MASTERARBEIT

Plugin-basiertes Requirements Tracing in der Softwareentwicklung

ausgeführt am Institut für

Softwaretechnik und Interaktive Systeme (IFS)

an der Technischen Universität Wien

unter der Anleitung von

ao. Univ.-Prof. Mag. Dipl.Ing. Dr. Stefan Biffl und Dipl.Ing. Matthias Heindl

durch

Gerhard Totz, BSc
2564 Weissenbach, Wiesengasse 1

_____ _____

(Datum) (Unterschrift Student)

Danksagung

Nun ist es an der Zeit mich für die großartige Unterstützung bei allen Beteiligten zu bedanken. Besonderer Dank gilt meinen Betreuern Dr. Stefan Biffl und DI Matthias Heindl, die mir immer mit Rat und Tat zur Seite standen und mir den einen oder anderen hilfreichen Tipp gegeben haben. Bedanken möchte ich mich auch bei allen Professoren und Vortragenden der TU Wien, die mir durch viele interessante Vorträge zu neuem Wissen verhalfen.

Auch bei der Firma Siemens PSE möchte ich mich vielmals für die großartige Unterstützung bei der Entwicklung und der Evaluierung des RT-Plugins bedanken. Dies verleihte meiner Arbeit einen praxisnahen Bezug, aus dem ich viel für die Zukunft mitnehmen konnte.

Ein großer Dank gilt auch meinen Eltern, ohne deren Unterstützung und Zuspruch ich das Studium nicht in der Form hätte absolvieren können. Sie haben mich finanziell und auch mental unterstützt und immer an mich geglaubt, auch wenn ich zwischendurch mal etwas unentschlossen in der Wahl der Fachrichtung war.

Kurzfassung

Requirements Engineering ist eine Disziplin, die sich mit der Erhebung und Verwaltung von Anforderungen an ein Softwareprodukt beschäftigt. Aufgrund von sich häufig ändernden Anforderungen ist deren Verfolgung (Requirements Tracing) ein wichtiges Instrument um Auswirkungen dieser Änderungen effizient abschätzen zu können. Außerdem ist Requirements Tracing von verschiedenen Standards, wie zum Beispiel CMMI, vorgeschrieben und stellt vor allem in sicherheitskritischen Projekten ein Muss dar.

Allerdings gibt es beim Requirements Tracing folgende Herausforderungen:

- traditionelle manuelle Ansätze (Excel Matrizen, etc.) sind sehr aufwändig vor allem bei einer hohen Anzahl von Anforderungen;
- Automatisierungsansätze liefern in hohem Maße unvollständige und teilweise falsche Verknüpfungen;
- vorhandene Ansätze unterstützen das Erstellen der Verknüpfungen über Toolgrenzen hinaus nicht.

Im Rahmen dieser Arbeit habe ich folgende Beiträge geleistet:

- *Entwicklung eines Plug-ins:*
  Ich habe ein in die Entwicklungsumgebung integriertes Tool (RT-Plugin) entwickelt, welches es dem Entwickler ermöglicht, in seiner gewohnten Entwicklungsumgebung Abhängigkeiten zwischen den Anforderungen und den Source-Code-Elementen (Klassen und Methoden) aufwandsschonend herzustellen. Durch diese Tool-Unterstützung wird das Tracing über Toolgrenzen hinaus möglich (Tracing von Anforderungen in einem Anforderungsmanagement Tool zu Source Code in einer Entwicklungsumgebung).
- *Erstevaluierung des Plug-ins:*
  Bei der Erstevaluierung wurde der plugin-basierte Tracing-Ansatz (RT-Plugin) mit traditionellen Ansätzen (Tracing mittels Excel bzw. RequisitePro) hinsichtlich Aufwand des Tracings, Korrektheit und Vollständigkeit der Traces verglichen.

- *Grobevaluierung durch Experten der Siemens PSE und Weiterentwicklung:* Die Grobevaluierung wurde von Requirements Engineering Experten der Siemens PSE durchgeführt. Das Ergebnis waren einige Verbesserungsvorschläge und zusätzliche Anforderungen an das RT-Plugin, wie zum Beispiel: Anzeigen der verknüpften Klassen bei den Anforderungen, Client Server Architektur der Schnittstelle zum Anforderungsmanagement Tool, Änderungsbenachrichtigung in der IDE wenn sich im Anforderungsmanagement Tool eine verknüpfte Anforderung geändert hat, Anzeigen der ausführlichen Anforderungsbeschreibung und deren Attribute und Historie. Das RT-Plugin wurde nach der Grobevaluierung weiterentwickelt und die neuen Anforderungen und Verbesserungsvorschläge umgesetzt.

- *Erstellung eines Konzeptes für eine detaillierte Evaluierung von Aufwand, Vollständigkeit und Korrektheit des Tracings im Rahmen eines Fallstudien-Projekts bei Siemens PSE*

Durch das RT-Plugin wird der Aufwand des Requirements Tracing im Vergleich zu den beiden anderen Ansätzen auf ein Sechstel reduziert, und auch eine höhere Vollständigkeit und Korrektheit der Traces erreicht (muss in weiteren Fallstudien überprüft werden). Durch die Einarbeitung von Änderungen, die von Siemens Praktikern eingebracht wurden, wurde die Praxistauglichkeit des Plug-ins erhöht.

Abstract

Requirements engineering is a discipline which deals with collecting and managing requirements for a software product. During development the requirements often change and requirements tracing is a meaningful instrument to evaluate the impact of these requirements changes effectively. Furthermore, requirements tracing is required in many standards like CMMI and is mandatory in safety critical projects.

Despite the benefits, there are some challenges in requirements tracing:

- traditional manual approaches (Excel matrices, etc.) are very expensive especially in projects with a high amount of requirements.
- Approaches with automated generation of traces deliver highly incomplete and partially incorrect traces.
- Existing approaches do not support tracing across tool borders.

Within this paper I provide the following contributions:

- *Plugin development:*
  I have developed a plugin (RT-Plugin) which is integrated in the development environment of the developer (Eclipse). With this plugin the developer can capture traces between requirements and source code elements (Classes and Methods) within his familiar environment very cost-efficiently. Also tracing over tool borders becomes feasible (tracing of requirements which are captured in a requirements management tool to source code elements within the development environment).
- *Preevaluation of the plugin:*
  Within this preevaluation the plugin-based approach (RT-Plugin) was compared to the traditional approaches (tracing in Excel and RequisitePro) concerning tracing effort, correctness and completeness of the captured traces.

- *Raw evaluation with experts of Siemens PSE and further development:* This evaluation was done by requirements engineering experts of Siemens PSE. As result of this evaluation, some change requests occurred, such as: displaying the traced classes for each requirement, client server architecture, and notification when a requirement has been changed in the requirements management tool, displaying the whole description of the requirement and all their attributes and history entries. The RT-Plugin was afterwards enhanced according to these new change requests.
- *Creation of a concept for a detailed evaluation of the RT-Plugin within a case study project at Siemens PSE concerning tracing effort, correctness and completeness.*

By the RT-Plugin comparing to other approaches the tracing effort can be reduced to a sixth part and also a higher correctness and completeness can be reached (has to be checked in further case studies). By incorporation of the change requests and enhancements, which came from Siemens practitioners, the RT-Plugin became a practicable tool.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

1 Einleitung

In der Softwareentwicklung stellt das Requirements Engineering einen bedeutenden Bereich dar. Unter dem Begriff Requirements Engineering werden die Aufgaben der Anforderungsfindung, der Anforderungsanalyse, der Validierung der Anforderungen und des Anforderungsmanagements zusammen gefasst. Bei der Anforderungsfindung, werden die Anforderungen, die an die zu entwickelnde Software gestellt werden, in Kooperation der Systementwickler, der Entwickler und der Kunden ausgearbeitet und definiert. Dabei gilt es alle wichtigen Punkte fest zu halten seien es Anforderungen an das System, die Performance oder die gewünschte Funktionalität der Software. Im Anschluss an die Anforderungsfindung werden diese im Rahmen der Anforderungsanalyse auf Realisierbarkeit, Korrektheit und Vollständigkeit überprüft und fehlerhafte Anforderungen ausgeschlossen. Wurden alle Anforderungen in dem Anforderungsdokument zusammengefasst, so werden diese im Rahmen der Validierung nochmals überprüft um sicher zu stellen, dass diese das System beschreiben, das es zu Implementieren gilt.

Im Rahmen des Anforderungsmanagements werden diese Anforderungen verwaltet und gewartet. Da es während des Entwicklungsprozesses immer wieder zu Änderungen der Anforderungen kommt, müssen diese Änderungswünsche sorgfältig verwaltet und dokumentiert werden. Für jeden Änderungswunsch muss festgestellt werden, welche Anforderungen betroffen sind und welche Artefakte ebenfalls angepasst werden müssen. Um dies zu bewerkstelligen werden die Anforderungen meist in Anforderungsmanagement Tools verwaltet, die es ermöglichen die Anforderungen mit den dazu gehörigen Artefakten zu verknüpfen. Der Vorteil dabei ist, dass leicht festgestellt werden kann welche Artefakte von der Änderung betroffen sind.

Der Begriff Requirements Tracing umfasst die Tätigkeiten der Erstellung der Verknüpfungen und stellt eine wichtige Rolle im Anforderungsmanagement dar. Auf die Definition und die Vorteile werde ich im Folgenden etwas genauer eingehen.[16]

Nun was ist Requirements Tracing? Gotel und Finkelstein[9] definieren Requirements Tracing wie folgt:

„Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction."

Es geht also um die Verfolgung der Anforderungen durch deren Lebenszyklus. In der Softwareentwicklung werden während des Projektes viele Dokumente erstellt, die meistens von einander abhängig sind. Ändert sich ein Dokument oder eine Anforderung in einem Dokument so hat dies Auswirkungen auf viele andere Dokumente. Um hier einen Überblick zu behalten, welche Dokumente von einer Änderung betroffen sind und eventuell angepasst werden müssen, werden meist Anforderungsmanagement Tools verwendet. In diesen Tools können die Anforderungen und die Dokumente verwaltet und auch Verknüpfungen (Traces) zu anderen Artefakten (anderen detaillierteren Anforderungsbeschreibungen oder externen Dokumenten und Diagrammen) erstellt werden.

Begriffsdefinitionen:

Tracing	Tracing beschreibt das Erstellen von Verknüpfungen zwischen Anforderungen und Artefakten.
Trace	Ein Trace bezeichnet demnach eine einzelne Verknüpfung einer Anforderung zu einem Artefakt.

Es gibt eine Vielzahl an Gründen, die für den Einsatz und die Durchführung des Tracings sprechen. Die wichtigsten sind:

- *Change Impact Analysis:*
  Wird eine Anforderung geändert, so kann der Projektmanager leicht den erstellten Verknüpfungen folgen und so die Artefakte, die möglicherweise von der Änderung betroffen sind, finden und überprüfen. Der Projektmanager kann dadurch leichter einen Überblick über den Aufwand der Änderung ermitteln und so über die Umsetzung entscheiden.
- *Unterstützung bei der Verifizierung:*
  Tracing wird von vielen Standards, wie zum Beispiel CMMI, vorgeschrieben. Besonders in sicherheitskritischen Projekten müssen diese Traces erstellt werden um alle Anforderungen bis hin zu den Testfällen verfolgen zu kön-

nen. Dadurch kann nachgewiesen werden, dass alle Anforderungen korrekt implementiert und auch getestet wurden.

- *Beitrag zum Programmverständnis:*
  Die Entwickler können durch die erstellten Verknüpfungen leichter die komplexen Komponenten der Software und deren Zusammenhänge verstehen. Dadurch bekommen die Entwickler auch einen besseren Überblick über das ganze Projekt. Dieser Punkt dient auch zur Steigerung der Motivation der Entwickler diese Traces zu erstellen.[20] Die Motivation der Entwickler spielt eine wichtige Rolle in Hinsicht auf die Vollständigkeit und Korrektheit der erstellten Abhängigkeiten. Eine hohe Motivation führt zu einer deutlich höheren Qualität.

Wenn es so viele gute Gründe gibt, die für das Tracing sprechen, warum wird dies dann nicht immer umgesetzt?

Der folgende Abschnitt beschreibt die Probleme, die bei der Umsetzung auftreten und mögliche Lösungsansätze.

1.1 Probleme und Lösungsansätze des Requirement Tracings

In diesem Abschnitt möchte ich auf die Probleme, die bei der Umsetzung des Tracings auftreten, und mögliche Lösungsansätze eingehen.

Das größte Problem in der Umsetzung liegt in der fehlenden Integration. Es gibt viele verschiedene Tools, die in der Erstellung der Dokumente, der Diagramme und der anderen Artefakte verwendet werden. Nicht all diese Tools bieten eine integrierte Lösung um Verknüpfungen mit Anforderungen, die in einem Anforderungsmanagement Tool verwaltet werden, zu erstellen. Der Benutzer muss für die Erstellung einer Verknüpfung zwischen einem Dokument und einer Anforderung immer in ein anderes Tool wechseln. Ein erhöhter Mehraufwand ist damit verbunden. Noch dazu muss der Benutzer sich mit dem Anforderungsmanagement Tool auskennen um eine Verknüpfung erstellen zu können und Zugang zu dem Tool haben. Dies ist mit zusätzlichem administrativen und auch finanziellen (Lizenzkosten) Aufwand verbunden.

Durch die fehlende Integration werden die Verknüpfungen meist erst im Nachhinein erstellt. Daraus folgt eine höhere Fehleranfälligkeit, da leichter auf Verknüpfungen vergessen wird oder falsch erstellt werden. Somit ist die Qualität der erstellten Ver-

knüpfungen gering und es stellt sich die Frage weshalb die Verknüpfungen überhaupt erstellt werden, wenn nicht davon ausgegangen werden kann, dass alle erstellten Verknüpfungen korrekt sind.

Werden die Entwickler durch ein aufwändiges Tracing und dem Wechseln zwischen mehreren Tools von deren ursprünglichen Arbeit, der Implementierung, abgehalten, sinkt die Motivation und somit die Qualität der erstellten Verknüpfungen. Dies wird oft unter dem Begriff „yet another tool syndrom“ beschrieben. Es tritt auf wenn der Entwickler viele verschiedene Tools verwenden muss. Eine bessere Lösung ist es, es dem Entwickler zu ermöglichen die Verknüpfungen zwischen den Anforderungen, die er implementiert, und dem Source Code, leicht und ohne großem Aufwand gleich in seiner Entwicklungsumgebung zu erstellen. So muss er nicht in ein anderes Tool wechseln und kann die Verknüpfungen gleich während der Implementierung erstellen.

Es gibt generell zwei verschiedene Lösungsansätze für das Problem der toolübergreifenden Erstellung der Verknüpfungen:

Lösungsansatz 1:

Erstellen der Verknüpfungen der Artefakte in anderen Tools, die auf die Anforderungen im Anforderungsmanagement Tool verweisen. Das Problem hierbei ist die Erstellung von redundanten Daten, was dazu führt, dass es sehr schwer ist die Daten aktuell zu halten.

