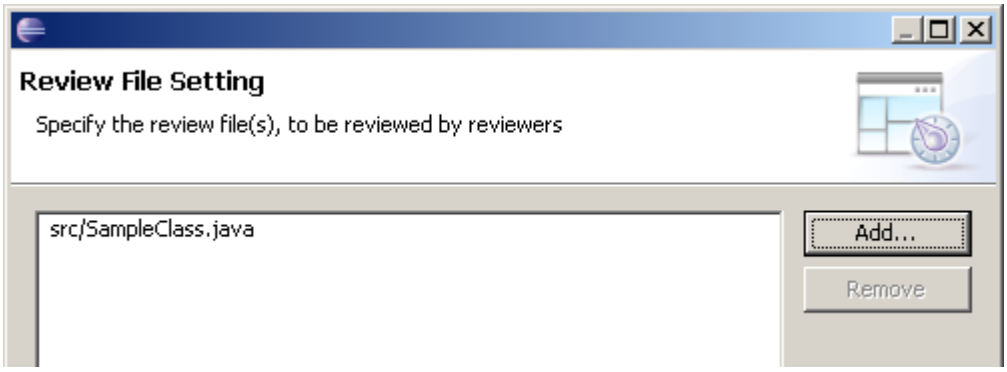



Im folgenden Wizard kann das Review dann konfiguriert werden.

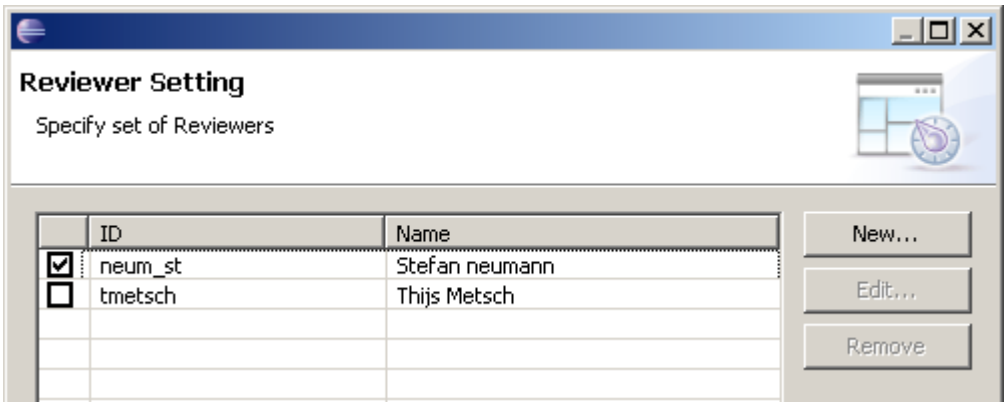
Auf Seite 1 werden ein Name und eine Beschreibung vergeben.




Auf Seite 2 werden die Dateien ausgewählt.