Lösungsansatz 2:

Die Integration existierender Tools, sodass der Benutzer die Verknüpfungen direkt in dem Tool erstellen kann mit dem er auch das Dokument oder Artefakt bearbeitet. Die kompletten Informationen über alle Verknüpfungen werden dann in dem Anforderungsmanagement Tool gesammelt und gespeichert.

Hier stellt sich die Frage welche der beiden Lösungsansätze besser ist. Wenn man die beiden Ansätze vergleicht, kommt man bald zu dem Schluss, dass der 2. Lösungsansatz wohl der bessere ist. Es stellt sich nun aber die Frage wie dies umgesetzt werden kann. Als mögliche Lösung bieten sich Plugins an, die die verschiedenen Tools um Funktionalitäten, wie das Verknüpfen mit Anforderungen, erweitern können.

Weshalb sollen Plugins verwendet werden?

Nun es ist eine relativ einfache Möglichkeit die Funktionalität eines Tools zu erweitern. Die Anforderungen, die in dem Anforderungsmanagement Tool definiert und gewartet werden, können importiert und in einem integrierten Fenster angezeigt werden. So können die Verknüpfungen erstellt und die Informationen über die Verknüpfungen in das Anforderungsmanagement Tool exportiert werden. Dort können dann alle Verknüpfungen zwischen den verschiedenen Artefakten dargestellt werden. Die Vorteile sind, dass die Daten zentral, zum Beispiel im Anforderungsmanagement Tool, gespeichert werden und der Benutzer die Verknüpfungen direkt in seiner Arbeitsumgebung / Arbeitstool erstellen kann. Er braucht auch keine zusätzlichen Tools erlernen und es fallen auch keine zusätzlichen Administrations- und Lizenzkosten an.

In der Praxis treten jedoch immer wieder verschiedenste Probleme bei der Erstellung der Verknüpfungen auf:

- Die Identifizierung und die Wartung der Verknüpfungen führen zu zusätzlichem Aufwand, welcher bei steigender Anzahl an Anforderungen untragbare Größen erreichen kann.
- Personen, die die Verknüpfungen erstellen haben nicht immer einen direkten Nutzen davon. So erstellt zum Beispiel der Entwickler die Verknüpfungen der Anforderungen mit dem Source Code. Jedoch werden diese Informationen vom Projektmanager verwendet um die Change Impact Analyse durchzuführen oder um den Projektfortschritt zu überwachen. Es sollte also auch darauf geachtet werden, dass die Entwickler selbst einen Nutzen vom Tracing haben.

Meistens werden die Verknüpfungen von den Entwicklern manuell erstellt, indem sie die Referenzen zu den Anforderungen direkt in den Artefakten eintragen[6]. Eine andere und oft benutzte Lösung ist das Erstellen von Matrizen, in denen die Abhängigkeiten abgebildet werden[15].

Es werden auch Tools verwendet, in denen die Traces erstellt und gespeichert werden[3][22]. Diese verursachen allerdings meist ein „yet another tool syndrom“, da die Benutzer das neue Tool nicht verwenden wollen. Zusätzlich entstehen dadurch redundante Daten, die die Verwaltung und Wartung erschweren.

Es gibt Fallstudien, die beschreiben wie die Integration von Tools für das Tracing in bestehende Plattformen unterstützt werden kann. Es wird auch beschrieben was man

aus diesen Studien lernen kann, vor allem über das Tracing zwischen Anforderungen und den Source Code Elementen.[20]

Folgende Punkte können daraus abgeleitet werden:

- *Schaffe eine problemlose Integration mit bestehenden Tools:*
  Entwickler haben meist nur geringes Interesse daran neue komplexe Tools zu verwenden, die sie nur davon abhalten ihre Arbeit zu machen. (das Entwickeln)
- *Führe nur kleine Änderungen in den Arbeitsablauf der Entwickler ein:*
  Der Schlüssel zum Erfolg ist die Reduktion der Änderung der Arbeitsabläufe der Entwickler für das Tracing.
- *Fokussieren auf bestehende Entwicklungsabläufe:*
  Die Erstellung von Verknüpfungen zu Artefakten macht nur Sinn, wenn diese Artefakte auch tatsächlich schon im Prozess verwendet werden. Denn das Erstellen von Verknüpfungen zu Artefakten, die niemand verwendet hat keine Berechtigung.

Bezug nehmend auf diese Punkte, meine ich, dass die Entwickler am besten motiviert werden können diese Verknüpfungen zu erstellen und zu warten, wenn der Aufwand möglichst gering gehalten wird und der Entwickler die Verknüpfungen in seiner gewohnten Entwicklungsumgebung erstellen kann und keine zusätzlichen Tools verwenden muss.

Teil meiner Arbeit war die Umsetzung dieses Lösungsansatzes. Dafür habe ich ein Plugin (RT-Plugin) entwickelt, das in die Entwicklungsumgebung(Eclipse) eingebettet ist und es dem Entwickler ermöglicht die Verknüpfungen zwischen den Anforderungen und den Source Code Elementen zu erstellen. Dabei habe ich besonders darauf geachtet, dass der Aufwand für die Erstellung der Verknüpfungen möglichst gering (auf einen Doppelklick je Verknüpfung reduziert) und die Handhabbarkeit besonders einfach ist. Zusätzliches Kriterium war es auch den Entwicklern einen Nutzen der erstellten Verknüpfungen zu verschaffen. Dies wurde realisiert indem bei jeder Anforderung auch der Klassenname derjenigen Klasse angezeigt wird mit der die Anforderung verknüpft ist. Der Entwickler sieht somit auch in seiner Entwicklungsumgebung welche Anforderungen in welchen Klassen schon implementiert wurden. Er kann durch einen Doppelklick auf den Klassennamen gleich die jeweilige Klasse öffnen und bearbeiten. Zusätzlich kann der Entwickler alle wichtigen Informationen zu jeder Anforderung in seiner Entwicklungsumgebung abrufen. Er kann sich die ausführliche Beschreibung, alle Attribute und die Änderungshistorie anzei-

gen lassen. Weiters wird optisch dargestellt wenn sich eine Anforderung im Anforderungsmanagement Tool geändert hat.

Das RT-Plugin bietet meiner Meinung nach folgende Vorteile:

- Es werden die Anforderungen aus dem Anforderungsmanagement Tool (RequisitePro) importiert ohne dabei ein zusätzliches Tool verwenden zu müssen.
- Die Verknüpfungen können in der Entwicklungsumgebung direkt während der Implementierung durch einen Doppelklick erstellt werden.
- Die erstellten Verknüpfungen werden in das Anforderungsmanagement Tool übertragen und können dort angezeigt werden.
- Die vorhandenen Verknüpfungen werden auch in der Entwicklungsumgebung bei den Anforderungen dargestellt.
- Alle wichtigen Informationen zu den Anforderungen können in der Entwicklungsumgebung angezeigt werden.
- Es gibt eine automatische Konsistenzprüfung, mit der die Verknüpfungen, die im Anforderungsmanagement Tool gespeichert sind, mit jenen der Klassen verglichen werden.

Das bedeutet, dass der Aufwand für den Entwickler eine Verknüpfung zu erstellen auf einen Doppelklick reduziert wurde und auch einen Nutzen davon hat, indem er selbst einen Überblick darüber bekommt welche Anforderungen wo implementiert wurden. Dies erhöht die Motivation des Entwicklers diese Verknüpfungen zu erstellen und zu warten.

Der weitere Aufbau der Arbeit gliedert sich wie folgt. Abschnitt 2 gibt einen Überblick über die bisher vorhandenen Lösungsansätze zu dem Thema Requirements Tracing aus der Literatur und vergleicht diese mit den Eigenschaften des pluginbasierten Ansatzes. In Abschnitt 3 wird das RT-Plugin, wie ich es ursprünglich entwickelt habe, genau beschrieben. Abschnitt 4 beschreibt die Forschungsfragen. In Abschnitt 5 werden die durchgeführte Erstevaluierung und die Grobevaluierung, durch Requirements Engineering Experten der Siemens PSE, des RT-Plugins und ein Konzept für eine Evaluierung im Rahmen einer detaillierten Fallstudie bei Siemens beschrieben. Abschnitt 6 dokumentiert die Ergebnisse der Evaluierung, die durchgeführten Weiterentwicklungen des RT-Plugins und diskutiert diese. In Abschnitt 7 wird der Einsatz des RT-Plugins und die Schnittstelle zu den Anforderungsmanagement Tool beschreiben. Abschnitt 8 beschreibt die in der Zukunft noch weiter zu

führenden Tätigkeiten. Abschnitt 9 gibt nochmals eine kurze Zusammenfassung der Arbeit. Im Appendix wird ein Beispiel einer einfachen Schnittstellenerweiterung beschrieben.

2 Verwandte Forschungsarbeiten

In diesem Abschnitt möchte ich auf die in der Literatur beschriebenen Ansätze des Requirement Tracings eingehen, einige davon vorstellen und diese mit den Eigenschaften des plugin-basierten Ansatzes vergleichen.

2.1 Ansätze nach Kosten / Nutzen

Zuerst möchte ich Ansätze vorstellen, die nicht auf die Tool - Unterstützung basieren sondern sich mit dem ökonomischen Aspekt beschäftigen. Diese Ansätze versuchen die Aufwände und Kosten, die beim Tracing entstehen, zu reduzieren indem diese die Anforderungen in verschiedenen Formen kategorisieren und priorisieren.

Bei dem VBRT (Value-based Requirements Tracing) Ansatz von Heindl und Biffl[12] wird der ökonomische Aspekt des Tracings in Betracht gezogen. Dabei wird nicht auf die Datenspeicherung und Datengenerierung eingegangen sondern eine Lösung beschrieben, um den Aufwand der Erstellung der Verknüpfungen zu reduzieren. Dies wird mittels Kategorisierung der Anforderung in 3 Kategorien bewerkstelligt. Dabei werden die Anforderungen nach deren Wichtigkeit eingeteilt und so die Detailliertheit des Tracings festgelegt. Verknüpfungen für Anforderungen mit hoher Wichtigkeit werden bis auf Methoden-Ebene und Anforderungen mit geringer Wichtigkeit nur bis auf Package-Ebene erstellt. Dies ermöglicht eine deutliche Aufwandsreduzierung von bis zu 35% im Vergleich zu einer vollständigen Erstellung aller vorhandenen Verknüpfungen. Dieser Ansatz stellt eine Lösung dar, die die Anzahl der zu erstellenden Verknüpfungen reduziert und dadurch eine Aufwands- und Kostenreduktion erreicht.

Ahn und Chong[1] beschreiben einen Ansatz der auf der Kategorisierung nach den einzelnen Features eines Softwareprojektes basiert. Es wird nach folgendem Prozessablauf vorgegangen: (1) Definition der Anforderungen, (2) Modellierung der Features, (3) Priorisieren der Features, (4) Verknüpfen der Anforderungen, und (5) Evaluierung der Verknüpfungen. Es werden die einzelnen Anforderungen den Features zugeordnet und danach entschieden welche Features den größten Aufwand und die

höchsten Kosten aufweisen (Schritt 3: Priorisieren). Die Anforderungen dieser Features werden genauer mit den Source Codeelementen verknüpft als die Anforderungen von weniger wichtigeren Features.

Da die beiden Ansätze nicht den Prozess der Trace Erstellung adressieren, sondern versuchen die Anzahl der zu erstellenden Verknüpfungen zu reduzieren, können diese Ansätze nur als mögliche Ergänzung angesehen werden. Mir geht es allerdings darum den Prozess der Trace Erstellung zu optimieren.

2.2 Technische Ansätze

In diesem Abschnitt möchte ich Ansätze vorstellen, die auf einer technischen Lösung mittels Tools basieren. Diese Ansätze versuchen durch verschiedene Tools das Tracing zu ermöglichen und zu erleichtern.

Die technischen Ansätze können weiters in zwei Gruppen eingeteilt werden, die manuellen und die automatisierten Ansätze.

Manuelle Ansätze

Manuelle Ansätze verwenden für die Erstellung der Verknüpfungen zwar Tools, jedoch werden diese eher mehr für die Speicherung benutzt. Die Verknüpfungen müssen vom Anwender per Hand in das Tool eingetragen werden.

Bei kleineren Projekten sind Matrizen ein beliebtes Instrument um die Verknüpfungen zu erstellen. Hierbei werden die Anforderungen sowohl vertikal als auch horizontal aufgelistet und wenn eine Abhängigkeit zwischen zwei Anforderungen besteht wird eine Markierung in der entsprechenden Zelle eingefügt. Ist die Anzahl der Anforderungen zu hoch (>250) so werden diese Matrizen sehr groß und unübersichtlich. Deshalb werden hierfür lieber Listen[16] verwendet in denen zu jeder Anforderung die Anforderungen, mit denen eine Abhängigkeit besteht, aufgelistet werden.

Sollen umfangreichere Tools verwendet werden, die mehr Funktionalität bieten und die Daten in Datenbanken speichern und verwalten, so wird sehr oft auf die Technik der Matrizen zurück gegriffen. RequisitePro[24] und Doors[26] sind zwei beliebte Tools, die dies ermöglichen.

Macfarlane und Reilly[19] beschreiben einen Ansatz, bei dem die Verknüpfungsinformationen nicht in den Artefakten sondern in den Objekten, welche die Verknüpfung darstellen, gespeichert werden. Es können die Verknüpfungen aber auch in beliebigen eigenen Datenbanken gespeichert werden.

Der Nachteil dieser eigenständigen und meist komplexen Tools ist, dass die Entwickler diese meist nicht gern verwenden wollen und die Verknüpfungen nicht gewissenhaft und vollständig erstellen und warten[10]. Dadurch, dass die Entwickler hier ihre eigene Entwicklungsumgebung verlassen und in ein anderes sehr komplexes Tool wechseln müssen werden die Verknüpfungen auch nicht immer gleich während der Implementierung erstellt sondern meist erst nach Fertigstellung einer Komponente oder in anderen periodischen Zyklen. Diese Zeitspanne zwischen der Implementierung und der Erstellung der Verknüpfungen kann zu unvollständigen und fehlerhaften Verknüpfungsinformationen führen.

Eine weitere Möglichkeit Verknüpfungen zu erstellen ist das Tagging. Dabei werden bestimmte Schlüsselwörter in den Dokumenten eingefügt, welche von Tools ausgelesen werden können.

Song[25] beschreibt so einen Ansatz mit dem Namen STAR Trac. Dabei wird die Information der Anforderung in einem bestimmten vorgegebenen Format in die Dokumente vom Entwickler eingefügt.

Das Tool besteht aus drei Komponenten:

- Information Extractor: Mit dieser Komponente werden die Dokumente nach den Einträgen durchsucht und die Verknüpfungsinformationen extrahiert.
- Information Manager: Er verwaltet die extrahierten Daten und speichert oder durchsucht diese.
- Information Viewer: Diese Komponente zeigt die Verknüpfungsinformationen an und ermöglicht Abfragen.

Ein anderes Tool, welches auch auf dem Tagging-Ansatz beruht beschreibt Chi Liang Ni [21]. Er beschreibt das UNIX Tool RADIX, welches jedoch über eine nicht besonders leserliche Syntax beruht.

Jackson[14] beschreibt einen auf Schlüsselwörtern basierten Ansatz bei dem in die Dokumente die Schlüsselwörter eingetragen werden und danach mit einem Tool diese ausgelesen und separat gespeichert werden können.