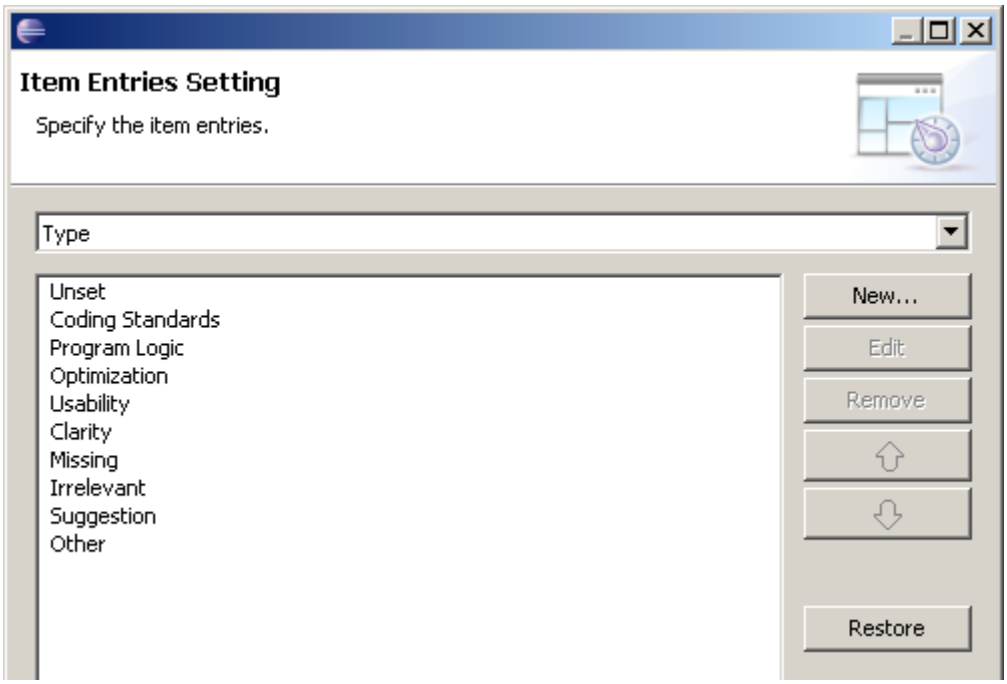



Auf Seite 3 werden die Reviewer ausgewählt.

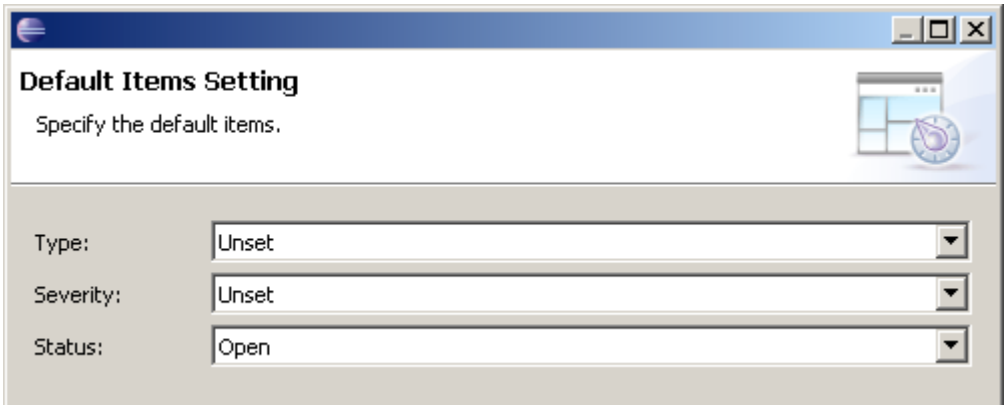



Auf Seite 4 wird der Reviewer ausgewählt, der in der Reviewsitzung Jupiter bedient.


Auf Seite 5 werden die Item Entries für Type, Severity und Status konfiguriert.




Auf Seite 6 werden dann die Default Items ausgewählt.