Der plugin-basierte Ansatz basiert auch auf dem Ansatz des Taggings. Jedoch muss der Entwickler die Schlüsselwörter nicht eintippen sondern diese werden automatisch bei Doppelklick eingefügt. Dadurch können Tippfehler vermieden werden.

Automatisierte Ansätze

Bei den automatisierten Ansätzen werden die Verknüpfungen automatisch von Tools erstellt. Dabei werden die Dokumente meist erst im Nachhinein durchsucht und mögliche Verknüpfungen herausgefiltert. So werden oft die Namen der Anforderungen in den Namen der Klassen und Methoden wieder gefunden.

Egyed und Grünbacher[4][5] beschreiben den Trace Analyzer, ein Tool das die Verknüpfungen zwischen den Anforderungen und den Source Codeelementen automatisiert erstellt. Dabei wird der Source Code während der Laufzeit untersucht. Dabei legt der Benutzer zuerst fest welche Anforderung er bearbeiten möchte und führt die dazugehörigen Aktivitäten im Programm aus. Dabei speichert der Trace Analyzer die Klassen und Methoden, die aufgerufen werden und erstellt so die Verknüpfungen zwischen den Anforderungen und den Source Code Elementen. Ein sehr ähnliches Tool ist das FasTLInK[8], welches Gates beschreibt.

Cleland-Huang, Zemont und Lukasik[2] beschreiben einen Prototyp namens TraCS, der als Framework für das Tracing fungiert und mehrere Methoden umsetzt. TraCS verwendet sowohl die Matrizen als auch die automatisierte Erstellung der Verknüpfungen durch Event basierte Systeme.

Lefering[17] beschreibt ein Framework namens IPSEN, in dem Integrationstools die Erstellung der Verknüpfungen zwischen Dokumenten automatisiert und durch Regeln gesteuert ausführen. Bei diesem Ansatz werden allerdings bislang noch keine Tools angeboten, die die Verknüpfungen in den Source Code anbieten.

Pinheiro und Goguen[22] beschreiben ein Tool namens TOOR, welches ein objektorientiertes Tool für die Erstellung von Verknüpfungen zwischen allen Artefakten ermöglichen soll.

Huffman-Hayes [10][11] beschreibt das Tool RETRO, welches ausschließlich für die Erstellung von Verknüpfungen entwickelt wurde. Mit diesem externen Tool können Tracing Matrizen erstellt werden. Es kann aber auch durch eine einfache XML Form mit anderen Tools aus dem Projektmanagement zusammenarbeiten. Als Basis hat RETRO eine Informationsbasierte Toolbox, welche das Filtern der Dokumente durch einen Algorithmus ermöglicht.

Poirot ist ein Tool, welches Lin [18] beschreibt. Poirot ist ein serverbasiertes System und verwendet ebenfalls einen Algorithmus um die Verknüpfungen zu filtern. Die Serveranwendung kann mittels XML Interface mit vielen CASE-Tools zusammenarbeiten.

2.3 Zusammenfassung

Wie in den vorigen Abschnitten beschrieben, gibt es derzeit viele verschiedene Ansätze zu diesem Thema. Jedoch beziehen sich diese nicht auf die Optimierung der Erstellung der Verknüpfungen selbst oder es werden Tools und Methoden angeführt, die nicht in die Tools der Entwickler integriert sind. Diese Tools und Methoden mögen alle ihre Vorteile haben, jedoch erfüllen diese nicht die Anforderungen von Neumüller[20], die ich in der Einleitung beschrieben habe.

Ansätze, die versuchen die Kosten/Nutzen Rechnung zu optimieren können nur als mögliche Ergänzung angesehen werden, da diese nicht an der Optimierung der Trace-Erstellung ansetzten. Sie können nicht die Motivation der Entwickler steigern die Verknüpfungen zu erstellen und somit nicht die Vollständigkeit und Korrektheit der erstellten Verknüpfungen positiv beeinflussen.

Manuelle tool-unterstütze Ansätze sind meist sehr Aufwändig in der Verknüpfungserstellung und resultieren in dem „yet another tool syndrom“ der Entwickler. Diese müssen dazu meist externe Tools verwenden, deren Handhabung oft sehr komplex ist. Zusätzlich zu dem erhöhten Aufwand in der Verknüpfungserstellung kommt noch der erhöhte finanzielle Aufwand hinzu, da die Entwickler auf die komplexen Tool eingeschult werden müssen und eigene Lizenzen benötigen.

Automatisierte tool-unterstütze Ansätze liefern oft fehlerhafte und unvollständige Verknüpfungsinformationen. Es werden Verknüpfungen erstellt, die gar keine sind oder auf existierende Abhängigkeiten vergessen. Um diese Informationen zu überprüfen und zu kontrollieren ist ein enormer Aufwand nötig.

Aus diesen Gründen habe ich ein Plugin entwickelt, welches in die Entwicklungsumgebung integriert ist und es dem Entwickler ermöglicht die Verknüpfungen besonders aufwandsschonend zu erstellen. Das Plugin kann mit allen beliebigen Anforderungsmanagement Tools kombiniert werden und bietet dem Entwickler nützliche Informationen bei der Implementierung direkt in seiner Entwicklungsumgebung.

3 Praktische Arbeit

In diesem Kapitel wird das von mir entwickelte RT-Plugin (RT steht für Requirements Tracing) beschrieben. Zuerst möchte ich einen kurzen Überblick über die Idee die hinter dem RT-Plugin steht geben. In den weiteren Teilabschnitten des Kapitels wird auf die Anforderungen, die an das RT-Plugin gestellt werden, eingegangen. Die Architektur, die den Aufbau des RT-Plugins beschreibt, veranschaulicht. Danach wir die Funktionsweise des RT-Plugins beschrieben. (In Kapitel 7 wird der Einsatz des RT-Plugins an Hand eines Beispiels veranschaulicht und auch die Schnittstelle genau beschrieben.)

Die zu Grunde liegende Idee des RT-Plugins war, dem Entwickler es so einfach wie möglich zu machen, den von ihm produzierten Code mit den Anforderungen, die in einem Anforderungsmanagement Tool verwaltet werden, zu verknüpfen.

Das bedeutet:

- Die Informationen über die Anforderungen müssen aus dem Anforderungsmanagement Tool (z.B.: RequisitePro) exportiert werden und dann in die Entwicklungsumgebung (Eclipse) importiert und in einem integrierten Fenster angezeigt werden. Aus welchem Anforderungsmanagement Tool die Anforderungen exportiert werden sollen kann in den Projekteinstellungen der Entwicklungsumgebung festgelegt werden.

- Der Entwickler, der gerade an einer Methode oder Klasse arbeitet, braucht nur den Cursor in den Bereich der Methode oder der Klasse platzieren und auf die gewünschte Anforderung in dem integrierten Fenster doppelklicken um die Verknüpfung herzustellen.

- Die eindeutigen Informationen zu der Anforderung (ID, Name) wird dann in den JavaDoc Bereich der jeweiligen Methode oder Klasse eingefügt. Wenn der Entwickler den Code speichert, werden die Informationen über die Verknüpfungen in das in den Projekteinstellungen definierte Anforderungsmanagement Tool exportiert. Dort können die Verknüpfungen zwischen den An-

forderungen und den Code Elementen in einer Matrix dargestellt werden. (In RequisitePro werden diese Verknüpfungen in einer Traceability Matrix dargestellt).

Generell soll das RT-Plugin dem Entwickler eine Möglichkeit bieten die Verknüpfungen zwischen den Anforderungen und den Code Elementen ohne großen Aufwand und während der Implementierung zu erstellen.

3.1 Anforderungen

In diesem Abschnitt werden die Anforderungen, die an das RT-Plugin gestellt werden genau beschrieben. Dabei stehen die einzelnen Teilabschnitte für die Benutzeranforderungen. Zu jeder Benutzeranforderung werden in einer Aufzählung die zugehörigen technischen Anforderungen angeführt.

3.1.1 Definieren der Projekteinstellungen

Der Benutzer soll mehrere projektspezifische Einstellungen vornehmen können, die das Handling des Plugins an seine Vorlieben anpassen. Zusätzlich soll der Benutzer die Schnittstelle für den Import und den Export der Anforderungen und der Verknüpfungen festlegen können.

- *Auswahl der Import Schnittstelle:*
  Der Benutzer soll aus einer Drop-down Liste der installierten Import - Schnittstellen wählen können.

- *Auswahl der Export Schnittstelle:*
  Der Benutzer soll aus einer Drop-down Liste der installierten Export - Schnittstellen wählen können.

- *Benachrichtigung wenn ein Code Element keine Verknüpfung zugewiesen bekommen hat:*
  Der Benutzer soll auswählen können, ob er beim Export benachrichtigt werden will, wenn ein Code Element keine Verknüpfung zu einer Anforderung hat. Dies soll mit einer CheckBox umgesetzt werden.

- *Export der Verknüpfungsinformationen bei jedem Speichern:*
  Der Benutzer soll auswählen können ob die Verknüpfungsinformationen bei jedem Speichern exportiert werden oder ob er dies manuell vornehmen will. Dies soll mit einer CheckBox umgesetzt werden.

- *Automatische Konsistenzprüfung:*
  Der Benutzer soll auswählen können ob die Konsistenzprüfung automatisch bei jedem Öffnen einer Code Datei durchgeführt wird. Dies soll mit einer CheckBox umgesetzt werden.

3.1.2 Importieren der Anforderungen

Die Liste der Anforderungen soll automatisch beim Öffnen von Eclipse aus dem in den Projekteinstellungen definierten Anforderungsmanagement Tool importiert werden. Wenn der Benutzer Eclipse startet und eine Code Datei eines Projektes öffnet und auch das Fenster des RT-Plugins (Requirement Tracing View) geöffnet hat sollen diese Anforderungen importiert und angezeigt werden.

- *Definieren der schnittstellenspezifischen Projekteinstellungen:*
  Der Benutzer soll in den Projekteinstellungen die schnittstellenspezifischen Einstellungen vornehmen können. Dafür sollen zwei Register in den Projekteinstellungen erstellt werden. Ein Register für die Import Schnittstelle und ein Register für die Export Schnittstelle. Auf diesen Registern werden alle schnittstellenspezifischen Felder angezeigt. Am Beispiel von der Schnittstelle für RequisitePro, kann der Benutzer die Projektdatei des RequisitePro Projektes auswählen, den Benutzernamen und das Passwort definieren. Weiters kann er aus der Liste der verfügbaren Anforderungstypen die zu Importierenden auswählen.

- *Der Import der Anforderungen soll ohne öffnen des Anforderungsmanagement Tools erfolgen:*
  Die Anforderungen sollen direkt von dem Anforderungsmanagement Tool importiert werden ohne das der Benutzer zuvor das Anforderungsmanagement Tool öffnen und dort Daten manuell Exportieren muss.

3.1.3 Anzeigen der Anforderungen

Die Anforderungen sollen in einem eigenen, integrierten Fenster / Register in Eclipse angezeigt werden. Die Anforderungen sollen in einer Baumstruktur dargestellt werden, wobei die Anforderungstypen die Hauptkategorisierung vornehmen.

- *Sortieren der Anforderungen nach deren Typ:*
  Die Anforderungen sollen deren Typ entsprechend sortiert werden. Dazu wird für jeden Typ ein Hauptknoten in der Baumstruktur erstellt. Die zugehörigen Anforderungen werden als Kindelemente hinzugefügt.

- *Anzeigen des Anforderungsprefix und der Anforderungsbezeichnung:*
  In der Liste sollen der Präfix und die Bezeichnung der Anforderungen angezeigt werden.

- *Anzeigen der Anforderungsbeschreibung:*
  Die Anforderungsbeschreibung jeder Anforderung soll als Kindelement angezeigt werden, so das man diese, wenn gewünscht, ausklappen kann.

- *Aktualisieren des Fensters / Registers:*
  Der Benutzer soll die Möglichkeit haben das Fenster / Register und somit die Liste der Anforderungen zu aktualisieren. Dies wird benötigt wenn sich die Anforderungen in dem Anforderungsmanagement Tool nach dem Öffnen von Eclipse geändert haben. Dazu soll ein Button in der Toolleiste des Fensters / Registers erstellt werden.

- *Starten der Konsistenzprüfung:*
  Der Benutzer soll die Möglichkeit haben die Konsistenzprüfung über einen Button in der Toolleiste des Fensters / Registers manuell zu starten.

- *Starten des Exports:*
  Der Benutzer soll die Möglichkeit haben den Export der Verknüpfungsinformationen über einen Button in der Toolleiste des Fensters / Registers manuell zu starten.

3.1.4 Verknüpfen der Anforderungen mit den Code Elementen

Wenn sowohl ein Editor mit einem Java Code und das RT-Plugin-Fenster geöffnet ist, soll der Benutzer die Anforderungen mit den Code Elementen verknüpfen können. Er soll dazu nur den Cursor in den Code Bereich der gewünschten Methode oder Klasse positionieren. Danach nur durch einen Doppel-Klick auf die zu verknüpfende Anforderung in der Liste des RT-Plugin-Fensters die Verknüpfung herstellen können. Die Information zu der Anforderung wird dann automatisch als Kommentar in den JavaDoc Bereich der jeweiligen Methode oder Klasse eingetragen. Die Information wird in folgendem Format eingetragen: „@requirement (ID) Prefix:Bezeichnung“.

3.1.5 Exportieren der erstellten Verknüpfungen

Wenn der Benutzer die Code Datei speichert oder manuell den Export Vorgang startet, sollen die Verknüpfungsinformationen exportiert und automatisch in das in den Projekteinstellungen definierte Anforderungsmanagement Tool importiert werden. (In RequisitePro sollen diese Informationen in einer Traceability Matrix angezeigt werden).

Die folgenden technischen Anforderungen beziehen sich auf die Schnittstelle zu RequisitePro.

- *Erstellen eins „Source Code“ Package in RequisitePro:*
  Es soll ein Package mit dem Namen „Source Code“ in RequisitePro erstellt werden, in dem alle importierten Verknüpfungsinformationen abgelegt werden. Dieses Package soll beim Import Vorgang automatisch erstellt werden.

- *Für die Methoden und Klassen sollen eigene Typen angelegt werden:*
  Es soll für die Methoden der Anforderungstyp „SRCM“ und für die Klassen der Anforderungstyp „SRCC“ in RequisitePro angelegt werden. Diese Typen sollen automatisch beim Import Vorgang erstellt werden.

- *Erstellen einer Anforderung für jedes Code Element:*
  Es soll für jede Methode und Klasse eine Anforderung angelegt werden, die diese repräsentieren.

- *Erstellen der Traceability Views:*
  In RequisitePro sollen für jeden Anforderungstyp je zwei Traceability Views erstellt werden. Eine zeigt die Verknüpfungsinformationen zwischen dem Anforderungstyp und den Methoden. Die Zweite zeigt die Verknüpfungsinformationen zwischen den Anforderungstyp und den Klassen.

- *Aktualisieren der Verknüpfungsinformationen:*
  Wenn die Verknüpfungsinformationen in Eclipse geändert wurden, sollen diese auch in RequisitePro aktualisiert werden. Dazu sollen alle zu diesen Code Elementen bestehenden Verknüpfungen gelöscht und die neuen erstellt werden.

- *Zeige einen ProgressDialog während dem Export Vorgang:*
  Während des Export Vorganges soll ein ProgressDialog gezeigt werden, der den Fortschritt des Vorganges anzeigt.

3.1.6 Automatische Konsistenzprüfung

Je nach den Projekteinstellungen soll eine Konsistenzprüfung bei jedem Öffnen einer Code Datei automatisch durchgeführt werden. Dazu werden die Anforderungen und die Verknüpfungsinformationen überprüft. Der Benutzer bekommt eine Benachrichtigung über die aufgetretenen Änderungen und kann wählen, welche er annimmt und welche er verwirft.

- *Wurde eine Anforderung gelöscht oder umbenannt:*
  Wenn der Code Editor geöffnet wird soll der Code nach den Anforderungseinträgen durchsucht und mit den importierten Anforderungen aus dem Anforderungsmanagement Tool verglichen werden. Wenn eine Anforderung umbenannt oder gelöscht wurde soll der Benutzer benachrichtigt werden. Der Benutzer kann entscheiden ob er diese Änderungen annimmt. Wenn ja, dann werden die Einträge im Code angepasst.

- *Wurden Verknüpfungen geändert:*
  Ebenso soll der Code nach bestehenden Verknüpfungen durchsucht werden, welche mit den Verknüpfungsinformationen aus dem Anforderungsmanagement Tool verglichen werden. Gefundene Änderungen werden dem Benutzer angezeigt, die er dann annehmen kann. Die neuen oder gelöschten Verknüpfungen werden automatisch aktualisiert.

3.2 Architektur

In diesem Abschnitt wird die Architektur des RT-Plugins näher beschrieben.

Da es viele verschiedene Anforderungsmanagement Tools gibt war es das Ziel das RT-Plugin so zu entwickeln, dass es mit möglichst allen Tools zusammenarbeiten kann. Um dies zu realisieren wurde das RT-Plugin mit einer Schnittstelle ausgestattet, über die man es einfach erweitern kann. So kann ein Entwickler für jedes Anforderungsmanagement Tool eine Erweiterung zu dem RT-Plugin entwickeln, das die Importdaten bereitstellt und die Exportdaten individuell verarbeitet. Durch diese Architektur ist das RT-Plugin vollkommen unabhängig von den Anforderungsmanagement Tools und muss nicht ständig den veränderten Anforderungsmanagement Tools angepasst werden. Die Anpassung beschränkt sich so auf die jeweilige Erweiterung des Anforderungsmanagement Tools.

Die folgende Abbildung 3.1 zeigt einen Überblick über diese Architektur.

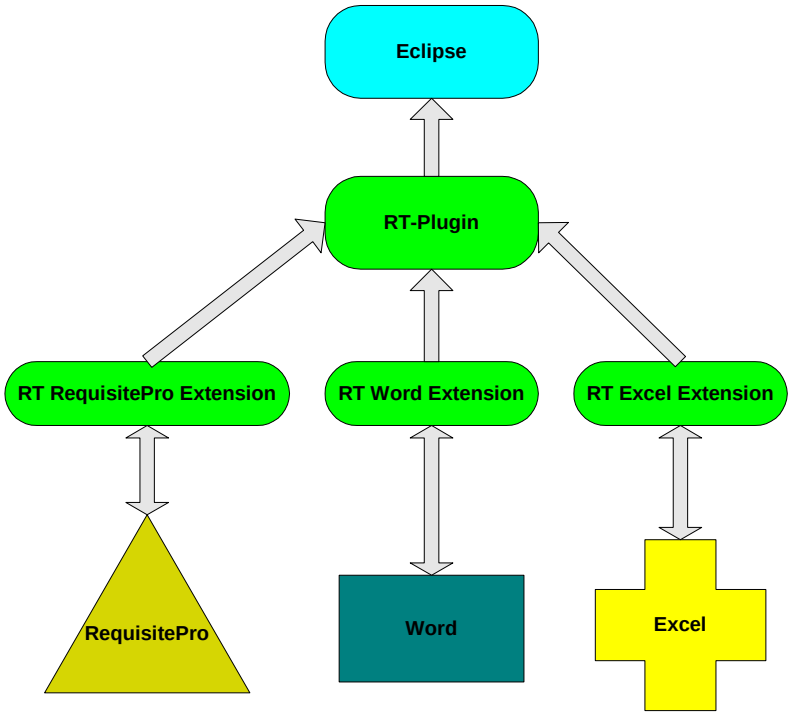

Abbildung 3.1: Architektur des RT-Plugins

Da Eclipse auf einer Architektur basiert, die aus einem kleinen Hauptmodul besteht und die komplette Funktionalität durch viele Plugins realisiert wird, war es leicht das RT-Plugin in Eclipse zu integrieren. Eclipse hat mehrere „Extension Points“ (Schnittstellen) definiert über die man die Funktionalität von Eclipse erweitern kann. Das RT-Plugin verwendet folgende „Extension Points“:

- *org.eclipse.ui.views:*
  Dieser Extension Point wird verwendet um ein Fenster / Register in Eclipse einzufügen, das im Weiteren dann die Anforderungen anzeigt.

- *org.eclipse.ui.propertyPages:*
  Dieser Extension Point wird verwendet um die projektspezifischen Einstellungen in Eclipse zu erweitern.

Das RT-Plugin bedient sich ebenso dieser Technik und definiert den Extension Point „rtextension“ über den man das RT-Plugin mit mehreren Schnittstellen zu den Anforderungsmanagement Tools erweitern kann. Diese Schnittstelle und wie man diese verwendet wird im Appendix B genauer beschrieben.

3.3 Funktionsweise

In diesem Abschnitt wird die Funktionsweise des RT-Plugins genau beschrieben. Wie werden die Einstellungen vorgenommen? Welche Funktionen hat das RT-Plugin-Fenster/Register und wie kann der Benutzer die Verknüpfungen zwischen den Anforderungen und den Code Elementen (Methoden und Klassen) erstellen? Die Funktionsweise wird im Appendix A an Hand eines Beispiels noch genauer demonstriert.

3.3.1 Projekteinstellungen

Das RT-Plugin bietet dem Benutzer mehrere Einstellungsmöglichkeiten, mit denen er die Funktions- und Arbeitsweise des RT-Plugins an seine Bedürfnisse anpassen kann. Nach dem Öffnen von Eclipse und des gewünschten Projektes können die Einstellungen vorgenommen werden. Dazu kann er das Dialogfenster der Projekteinstellungen durch markieren des Projektes im Navigator-Fenster und klicken der rechten Maustaste, über das Popup Menü, öffnen.

Die folgende Abbildung 3.2 zeigt dieses Dialogfenster.

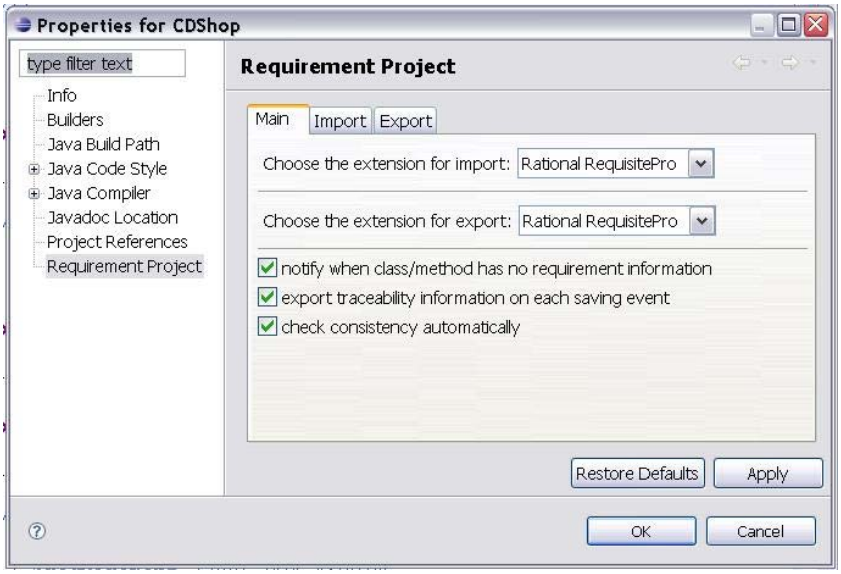

Abbildung 3.2: Dialogfenster der Projekteinstellungen

Nach Auswahl des „Requirement Project“ Eintrags auf der linken Seite des Dialogfensters können nun die Einstellungen für das RT-Plugin vorgenommen werden. Das Dialogfenster zeigt drei Register.

Im Register „Main“ werden die grundlegenden Einstellungen des Plugins festgelegt. Dazu gehört die Auswahl der Import- und Export- Erweiterung. Weiters kann der Benutzer das Handling des Plugins anpassen. Er kann festlegen, ob er beim Export der Verknüpfungsinformationen darüber informiert werden möchte, wenn eine Methode oder Klasse zum Zeitpunkt des Exports keine Verknüpfungsinformationen aufweist. Weiters kann der Benutzer festlegen, ob bei jedem Speichern der Code Datei der Export automatisch gestartet werden soll oder ob er den Export lieber manuell startet. Dann kann der Benutzer noch einstellen, ob bei jedem Öffnen einer Code Datei die Konsistenzprüfung durchgeführt werden soll oder ob er dies ebenfalls manuell durchführen will.

In den Registern „Import“ und „Export“ können die erweiterungsspezifischen Einstellungen für den Datenimport und den Datenexport vorgenommen werden.

3.3.2 Requirement Tracing View

In diesem Abschnitt wird das Fenster/Register des RT-Plugins, welches die Liste der Anforderungen zeigt, näher beschrieben. Das Fenster/Register kann der Benutzer über das Menü *Window / Show View / Other* und der Auswahl der Requirement Tra-

cing View im Paket Requirement Tracing öffnen. Das Fenster/Register befindet sich dann meist im unteren Bereich der Eclipse IDE und kann mit der Maus auf die rechte Seite neben dem Code Editor verschoben werden.

Die folgende Abbildung 3.3 zeigt dieses Fenster/Register.

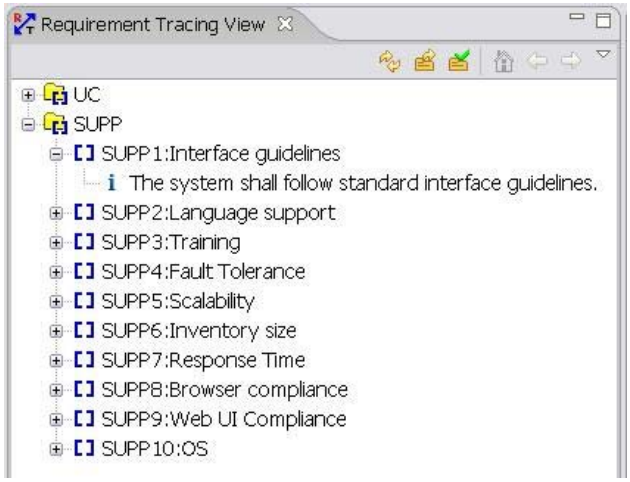

Abbildung 3.3: Requirements Tracing View

Die Abbildung 3.3 zeigt sehr schön wie die Liste der Anforderungen abgebildet wird. Die Abbildung der Anforderungen erfolgt in einer Baumstruktur, wobei die Ordner den Anforderungstypen entsprechen. In diesen Ordnern werden dann die Anforderungen im Format „Prefix:Name“ angezeigt. Jede Anforderung hat noch ein Kindelement, das die Beschreibung zeigt.

In der Toolbar des Requirement Tracing Views sind drei Buttons dargestellt, die folgende Funktionalität haben.

Mit dem Button kann das Fenster/Register aktualisieren werden, d.h. die Anforderungen neu laden und die Anzeige aktualisieren. Dies wird benötigt, wenn während der Arbeit in Eclipse im Anforderungsmanagement Tool die Anforderungen bearbeitet werden. Die neuen Anforderungen werden aus dem Anforderungsmanagement Tool geladen und angezeigt.

Mit dem Button kann der Export Vorgang manuell gestartet werden. Der Benutzer kann den Export Vorgang immer dann starten, wenn er Änderungen vorgenommen hat.

Mit dem Button kann der Benutzer die Konsistenzprüfung manuell durchführen. Dies macht allerdings nur dann Sinn, wenn noch keine neuen Verknüpfungen zwischen den Anforderungen und den Methoden oder Klassen erstellt wurden und auch der Export Vorgang noch nicht durchgeführt wurde.

3.3.3 Erstellung der Verknüpfungen

In diesem Abschnitt wird beschrieben wie eine Verknüpfung zwischen einer Anforderung und einer Klasse oder Methode erstellt wird.

Voraussetzung für die Erstellung einer Verknüpfung ist, dass der Benutzer eine Klasse mit dem JavaEditor von Eclipse geöffnet hat. Weiters muss das Requirement Tracing View Fenster des RT-Plugins geöffnet und in den Projekteinstellungen die Import- und Export Erweiterungen mit deren spezifischen Einstellungen korrekt vorgenommen sein. Wenn alle Einstellungen vorgenommen und alle angeführten Bedingungen erfüllt sind, dann wird die Liste der Anforderungen in dem Fenster/Register angezeigt.

Die folgende Abbildung 3.4 zeigt Eclipse mit einer geöffneten Klasse und dem geöffneten Fenster/Register.

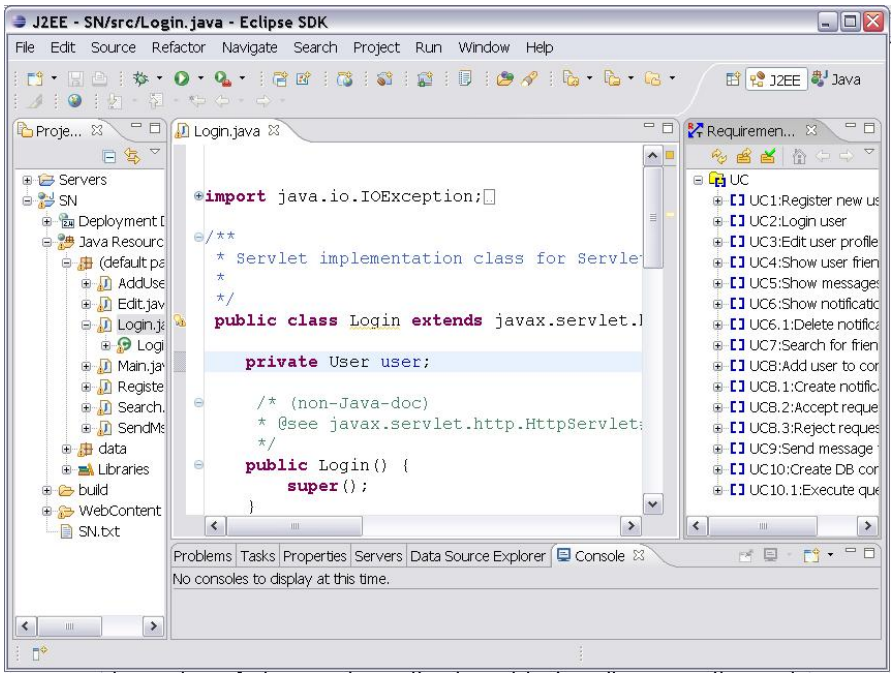

Abbildung 3.4: Eclipse mit Klasse und RT-Fenster

Um eine Verknüpfung zwischen einer Methode und einer Anforderung zu erstellen, muss der Benutzer nur den Cursor in den Bereich der Methode (zwischen den zwei geschwungenen Klammern der Methode) positionieren und mit der Maus auf die gewünschte Anforderung doppelt klicken. Dann wird automatisch eine Verknüpfung erstellt. Dazu wird in den JavaDoc Bereich der Methode ein Eintrag mit der Anforderungsinformation eingefügt.