### 12.1.4 Review entfernen

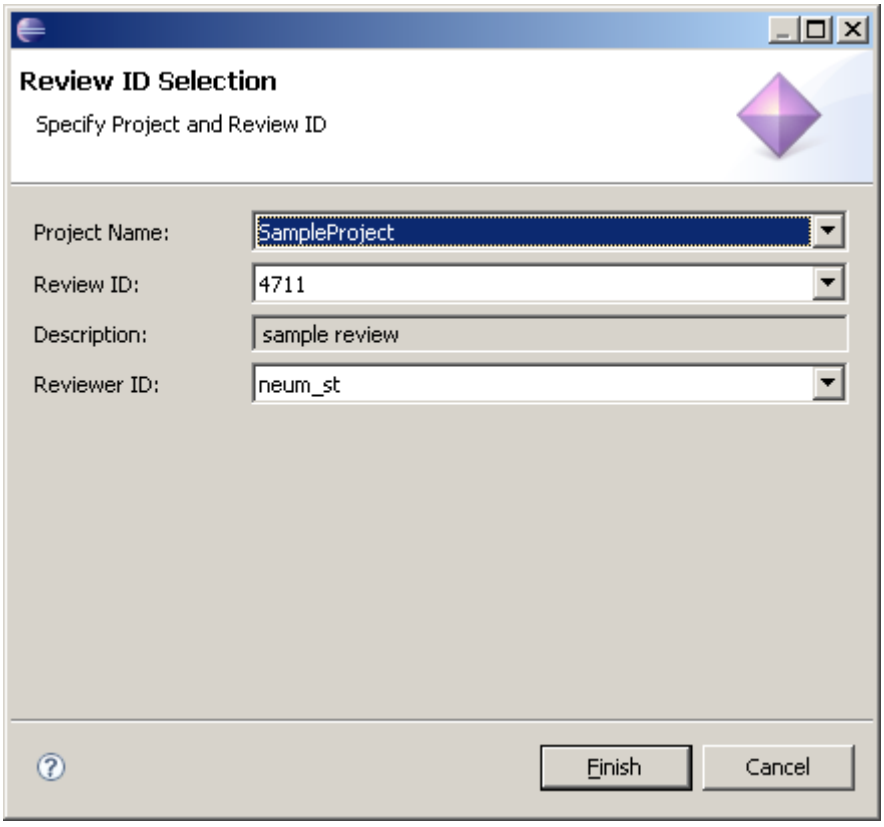
Reviews können über die "Review" Page in den Projekteigenschaften entfernt werden.

### 12.1.5 Review editieren

Reviews können über die "Review" Page in den Projekteigenschaften editiert werden.

### 12.1.6 Review starten

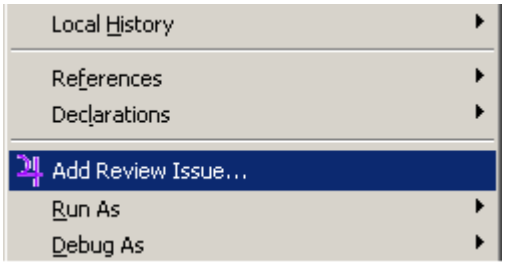
Nach Erzeugen eines neuen Reviews startet das Review automatisch. Ein bereits bestehendes Review kann über den Jupiter Pulldown Button gestartet werden. Nach Auswahl von "Open Review..." startet ein Auswahldialog.

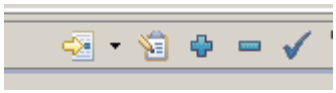



Dort wird neben dem Projekt und dem Review auch der Reviewer ausgewählt. Nach dieser Auswahl wird der „Review Issue Table“ aktualisiert und es kann mit dem Review begonnen werden.

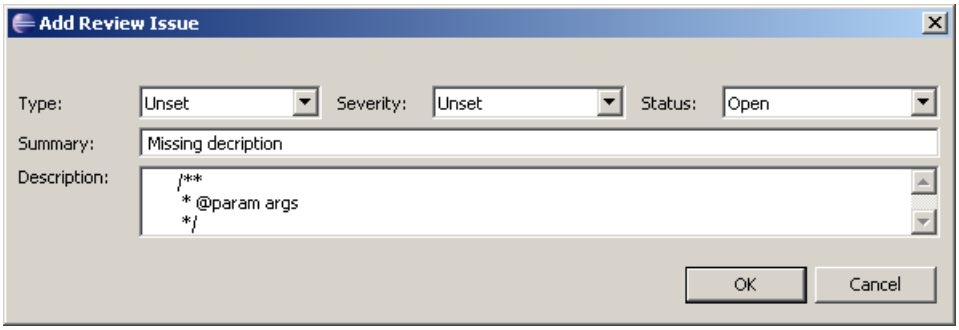
### 12.1.7 Review-Issues hinzufügen

Review-Issues können auf zweierlei Arten hinzugefügt werden. Zum einen durch einen Rechtsklick im Sourcecode und dann über den Eintrag "Add Review Issue" und zum anderen über den „Review Issue Table“ durch Betätigen der "+" Taste.



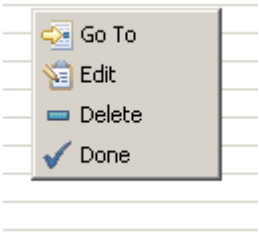



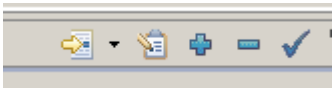
Dabei wird jeweils der "Add Review Issue" Dialog gestartet, in dem der Review-Issue konfiguriert werden kann.




### 12.1.8 Review-Issues editieren

Das Editieren eines Review-Issues erfolgt über den Edit-Button im „Review Issue Table“ oder über "Edit" im Kontextmenü.





Nach Auswahl erscheint der "Edit Review Issue" Dialog. Dieser entspricht dem "Add Review Issue" Dialog. Statt der Defaultwerte für Type, Severity und Status sind dort nun die Werte des ausgewählten Review-Issues eingetragen.

### 12.1.9 Review-Issue löschen

Das Löschen eines Review-Issues erfolgt über den Remove-Button im „Review Issue Table“ oder über "Remove" im Kontextmenü.

### 12.1.10 Status eines Review-Issue auf "Closed" setzen

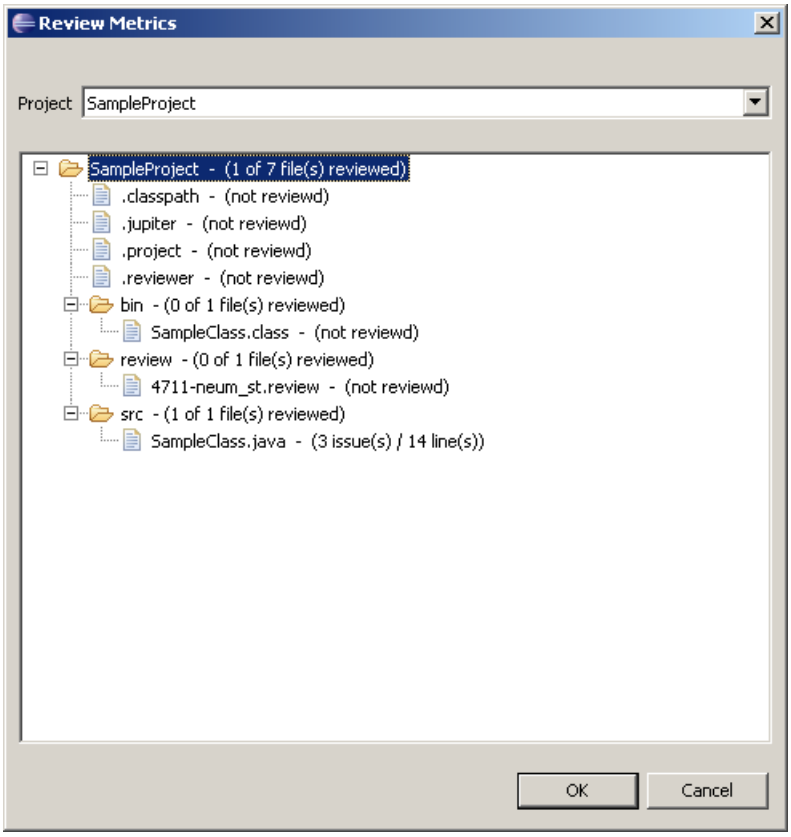
Um in der Nachbereitung den Status eines Review-Issue schnell auf "Closed" setzen zu können, verfügt der „Review Issue Table“ über einen Menüeintrag "Done". Nach Auswahl von "Done" wird der Status des Review-Issues direkt auf "Closed" gesetzt.

### 12.1.11 Sprung in den Sourcecode

Über den "goto"-Button im „Review Issue Table“ springt Jupiter direkt an die Stelle im Sourcecode, an der der Review-Issue eingefügt wurde.