Die folgende Abbildung 3.5 zeigt den Zustand nach dem Erstellen einer Verknüpfung.

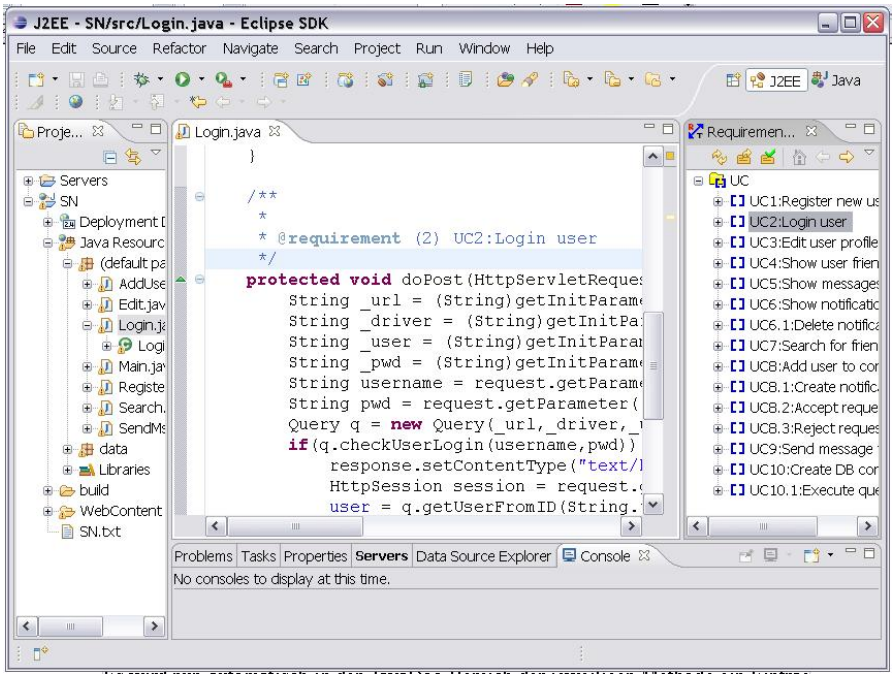

Abbildung 3.5: Eclipse mit Klasse und Verknüpfung

Wie in der Abbildung 3.5 ersichtlich ist wurde in den JavaDoc Bereich der Methode `doPost()` die Information der Anforderung eingefügt. Beim nächsten Export wird die Verknüpfung in das ausgewählte Anforderungsmanagement Tool exportiert und kann dort in einer Übersichtsmatrix angezeigt werden.

Eine ausführliche Beschreibung aller Funktionalitäten des RT-Plugins an Hand eines Beispiels findet sich in Kapitel 7.

4 Forschungsfragen

In diesem Abschnitt möchte ich auf die Forschungsfragen eingehen.

Welche Zusammenhänge gibt es?

Welche Vorteile bringt das RT-Plugin?

Welches Verbesserungspotential gibt es?

4.1 Faktoren des Requirement Tracings

Wie schon erwähnt, stellt die Durchführung des Requirement Tracings nicht nur eine Verbesserung der Projektübersicht und Projektplanung für den Projektleiter dar sondern ist vor allem für spätere Aufgaben wie die Change Impact Analyse von großer Bedeutung. In der Change Impact Analyse werden die von einer Änderungsanfrage betroffenen Artefakte gesucht und zusammengestellt um den Aufwand der Umsetzung dieser Änderungsanfrage besser abzuschätzen und kalkulieren zu können.

In der Abbildung 4.1 werden die Faktoren, die das Requirements Tracing beeinflussen, dargestellt [13].

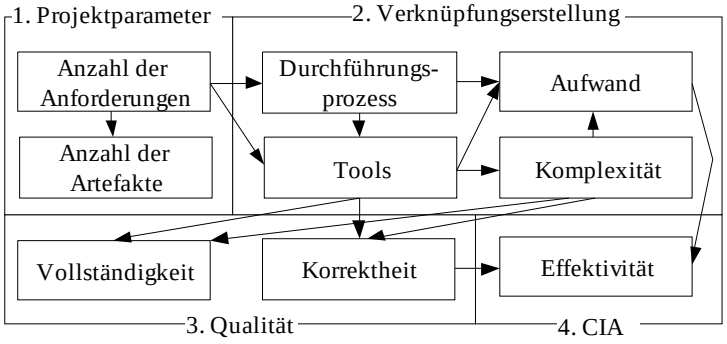

Abbildung 4.1: Faktoren die das Requirements Tracing beeinflussen

Die Faktoren werden in mehrere Blöcke zusammengefasst.

Im Block *Projektparameter* (1) werden die projektspezifischen Parameter zusammengefasst. Dazu zählt die Anzahl der Anforderungen, welche den generellen Umfang des Projektes beeinflusst. Diese kann auch die Anzahl der Artefakte mit beeinflussen. Umso größer die Anzahl der Anforderungen umso größer ist auch die Anzahl der Klassen und somit auch die Anzahl der Verknüpfungen, die erstellt werden müssen. Auf Grund der Anzahl der Anforderungen wird die Art der Umsetzung (Durchführungsprozess) und die Tools, die verwendet werden, beschlossen. Wenn die Anzahl der Anforderungen gering ist, dann zahlt es sich nicht aus ein umfangreiches Anforderungsmanagement Tool zu verwenden, dies würde zu einem unnötigen Overkill führen. Ebenso kann auf die Durchführung des Requirements Tracings in einem kleinen Projekt verzichtet werden, da der Aufwand in keiner Relation zum Nutzen stehen würde.

Im Block *Verknüpfungserstellung* (2) werden die Faktoren zusammengefasst, die die Qualität der Verknüpfungen beeinflussen. Im Durchführungsprozess wird festgelegt, wann die Verknüpfungen erstellt werden sollen. Sollen diese während der Implementierung oder erst immer im Nachhinein, nach Fertigstellung eines (Teil-)Paketes, erstellt werden. Beim Einsatz eines integrierten Tools, wie zum Beispiel das RT-Plugin, können die Verknüpfungen gleich während der Implementierung vom Entwickler erstellt werden. Dies führt unweigerlich zu einer höheren Qualität der erstellten Verknüpfungen. Qualität bedeutet, dass die erstellten Verknüpfungen vollständig und auch korrekt sind. Werden die Verknüpfungen erst im Nachhinein erstellt, so kann es vorkommen, dass der Entwickler auf die eine oder andere Abhängigkeit zwischen Artefakten vergisst und die Vollständigkeit darunter leidet. Wenn der Aufwand für die Erstellung der Verknüpfungen zusätzlich noch verringert und erleichtert wird, dann führt dies zu einer höheren Motivation des Entwicklers diese Verknüpfungen gleich bei der Implementierung zu erstellen und zu warten wodurch die Qualität ebenfalls steigt. Einfachere und weniger komplexe Tools führen zusätzlich zu einer Verringerung des Aufwands für den Entwickler.

Im Block *Qualität* (3) sind die zwei wichtigsten Faktoren der Qualität der erstellten Verknüpfungen angeführt, Vollständigkeit und Korrektheit. Vollständigkeit bedeutet, dass alle existierenden Abhängigkeiten zwischen allen Artefakten erstellt wurden. Es wurden alle Abhängigkeiten zwischen den Anforderungen und den Klassen und Methoden des Source Codes erstellt. Korrektheit bedeutet, dass alle erstellten Verknüpfungen auch tatsächlich auf eine Abhängigkeit zwischen zwei Artefakten beruhen. Dies bedeutet, wenn es eine Verknüpfung zwischen einer Anforderung und

einer Klasse oder Methode besteht, dass in der jeweiligen Klasse oder Methode auch diese Anforderung zumindest teilweise implementiert wurde. Umso höher die Vollständigkeit und die Korrektheit der erstellten Verknüpfungen ist, umso höher ist somit auch der Wert der erstellten Verknüpfungen für die Durchführung einer Change Impact Analyse. Der Projektleiter kann somit den erstellten Verknüpfungen vertrauen und leichter alle möglichen, von der Änderungsanfrage betroffenen Artefakte finden.

Aus diesen Zusammenhängen ergibt sich, dass es bei einem integrierten, plugin–basierten Ansatz, der die Komplexität der Erstellung der Verknüpfungen und auch die Zeitspanne zwischen Erstellung der Artefakte und der Verknüpfungen reduziert, auch den durchschnittlichen Aufwand für die Verknüpfungserstellung reduziert. Dies führt zu einer Erhöhung der Effektivität und zu einer höheren Akzeptanz und Motivation des Entwicklers. Eine höhere Effektivität reduziert wiederum das Risiko falsche und nicht vollständige Verknüpfungen zu erhalten.

4.2 Ansätze des Requirement Tracings

Wie schon in Abschnitt 2 erwähnt gibt es mehrere Ansätze für die Erstellung der Verknüpfungen. Da die automatisierte Erstellung noch nicht den Weg in die Praxis gefunden hat, werden die Verknüpfungen vor allem manuell mit Hilfe der verschiedensten Tools erstellt.

Die manuellen Ansätze können weiter in zwei Typen eingeteilt werden:

- *Systematische Ansätze:*
  Die Anforderungen werden in einer Datenbank, zum Beispiel mit RequisitePro, gespeichert und verwaltet. Um eine Verknüpfung einer Anforderung mit einem Source Code Element zu erstellen muss zuerst ein Objekt in RequisitePro erstellt werden, welches die jeweilige Klasse oder Methode darstellt. Danach können in einer Matrix die Verknüpfungen manuell eingetragen werden. Da diese Verknüpfungen in einem für den Entwickler zusätzlichen Tool erstellt werden müssen und dies auch besonders aufwändig ist, werden die Verknüpfungen in Intervallen, zum Beispiel immer nach Fertigstellung einer Komponente, erstellt. Dies führt zu einer höheren Zeitspanne zwischen Implementierung und der Erstellung der Verknüpfung.

- *Kontinuierliche Ansätze:*
  Wird für die Erstellung der Verknüpfungen ein integriertes Tool, wie das RT-Plugin, eingesetzt, so kann der Entwickler direkt in seiner gewohnten Entwicklungsumgebung die Verknüpfungen ohne hohem Zeitaufwand gleich während der Implementierung erstellen. Dies führt zu einer enormen Reduktion des Aufwands und führt zu einem positiven Einfluss auf die Vollständigkeit und Korrektheit der erstellten Verknüpfungen.
-

4.3 Forschungsgründe und -fragen

Wie schon in den letzten Abschnitten beschrieben ist der Aufwand für die Erstellung der Verknüpfungen von besonderer Bedeutung und beeinflusst indirekt die Effektivität bei der Durchführung einer Change Impact Analyse. Die zwei Ansätze für die Erstellung der Verknüpfungen (Systematische und Kontinuierliche) unterscheiden sich in deren Tools, die verwendet werden und im Zeitpunkt der Erstellung der Verknüpfungen.

Bei gegebenen Projektparametern (aus Abbildung 4.1), die Anzahl der Anforderungen, die Anzahl der Verknüpfungen und die daraus resultierende Anzahl der zu erstellenden Verknüpfungen, die für alle Ansätze die gleichen sind, ergeben sich die Forschungsfragen wie folgt:

1. In wie weit kann der plugin–basierte Ansatz den Aufwand für die Erstellung der Verknüpfungen im Vergleich zum systematischen Ansatz, reduzieren?

Ein Vorteil des RT-Plugins im Vergleich zu dem systematischen Ansatz mit RequisitePro ist, dass die Objekte, die die Source Code Elemente darstellen nicht manuell erstellt werden müssen. Dies wird vom RT-Plugin automatisch durchgeführt. Wird Excel als Tool verwendet, so muss der jeweilige Klassen und Methodenname manuell in den spalten eingetragen werden. Dabei können leicht Tippfehler auftreten.

2. In wie weit kann der reduzierte Aufwand und die reduzierte Zeitspanne zwischen der Implementierung und der Erstellung der Verknüpfungen des plugin–basierten Ansatzes die Vollständigkeit und die Korrektheit der erstellten Verknüpfungen im Vergleich zu systematischen Ansätzen verbessern?

Auf Grund des reduzierten Aufwands und der Einfachheit der Verknüpfungserstellung kann der Entwickler mehr Verknüpfungen erstellen und somit die Vollständigkeit erhöhen. Bezüglich der Korrektheit liegt die einzige Fehlerquelle beim Entwickler selbst, der eine falsche Anforderung selektieren kann. Im Vergleich bietet der systematische Ansatz mehr mögliche Fehlerquellen, wie zum Beispiel Tippfehler oder Navigationsfehler. Durch den geringeren Aufwand und die höhere Akzeptanz und Motivation der Entwickler beim Einsatz des RT-Plugins sollte dies zu einer höheren Vollständigkeit und Korrektheit der erstellten Verknüpfungen führen.

3. Ist das RT-Plugin tauglich für den Einsatz in der Praxis?

Weiters soll überprüft werden ob das RT-Plugin praxistauglich ist und auch in realen Projekten eingesetzt werden kann. Dazu sollen Praktiker der Siemens PSE das RT-Plugin evaluieren und die praxistauglichkeit bewerten.

5 Evaluierung

In diesem Abschnitt werden die durchgeführte Evaluierung und ein Konzept für eine spätere und detailliertere Evaluierung im Rahmen eines Fallstudienprojektes in der Siemens PSE beschrieben.

5.1 Erstevaluierung

Bei der Erstevaluierung wurde die Erstellung der Verknüpfungen zwischen den Anforderungen und den Sourcecode Elementen betrachtet. Dabei soll der pluginbasierte mit den manuellen Ansätzen verglichen werden. Als Repräsentatoren für die manuellen Ansätze wurden die Tools RequisitePro und Excel, die sehr oft in der Praxis verwendet werden, herangezogen. Dabei wurden in einem kleineren Projekt die Verknüpfungen mit jedem der drei Tools erstellt und die Aufwände bei der Erstellung gemessen.

Das Projekt umfasste 19 Anforderungen. Der Sourcecode bestand aus 10 Klassen und 78 Methoden. Gemessen wurde der Aufwand für die Vorbereitung, die Erstellung der Verknüpfungen und für die Erstellung der Übersichtsmatrix.