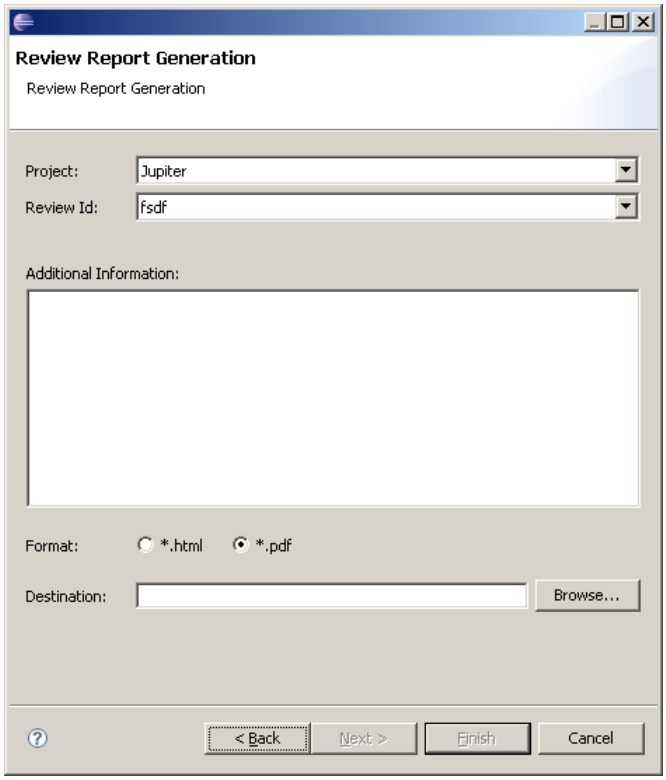
### 12.1.12 Reviewmetriken anzeigen

Die Reviewmetriken können über den Pulldown-Button ("Show Review Metrics...") angezeigt werden. Die Reviewmetriken zeigen an, welche Dateien bereits Teil eines Review waren und wie viele Review-Issues in diesen Dateien existieren.



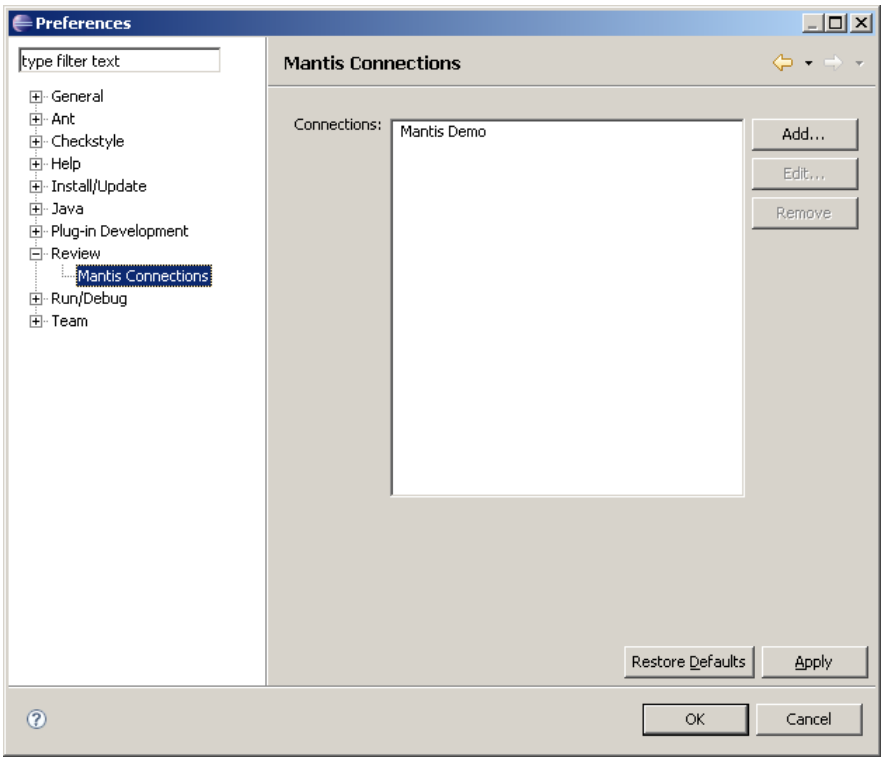

### 12.1.13 Reviewreport generieren

Ein Review Report kann über Export -> Review -> Review Report generiert werden.