Dabei waren je Tool folgende Aufgaben durchzuführen:

Vorbereitungen

Zu den Vorbereitungen zählen alle Aufgaben, die durchgeführt werden müssen um mit dem Erstellen der Verknüpfungen beginnen zu können. Bei dem RT-Plugin mussten hierfür die Projekteinstellungen vorgenommen werden. Dazu zählt das Auswählen des RequisitePro Projektes und der Anforderungstypen, die importiert werden sollen. Bei RequisitePro musste hierfür ein Package und zwei Anforderungstypen für die Klassen und Methoden angelegt werden. Für Excel musste nur die Tabelle mit den Anforderungen, die schon in einer Tabelle vorhanden sind, geöffnet werden.

Erstellen der Verknüpfungen

Hierzu zählen alle Tätigkeiten, die für das Erstellen einer Verknüpfung notwendig sind. Bei dem RT-Plugin beschränkt sich die Tätigkeit auf das Positionieren des Cursors im Methodenbereich, das Auswählen der Anforderung im Requirements Tracing View und dem Doppelklick um die Verknüpfung zu erstellen. In RequisitePro muss hierfür ein Element, welches die Methode oder Klasse darstellt, angelegt werden. Zusätzlich muss die Verknüpfung entweder in den Einstellungen der jeweiligen Anforderung erstellt oder in einer im Voraus erstellten Matrix eingetragen werden. In Excel muss für das Erstellen einer Verknüpfung der Methoden- oder Klassenname in einer Spalte eingetragen werden. Danach muss in der richtigen Zelle, in der sich die Reihe der jeweiligen Anforderung und die Spalte der Methode oder Klasse kreuzen, eine Markierung gesetzt werden.

Erstellen der Übersichtsmatrix

Die Übersichtsmatrix ist besonders wichtig, da diese einen schnellen Überblick verschafft. Bei dem RT-Plugin sind hierfür keine zusätzlichen Tätigkeiten erforderlich, da diese beim Export der Verknüpfungen automatisch erstellt werden. In RequisitePro muss hierfür eine View erstellt werden, die einiger Einstellungen bedarf. In Excel fallen hier ebenfalls keine zusätzlichen Aufgaben an, da das Erstellen der Verknüpfungen in einer Matrix geschieht.

5.2 Feedback von Praktikern

Um das RT-Plugin nicht nur in der Theorie zu testen und zu beschreiben sondern auch einen Bezug zur Praxis zu bekommen, wurde das Plugin von Requirements Engineering Experten der Siemens PSE CSS, wie es in Abschnitt 3 beschrieben ist, evaluiert und begutachtet. Zu den Experten zählten Projektleiter, Analysten und Entwickler. Ziel war es, ein fundiertes Feedback über die Funktionalität und mögliche Schwächen des RT-Plugins zu erhalten um es in weiterer Folge zu verbessern und den Bedürfnissen der Praxis anzupassen.

5.3 Evaluierungskonzept für eine Fallstudie

Dieser Abschnitt beschreibt ein Evaluierungskonzept nach dem das RT-Plugin im Rahmen einer detaillierten Fallstudie bei Siemens PSE evaluiert und mit anderen Ansätzen verglichen werden soll. Das Konzept wurde unter Berücksichtigung der im ViSEK[7] Report angeführten Richtlinien erstellt.

5.3.1 Zieldefinition

Ziel einer umfassenden Fallstudie ist es die in Abschnitt 3.4 aufgestellten Forschungsfragen zu beantworten.

1. In wie weit kann der plugin–basierte Ansatz den Aufwand für das Tracing im Vergleich zu anderen Ansätzen, reduzieren?

Durch den Vergleich des plugin-basierten Ansatzes mit anderen Ansätzen des Requirements Tracing soll erhoben werden ob sich durch das RT-Plugin der Tracing Aufwand im Vergleich deutlich reduzieren lässt.

2. In wie weit kann der reduzierte Aufwand und die reduzierte Zeitspanne zwischen der Implementierung und der Erstellung der Verknüpfungen des plugin–basierten Ansatzes die Vollständigkeit und die Korrektheit der erstellten Verknüpfungen im Vergleich zu andern Ansätzen verbessern?

Zusätzlich soll erhoben werden ob sich durch die einfache Handhabung des RT-Plugins und die Reduktion des Tracing Aufwands der Grad der Vollständigkeit und der Korrektheit der erstellten Verknüpfungen erhöht. Der Schlüssel dazu liegt unter anderem in der Bereitschaft und der Motivation der Entwickler. Ist die Motivation der Entwickler hoch, ist auch der Grad der Vollständigkeit und der Korrektheit hoch.

3. Kann das RT-Plugin mit seiner Funktionalität und des Reduzierten Aufwands die Motivation der Entwickler steigern?

Stört das Tracing den ursprünglichen Prozess der Entwickler (die Implementierung) nicht oder nur geringfügig, so ist der Entwickler eher bereit die Verknüpfungen zu erstellen als wenn er in ein anderes Tool wechseln muss und in seiner Arbeit unterbrochen wird.

Um eine Antwort auf diese Fragen zu erhalten, muss das RT-Plugin in einem realen Projekt eingesetzt, genau untersucht und mit anderen Ansätzen verglichen werden.

Als Vergleichsobjekte werden die beiden gängigen Ansätze bei Siemens PSE mit den Tools RequisitePro und Excel heran gezogen. Es wird das Tracing in einem aussagekräfigen Projekt bei Siemens PSE eingesetzt. Als Vergleichswerte werden die schon vorhandenen Erfahrungswerte der Siemens PSE aus anderen Projekten mit den Tools RequisitePro und Excel herangezogen.

5.3.2 Projektauswahl

Um die gemessenen Werte besser mit den vorhandenen Erfahrungswerten aus andern Projekten vergleichen zu können sollte ein Projekt durchschnittlicher Größe (bei Siemens PSE) gewählt werden.

Gemessen wird dieses an folgenden Projektparametern:

- Anzahl der Anforderungen,
- Anzahl der Klassen und Methoden,
- Anzahl und Erfahrung der Teammitglieder

5.3.3 Messung

Die Messung unterteilt sich in einen quantitativen und einen qualitativen Teil.

Quantitative Messung

Während der Entwicklung soll der Tracing Aufwand gemessen werden. Dafür müssen die Entwickler ihre Aufwände niederschreiben. Nach Fertigstellung jeder Komponente soll die Vollständigkeit und die Korrektheit der erstellten Verknüpfungen stichprobenartig für mehrere Anforderungen überprüft werden.

Qualitative Messung

Bei der qualitativen Messung sollen subjektive Eindrücke des RT-Plugins festgehalten werden. Dazu soll beobachtet werden wann und wie genau die Entwickler die Verknüpfungen erstellen. Werden diese kontinuierlich während der Implementierung erstellt oder in Intervallen etwa immer erst am Wochenende oder nach Fertigstellung einer Komponente? Dies soll vom Leiter der Studie beobachtet und festgehalten werden.

Zusätzlich soll eine qualitative Bewertung des RT-Plugins durch Interviews durchgeführt werden. Hier soll festgehalten werden wie zufrieden die Entwickler und Projektleiter mit dem Plugin sind. In den Interviews sollen folgende Fragen gestellt werden:

Fragen an die Projektmanager:

- Hat das RT-Plugin Vorteile im Vergleich zu bisherigen Methoden des Requirements Tracing?
- Hat das RT-Plugin Vorteile in Bezug auf die Projektplanung und Projektübersicht? Wenn ja, welche?
- Beeinflusst das RT-Plugin die Durchführung einer Change Impact Analyse?
- Wie wurde das RT-Plugin ihrer Meinung nach von den Entwicklern angenommen und verwendet?

Fragen an die Entwickler:

- Wie gefiel das RT-Plugin?
- Waren die Funktionen des RT-Plugins handhabbar?
- Brachte das Plugin einen zusätzlichen Nutzen?
- Hat das Tracing durch das Plugin ihren Arbeitsablauf gestört?
- Wann haben sie die Traces erstellt?

5.3.4 Auswertung

Nach Fertigstellung des Projektes und der Studie, müssen die gemessenen Daten gesammelt und ausgewertet werden. Die Werte der quantitativen Messung müssen bereinigt werden um diese mit den Erfahrungswerten aus anderen Projekten vergleichen zu können. Die Ergebnisse der qualitativen Messung und deren Interviews sollen zusammengefasst werden und so eine abschießende Bewertung des RT-Plugins erstellt werden.

6 Ergebnisse und Diskussion

In diesem Abschnitt werden die Ergebnisse der Evaluierung dokumentiert und diskutiert.

6.1 Ergebnisse der Erstevaluierung

Die Durchführung der Erstevaluierung zeigte deutlich, dass der Aufwand für die Erstellung der Verknüpfungen mit Hilfe des RT-Plugins enorm reduziert werden kann. Der Aufwand verringert sich im Vergleich zu den beiden Tools RequisitePro und Excel auf ein Sechstel. Dieser Vorteil folgt aus Reduktion des Aufwands für die Erstellung einer Verknüpfung auf einen Doppelklick.

Umso größer die Anzahl der zu erstellenden Verknüpfungen umso größer wird der Vorteil gegenüber den beiden anderen Tools, deren Aufwand eine Verknüpfung zu erstellen um ein erhebliches größer ist. Auch die Vollständigkeit und Korrektheit kann durch den geringen Aufwand, der daraus resultierenden gesteigerten Motivation der Entwickler, positiv beeinflusst werden.

6.2 Feedback der Praktiker

Bei der Evaluierung des RT-Plugins durch die Requirements Engineering Experten der Siemens PSE CSS sind einige Verbesserungsvorschläge entstanden. In den nachfolgenden Teilabschnitten werden diese beschrieben und in Abschnitt 6.2.2 mit der bisherigen Umsetzung verglichen.

6.2.1 Verbesserungsvorschläge der Experten

Die Verbesserungsvorschläge der Experten sind nach den ursprünglichen Benutzeranforderungen aus Abschnitt 3.1 gegliedert.

6.2.1.1 Änderungen in der Architektur der Schnittstelle

- *Client – Server Architektur der Schnittstelle:*
  Da es nicht möglich war eine RequisitePro Projektdatei aus dem Netzwerk aus zu wählen soll die Schnittstelle zu RequisitePro in einer Client – Server Architektur umgesetzt werden. Dies ermöglicht das verteilte, gleichzeitige Arbeiten im Netzwerk.

6.2.1.2 Änderungen in den Projekteinstellungen

- *Definieren des Formates für den JavaDoc Eintrag:*
  Der Benutzer soll das Format des JavaDoc Eintrags definieren können. Dazu soll der Benutzer den Text des Eintrags mit den zwei Schlüsselwörtern „$ID“ und „$NAME“, die für die ID und den Namen der Anforderung stehen, beliebig gestalten können. Die Schlüsselwörter sollen auch mehrmals in beliebiger Reihenfolge eingetragen werden können. Es muss allerdings zumindest ein Zeichen vor, nach und zwischen den beiden Schlüsselwörtern stehen um diese auflösen zu können. Ein Beispiel für dieses Format wäre:
  „@see ($ID) $NAME“.
  Wenn in der erstellten JavaDoc in jeder verknüpften Methode ein Link zu der Beschreibung der jeweiligen Anforderung angezeigt werden soll, dann kann man dies durch Anpassung des Formates bewerkstelligt werden. So ist folgendes Format eine Möglichkeit dies umzusetzen:
  „@see $NAME“.
  Dadurch wird dann in der erstellten JavaDoc bei jeder verknüpften Methode oder Klasse ein Link mit dem Namen der Anforderung dargestellt, der dann eine Html-Seite mit der Anforderungsbeschreibung aufruft. Diese Anforderungsbeschreibungen müssen dann im Unterverzeichnis „./requirements“ mit ihrer ID als Namen und der Endung „.html“ abgelegt sein.

- *Definieren der Importeinstellungen für RequsitePro:*
  Der Benutzer soll in den Importeinstellungen für die RequisitePro Schnittstelle zusätzliche Einstellungen vornehmen können. Er soll die Serveradresse, und den Port angeben können, unter der die Serveranwendung der Schnittstelle läuft. Er soll das RequsitePro Projekt in einer Liste der verfügbaren Projekte auswählen können. Der Benutzer soll zusätzlich aus einer Liste der erstellten Views (in RequisitePro kann man die Anforderungen mit Hilfe von Views filtern) eine oder mehrere auswählen können. Zusätzlich soll der Benutzer die Möglichkeit haben, einen Link zu einer Webseite, auf der die Anforderungs-

beschreibungen gespeichert sind, zu definieren. Dieser Link soll dann in den JavaDoc Eintrag eingebunden werden um in der generierten JavaDoc einen Link zu den Anforderungsbeschreibungen darstellen zu können.

- *Definieren der Exporteinstellungen für RequisitePro:*
  Der Benutzer soll die Serveradresse und den Port angeben können, unter der die Serveranwendung der Schnittstelle läuft. Er soll das RequisitePro Projekt in einer Liste der verfügbaren Projekte auswählen können.

6.2.1.3 Änderungen in der Anzeige der Anforderungen

- *Anzeigen der Anforderungsbeschreibung:*
  Als Kindelement soll ein Link zu der Anforderungsbeschreibung angezeigt werden, so das der Benutzer durch einen Doppelklick die Anforderungsbeschreibung in einem Dialog öffnen kann. Der Dialog zeigt in einem Fenster ein mehrzeiliges Textfeld, welches die ausführliche Beschreibung darstellt. Zusätzlich werden in dem Dialog die Attribute und die Historie aller Änderungen der Anforderungen in Tabs dargestellt. Dies wird besonders dann notwendig wenn die Anforderungsbeschreibung ausführlicher ist und mehrere Zeilen umfasst. Im Dialog soll ein Link auf die in den Importeinstellungen des Projektes definierter Webseite, die eine ausführliche Anforderungsbeschreibung zeigt, dargestellt werden. Durch klicken auf diesen Link soll sich die Webseite in einem Browser öffnen. Zusätzlich soll es in dem Dialog eine Möglichkeit geben, die „Suspect“-Markierung der Anforderung zu löschen. Die „Suspect“-Markierung zeigt an, dass sich die Anforderung in RequisitePro geändert hat. Die Beschreibung zu der jeweiligen Anforderung soll immer erst beim Öffnen des Dialoges geladen werden um den Speicher zu entlasten.

- *Anzeigen der Klassen, die eine Verknüpfung zu der Anforderung hat:*
  Bei den jeweiligen Anforderungen sollen die Klassennamen jener Klassen als Kindelemente angezeigt werden, die eine Verknüpfung zu der Anforderung haben. Bei Doppelklick auf den Klassennamen soll die jeweilige Klasse im Editor geöffnet werden.

- *Anzeigen wenn sich eine Anforderung in RequisitePro geändert hat:*
  Hat sich in RequisitePro eine Anforderung geändert, die mit einem Sourcecode Element verknüpft ist, so soll dies durch ein anderes Icon der Anforderung dargestellt werden (die Anforderung wird als „Suspect“ markiert). So kann

der Entwickler sehen wenn sich eine Anforderung geändert hat und entsprechend reagieren. Es soll dem Benutzer auch möglich sein diese „Suspect"-Markierung zu löschen, wenn er die Änderungen gelesen und umgesetzt hat.

6.2.1.4 Änderungen beim Export

- *Attribut für Anforderungstypen SRCC und SRCM in RequisitePro:*
  Die erstellten Anforderungstypen in RequisitePro sollen ein Attribut „packagename" haben, in dem der Packagename der Klasse gespeichert wird. Zusätzlich soll dieser auch in der Beschreibung gespeichert werden.