### 12.1.14 Mantis-Verbindungen konfigurieren

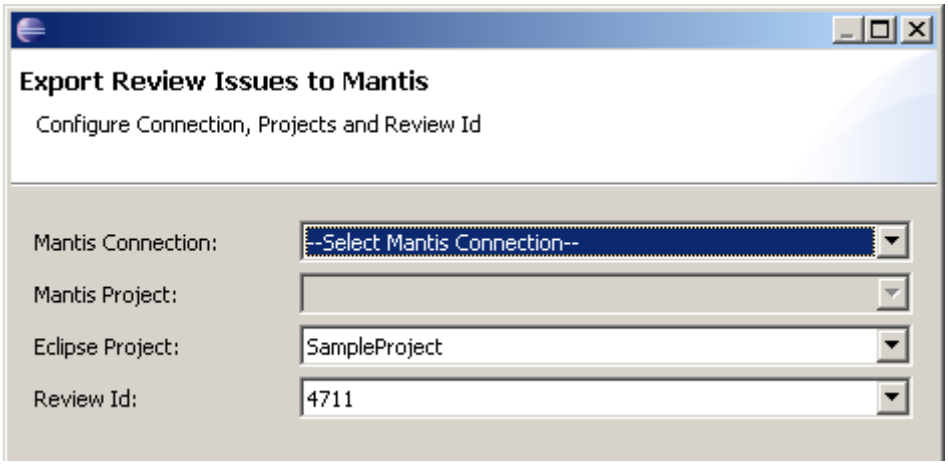
Die Mantis-Verbindungen können über Window -> Preferences -> Review konfiguriert werden.



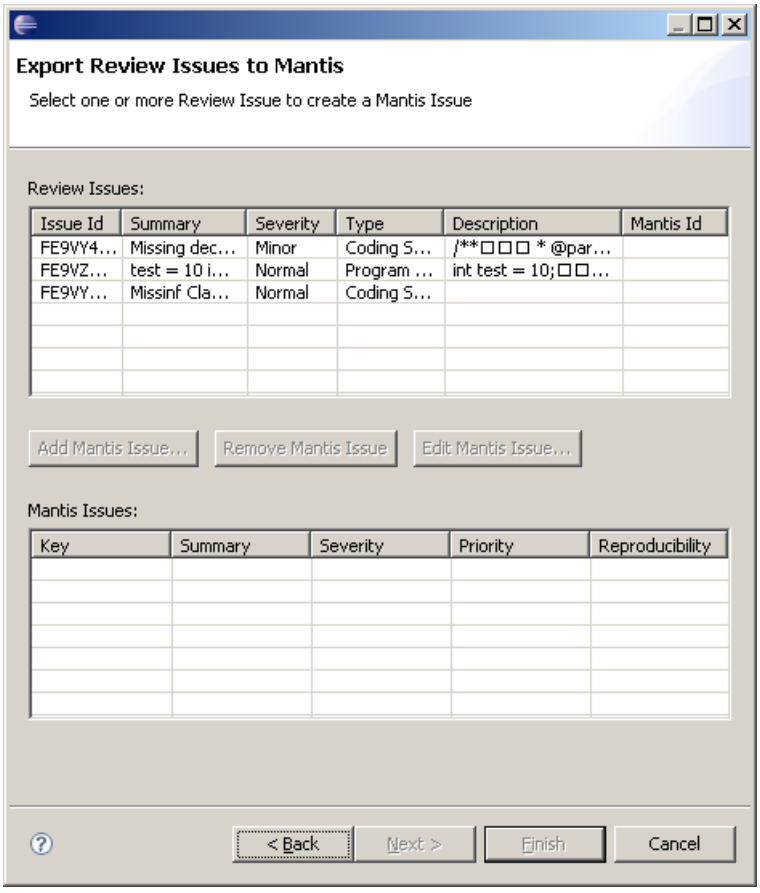

### 12.1.15 Review-Issue nach Mantis exportieren

Die Review-Issues können über Export -> Review -> Mantis nach Mantis exportiert werden.