6.2.2 Vergleich der umgesetzten Verbesserungsvorschläge

In diesem Abschnitt werden die umgesetzten Änderungen mit der vorherigen Version verglichen und deren Vorteile diskutiert.

6.2.2.1 Architektur der Schnittstelle zu RequisitePro

Die Schnittstelle zu dem Anforderungsmanagement Tool RequisitePro hat sich in der Architektur geändert. So wurde diese nun in einer Client-Server Architektur umgesetzt. Dazu wurde eine Serveranwendung entwickelt, die auf dem Server, auf dem auch das RequisitePro installiert ist und wo sich die Projektdatei befindet, läuft. Die Serveranwendung wartet auf Anfragen von den Clients, die auf den Rechnern der Entwickler laufen. Die Einstellungen der Serveranwendung können in Dateien festgelegt werden. So werden die für die Entwickler auswählbaren Projekte definiert.

Durch den Einsatz der Client-Server Architektur ist es möglich verteilt im Netzwerk zu arbeiten. Zusätzlich verringert es die Lizenzkosten erheblich, da diese nur für jede einzelne Anfrage berechnet werden.

6.2.2.2 Projekteinstellungen

In den Projekteinstellungen wurden einige Änderungen durchgeführt. So wurde im "Main" Tab der Projekteinstellungen ein Eingabefeld hinzugefügt, in dem das Format für den JavaDoc Eintrag definiert werden kann. Dies hat den Vorteil, dass das Format den jeweiligen Bedürfnissen angepasst werden kann. Dies ist besonders für die weitere Verarbeitung der Informationen notwendig. So kann in den JavaDoc Be-

reich ein Link zu einer Webseite oder zu einem Dokument, in dem die Anforderung beschrieben ist, eingefügt werden.

Abbildung 6.1 zeigt einen Vergleich.

Vorher: **Nacher:**

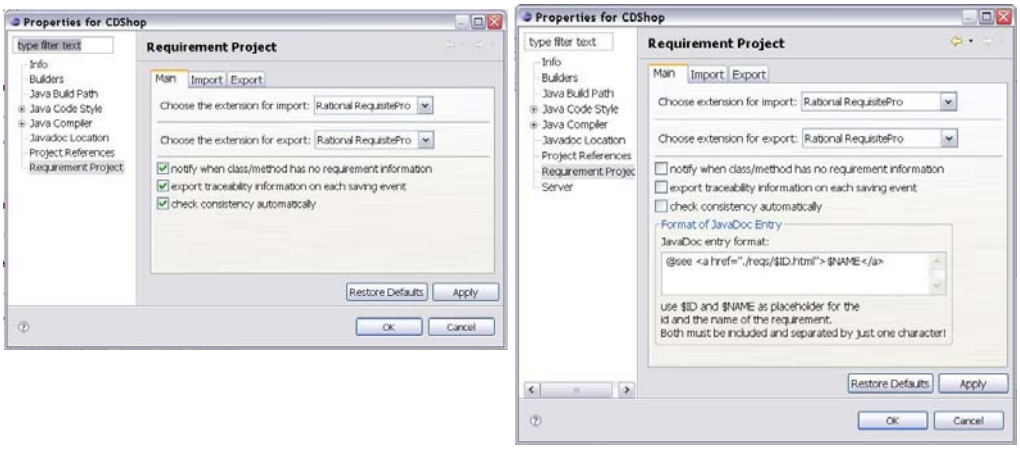

Abbildung 6.1: Vergleich Projekteinstellungen Main-Tab

Aber auch im “Import” -Tab wurden einige Änderungen vorgenommen. Auf Grund der Architekturänderung der Schnittstelle zu RequisitePro musste auch der Import-Tab angepasst werden. So werden nun Eingabefelder benötigt, in denen die Serverdaten definiert werden können. Statt dem Eingabefeld für die RequisitePro Projektdatei wird nun eine Liste mit verfügbaren Projekten, die in der Serveranwendung definiert werden, angezeigt. Aus dieser Liste kann der Benutzer dann eine Projektdatei auswählen. Zusätzlich soll der Benutzer auch eine oder mehrere Views auswählen können. Diese Views können in RequisitePro definiert werden. Mit Hilfe dieser Views können die Anforderungen gefiltert werden. Dadurch kann sich jeder Entwickler nur die Anforderungen importieren, die er für die Entwicklung benötigt und reduziert somit die Anzahl der aufgelisteten Anforderungen in seiner Entwicklungsumgebung. Dadurch kann der Entwickler die für ihn interessanten Anforderungen schneller finden.

Zusätzlich wurde auch noch ein Eingabefeld hinzugefügt, in dem der Benutzer die URL zu der ReqWeb Seite definieren. ReqWeb ist eine Erweiterung von RequisitePro, die die Anforderungsbeschreibungen in einer Webseite darstellt. Dieser Link kann ebenfalls als JavaDoc Eintrag verwendet werden und wird im Dialog der Anforderungsbeschreibung angezeigt.

Abbildung 6.2 zeigt den Vergleich.

Vorher: **Nachher:**

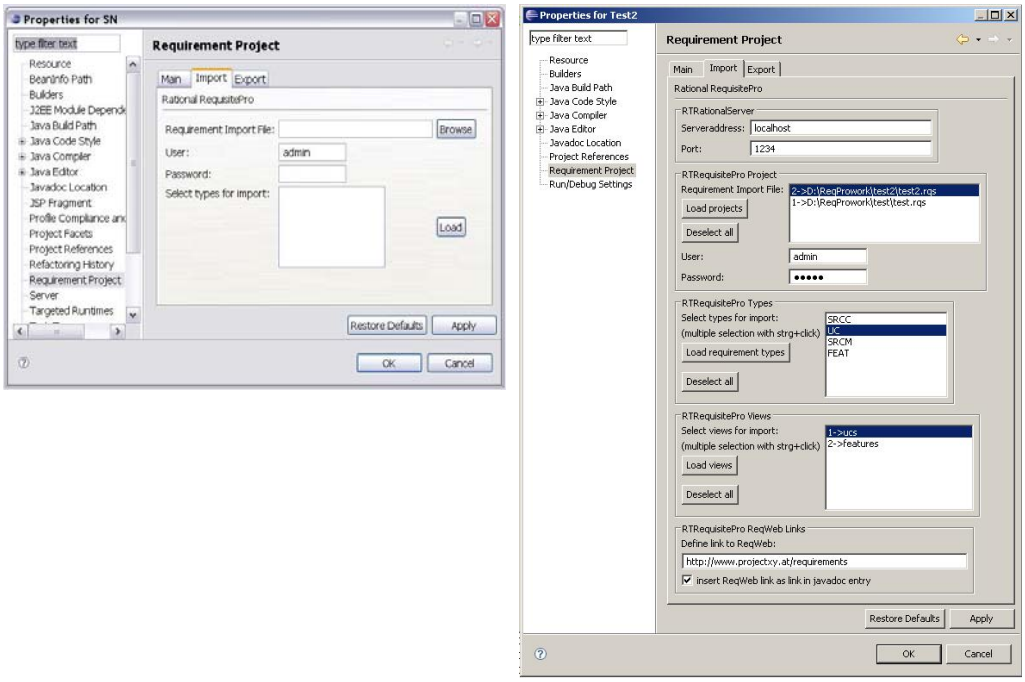

Abbildung 6.2: Vergleich Projekteinstellungen Import-Tab

Auf dem „Export“-Tab wurden auf Grund der Architekturänderung ebenfalls Eingabefelder für die Serverdaten hinzugefügt.

6.2.2.3 Anzeige der Anforderungen

Im Requirements Tracing View, in der die Anforderungen aufgelistet sind wurden Änderungen vorgenommen. So wird die Beschreibung der Anforderungen nicht mehr als Kindelement in der Baumstruktur dargestellt sondern stattdessen ein Link mit dem durch Doppelklick ein Dialog geöffnet werden kann. In diesem Dialog wird dann auf mehreren Tabs die ausführliche Information zu der jeweiligen Anforderung angezeigt. So wird nicht nur die ausführliche Beschreibung dargestellt sondern auch die Historie der Änderungen der Anforderung und deren Attribute angezeigt. Zusätzlich wird aus Gründen der Performanceverbesserung und Speicherentlastung die ausführliche Information zu den Anforderungen erst bei öffnen des Dialoges geladen.

Die folgende Abbildung 6.3 zeigt diesen Dialog.

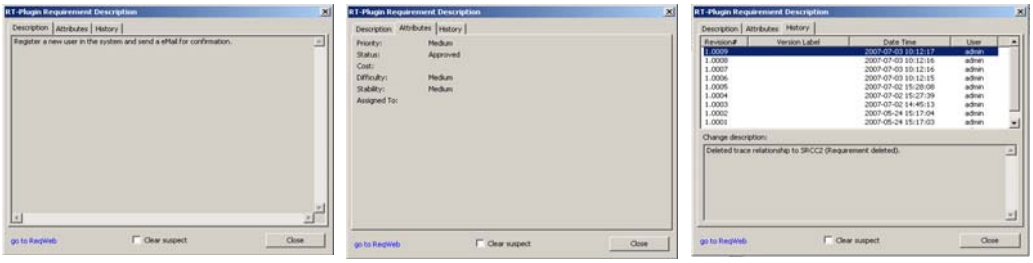

Abbildung 6.3: Dialog der ausführlichen Anforderungsbeschreibung

Zusätzlich werden in der Requirements Tracing View zu jeder Anforderung die Klassennamen der verknüpften Klassen als Kindelemente angezeigt. So kann der Entwickler gleich erkennen in welcher Klasse die jeweilige Anforderung schon implementiert ist. Durch Doppelklick auf den Klassennamen öffnet sich die Klasse im Editorfenster.

Die folgende Abbildung 6.4 zeigt dies.

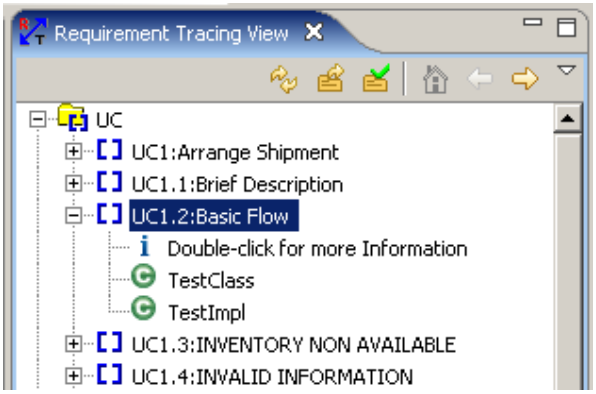

Abbildung 6.4: Requirements Tracing View

Hat sich eine Anforderung in RequisitePro, die auch eine Verknüpfung zu einem Source Codeelement hat, geändert, so wird dies durch ein anderes Icon der jeweiligen Anforderung dargestellt. So kann der Entwickler gleich erkennen, dass sich etwas geändert hat und kann in der Anforderungsbeschreibung nachlesen und mögliche Änderungen umsetzen. Diese Suspect Markierung wird nur beim Laden der Anforderungen ausgelesen. Um die aktuellen Zustände der Anforderungen zu erhalten muss die Ansicht neu geladen werden. In weiteren Entwicklungsstufen soll dies durch eine automatische Aktualisierung der Ansicht durch einen Timer (alle 10 Minuten) oder durch Notifizierungssysteme erfolgen.

6.2.2.4 Export

Das hinzufügen des Packagenamens und Attributs war eine notwendige Erweiterung um das Problem der gleichnamigen Klassen in verschiedenen Packages zu lösen. So sind alle Klassen in einem Projekt eindeutig identifizierbar.

Im Abschnitt 7 werden die Funktionalität und der Einsatz des RT-Plugins genau dokumentiert.

6.3 Diskussion

Die Evaluierung durch die Requirements Engineering Experten der Siemens PSE brachte wichtige Ergebnisse. So wurden mehrere Verbesserungen an dem RT-Plugin umgesetzt. Dies führt zu einer noch besseren Funktionalität und höherer Praxisnähe. Die Erstevaluierung und die Einschätzung der Experten bezüglich der Aufwandsreduktion und der Qualitätsverbesserung der erstellten Verknüpfungen sind vielversprechend für eine weitere Entwicklung dieses Ansatzes. Zusätzlich führt die Reduktion des Aufwands, für die Erstellung der Verknüpfungen, und den zusätzlichen Funktionalitäten, wie zum Beispiel das Anzeigen der ausführlichen Anforderungsinformationen und der schon vorhandenen Verknüpfungen, zu einer höheren Motivation der Entwickler diese Verknüpfungen auch tatsächlich zu erstellen und zu pflegen.

In der folgenden Abbildung 6.5 wird der positive und negative Zyklus des Requirement Tracings veranschaulicht.

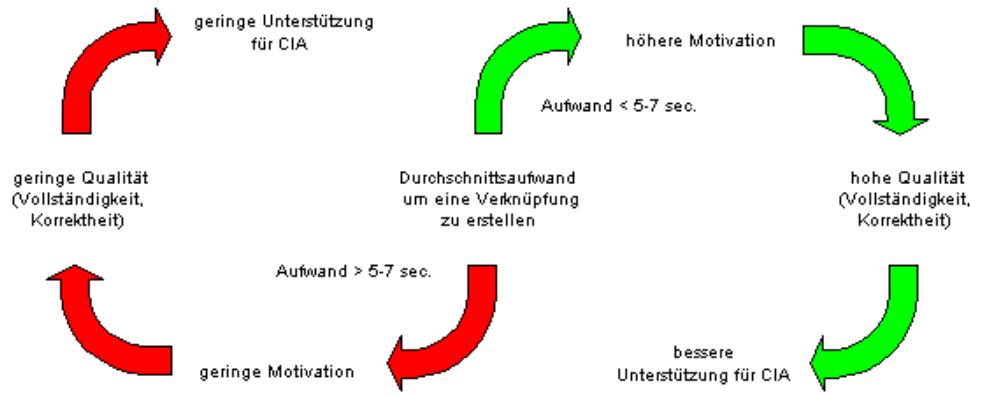

Abbildung 6.5: positiver und negativer Motivationszyklus

Wie in der Abbildung 6.5 dargestellt, sollte der Aufwand um eine Verknüpfung zu erstellen, nicht über 7 Sekunden ansteigen. Dies würde zu einer geringeren Motivation der Entwickler führen diese Verknüpfungen gewissenhaft zu erstellen. Diese geringe Motivation führt zu einer geringeren Qualität der erstellten Verknüpfungen in Hinsicht auf die Vollständigkeit und die Korrektheit. Wenn diese nicht vollständig und korrekt sind, dann bieten diese auch keine große Hilfe und Unterstützung in weiteren Tätigkeiten wie der Change Impact Analyse(CIA).