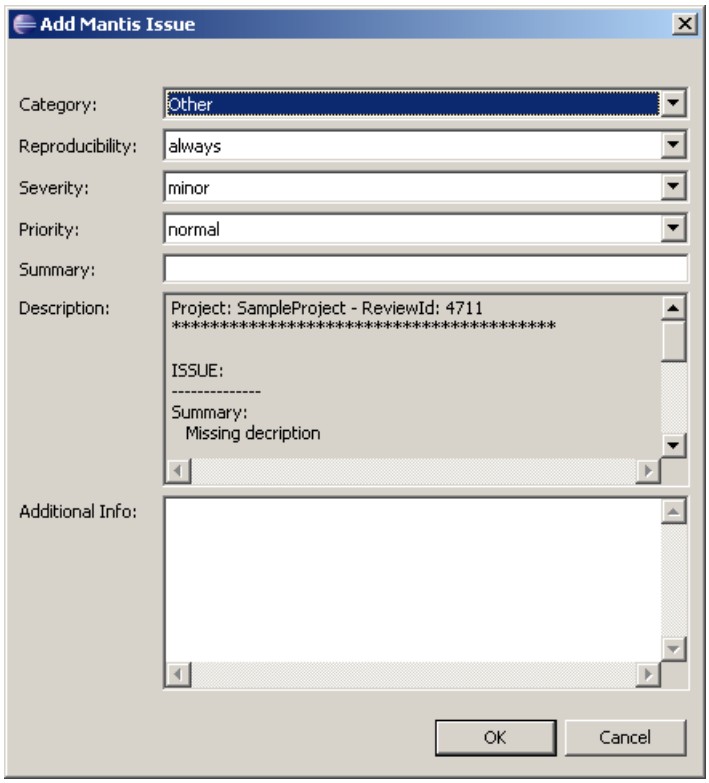
Dabei werden zuerst die Mantis-Verbindung, das Mantis-Projekt, das Eclipse-Projekt und das Review ausgewählt.



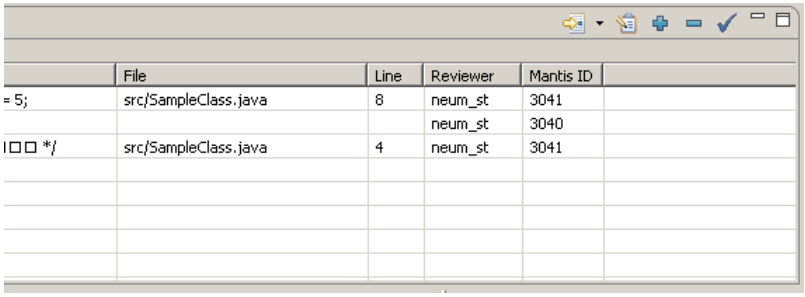
Danach können die Review-Issues durch Mehrfachauswahl in der Tabelle beliebig gruppiert werden.




Nach Auswahl von "Add Mantis Issue" kann der Mantis-Issue konfiguriert werden. Die Review-Issues stehen im Description Feld.




Nach dem erfolgreichen Export wird den Review-Issues automatisch die entsprechende Mantis-Id zugewiesen.




### 12.1.16 Default-Checkliste

Eine Default-Checkliste befindet sich unter Help -> Cheat Sheets... -> Review -> Review Checklist

## 12.2 CD ROM

Eine CD ROM befindet sich am Ende dieser Arbeit.

Sie beinhaltet:

- Fertiges Plugin
- Sourcecode des Plugins
- Diese Arbeit im pdf Format

## Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Datum, Ort Unterschrift