Durch das RT-Plugin konnte der Aufwand eine Verknüpfung zu erstellen auf einen einfachen Doppelklick reduziert werden, der jedenfalls unter dieser kritischen 7 Sekunden Grenze liegt. Somit kommen wir in den positiven Zyklus. Durch den geringen Aufwand und der zusätzlichen Informationen, die der Entwickler durch das RT-Plugin in seiner Entwicklungsumgebung bekommt, kann die Motivation erheblich gesteigert werden. Dies führt zweifelsohne zu einer höheren Qualität der erstellten Verknüpfungen, da die Fehlerquelle auf ein „Verklicken“ des Entwicklers beschränkt wird. Die höhere Qualität der Verknüpfungen führt zu einer besseren Unterstützung für die nachfolgenden Tätigkeiten (CIA), da sich der Projektleiter auf die Informationen verlassen kann.

Das RT-Plugin und er plugin-basierte Ansatz stellt somit meiner Meinung nach einen sehr vielversprechenden Ansatz im Requirements Tracing dar und sollte weiter verfolgt und verbessert werden

7 Anwendungs- und Schnittstellenbeschreibung

In diesem Kapitel wird beschrieben wie das RT-Plugin in Zusammenarbeit mit RequisitePro verwendet werden kann. Dabei werden auch die neu hinzu gekommenen Funktionalitäten erklärt. Im zweiten Teil des Kapitels wir die Schnittstelle zu den Anforderungsmanagement Tools beschrieben. Durch diese Schnittstelle kann ein jedes Anforderungsmanagement Tool mit dem RT-Plugin zusammenarbeiten.

7.1 Anwendungsbeschreibung an Hand eines Beispiels

In diesem Abschnitt werden der Einsatz und die, nach der Evaluierung erweiterte, Funktionalität des RT-Plugins an Hand eines Beispiels beschrieben. In diesem Beispiel wird das Zusammenspiel des RT-Plugins mit dem Anforderungsmanagement Tool Rational RequisitePro von IBM gezeigt. Dabei werden die in RequisitePro verwalteten Anforderungen importiert und die erstellten Verknüpfungsinformationen wieder in RequisitePro re-importiert. Dort werden diese Verknüpfungen zwischen den Anforderungen und den Codeelementen (den Klassen und Methoden) in einer Traceability Matrix dargestellt. Es werden zwei unterschiedliche Ansichten der Traceability Matrix erstellt. Die Eine zeigt die Verknüpfungen auf Methodenebene und die Zweite zeigt die Verknüpfungen auf der Klassenebene. In den folgenden Teilabschnitten werden die Voraussetzungen in RequisitePro, die Einstellungen im RT-Plugin, die Erstellung der Verknüpfungen und als Ergebnis die re-importierten Verknüpfungsinformationen in RequisitePro genauer beschrieben.

7.1.1 Voraussetzungen in RequisitePro

Voraussetzung um das Plugin mit RequisitePro verwenden zu können ist, dass in RequisitePro ein Projekt erstellt wurde, in dem die Anforderungen erstellt wurden und dort auch weiterhin verwaltet werden. Es muss also in RequisitePro ein Projekt und zumindest ein Anforderungstyp erstellt werden. Dies ist notwendig um Anforde-

rungen in RequisitePro erstellen zu können. Für dieses Beispiel wurde ein leeres Projekt erstellt. Es wurde ein Package namens „Requirements“, in dem dann alle Anforderungen gespeichert werden und ein Anforderungstyp namens „Use Case“ (UC) erstellt. Danach konnten die Anforderungen angelegt werden.

Abbildung 7.1 zeigt RequisitePro mit dem Beispielprojekt.

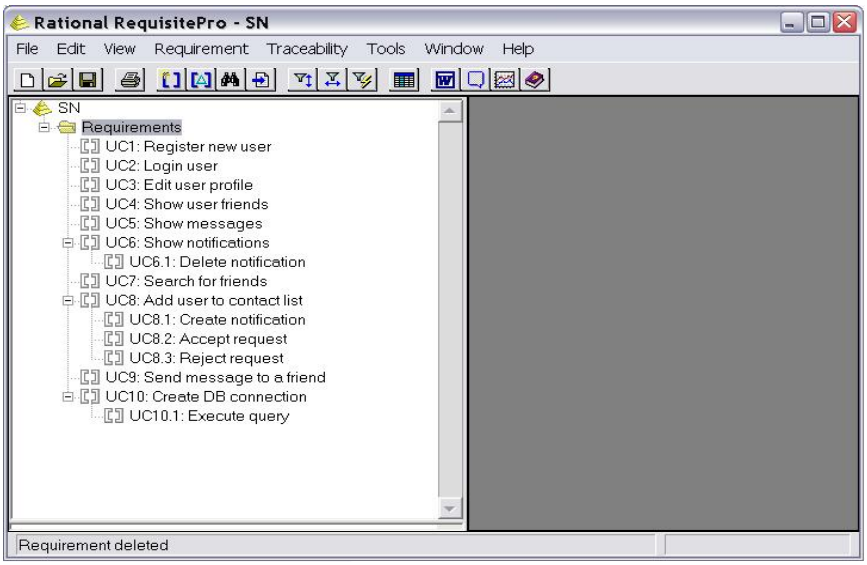

Abbildung 7.1: RequisitePro mit Beispielprojekt

7.1.2 Einstellungen in RT-Plugin

Um nun die Anforderungen aus RequisitePro importieren und verknüpfen zu können müssen in Eclipse noch die projektspezifischen Einstellungen vorgenommen werden. Dazu sind über das Menü die Projekteinstellungen öffnen, dann auf der linken Seite in der Liste das Element „Requirement Project“ auswählen. Danach sollte das Dialogfeld der folgenden Abbildung 7.2 entsprechen.

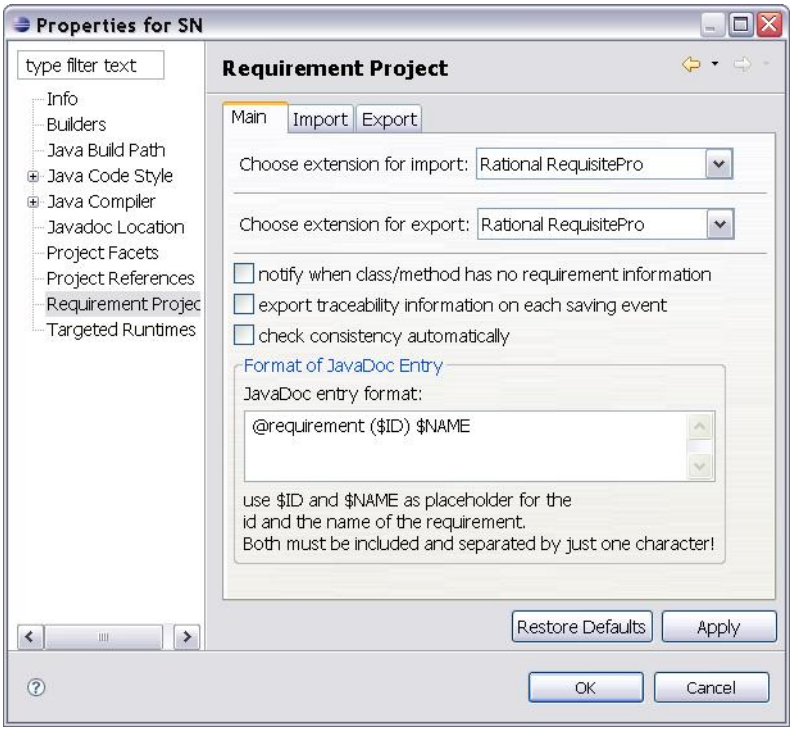

Abbildung 7.2: Dialog der Projekteinstellungen

Nun können die Erweiterungen, die für den Datenimport und den Datenexport verwendet werden sollen, ausgewählt werden. Da in diesem Beispiel die Zusammenarbeit mit RequisitePro beschrieben wird, wähle wir hier die Erweiterung „Rational RequisitePro“.

Nun müssen noch die erweiterungsspezifischen Einstellungen vorgenommen werden. Dazu markieren wir zuerst den Tab „Import“. Die Ansicht des Dialoges ändert sich und sollte nun der folgenden Abbildung 7.3 entsprechen.

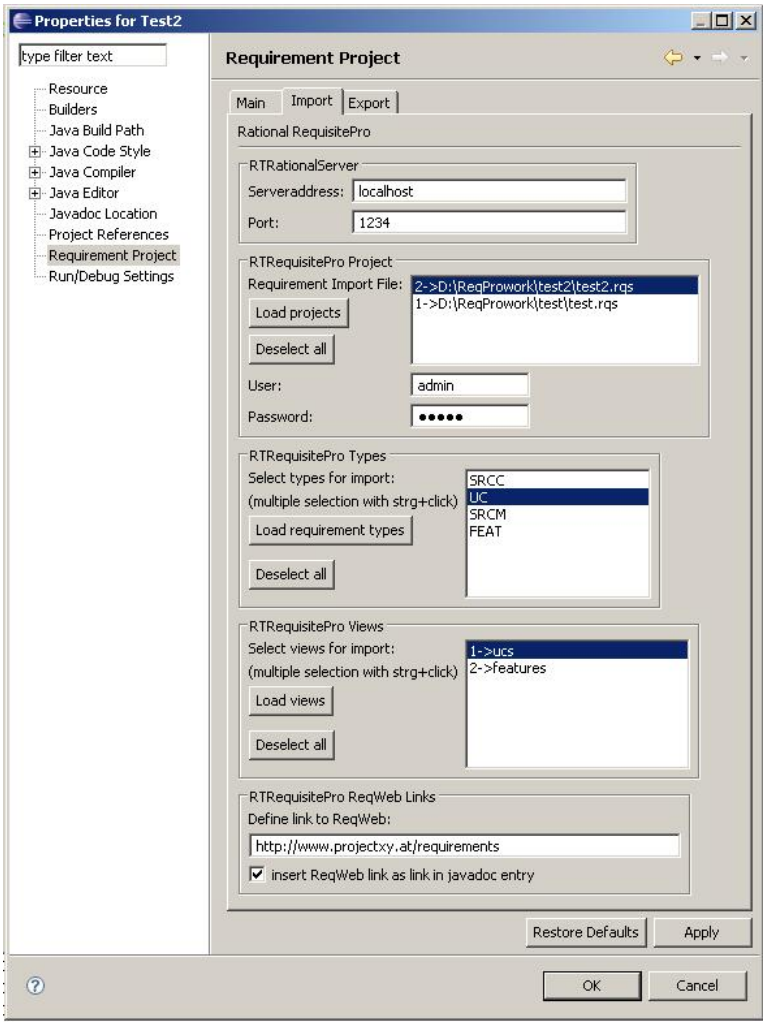

Abbildung 7.3: Dialog der Projekteinstellungen - Import Tab

Zu Beginn muss die Adresse des Servers und der Port auf dem die Serveranwendung läuft, definiert werden. Im Anschluss kann über den Button „Load projects“ die Liste der zur Verfügung stehenden Projekte geladen und eines davon ausgewählt werden. Danach kann noch der Benutzername und das Passwort (sofern dies in RequisitePro eingestellt wurde) eingegeben werden. Jetzt können die vorhandenen Anforderungstypen durch betätigen des Buttons „Load requirement types“ geladen werden. Nun können die gewünschten Anforderungstypen, deren Anforderungen importiert werden sollen ausgewählt werden. Durch betätigen des Buttons „Load views“ können auch die vorhandenen Views importiert werden. Der Benutzer kann also auch eine View für den Import auswählen, dabei werden dann nur jede Anforderungen importiert, die auch in der View angezeigt werden. So kann der Benutzer die Anforderungen in RequisitePro noch besser filtern. Eine Kombination beider Auswahlmöglichkeiten ist ebenfalls möglich. Zuletzt kann noch der Link zu der Webseite, auf der die

Anforderungsbeschreibungen als HTML-Seiten gespeichert sind, eingetragen werden. Wird das Optionsfeld „insert ReqWeb link as link in JavaDoc entry“ selektiert, so wird dieser Link auch für den JavaDoc-Eintrag berücksichtigt. Dabei wird automatisch der Link in folgendem Format eingetragen:

@see *JavaDoc Entry Format*

Zur Erläuterung: Die kursiven Wörter stellen Schlüsselwörter dar, wobei *Link* für den ReqWeb Link, *DocKey* für den Dokumentenschlüssel aus der Beschreibung, *Bookmark* für den Bookmark aus der Beschreibung und *JavaDoc Entry* Format für das auf dem Main Tab definierte Eintragsformat steht.

Nun müssen noch die erweiterungsspezifischen Einstellungen im Tab „Export“ vorgenommen werden. Hier wird ebenso dieselbe Projektdatei von RequisitePro wie auf dem „Import“ Tab ausgewählt.

Die vorgenommenen Einstellungen mit „OK“ bestätigen und speichern. Wenn das „Requirement Tracing View“ - Fenster noch nicht geöffnet ist, dann kann dieses über das Menü „Window / Show View / Other“ im Package „Requirement Tracing Plugin“ geöffnet werden. Danach sollte Eclipse etwa der folgenden Abbildung 7.4 entsprechen.

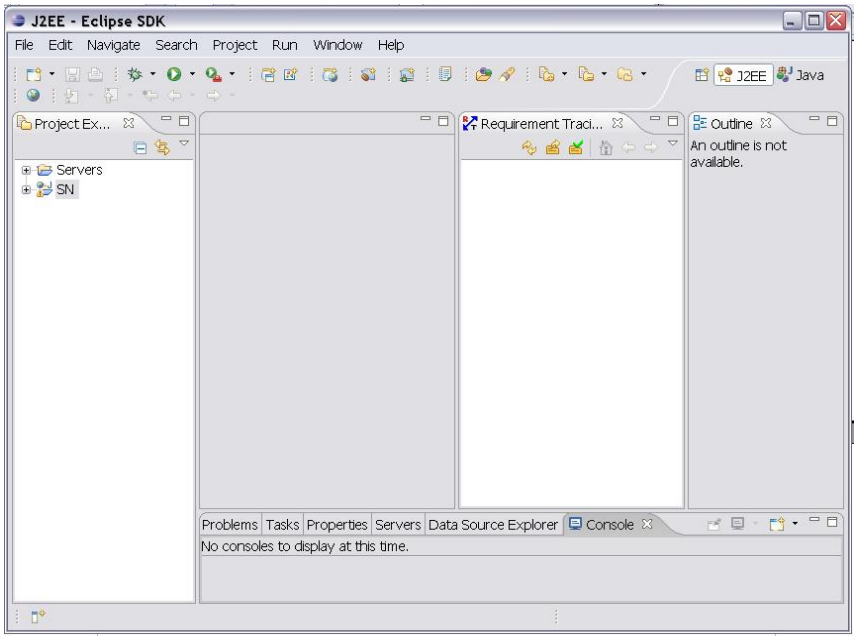

Abbildung 7.4: Eclipse IDE mit geöffnetem Requirement Tracing View - Fenster

7.1.3 Erstellen der Verknüpfungen

Um Verknüpfungen zwischen den Codeelementen (den Klassen und Methoden) und den importierten Anforderungen erstellen zu können muss eine Datei mit Java Code geöffnet werden. Danach werden die Anforderungen automatisch von RequisitePro importiert und im “Requirement Tracing View” - Fenster angezeigt. Die Anforderungen werden ihren Typen entsprechend zusammengefasst dargestellt. Die Darstellung erfolgt in einer einfachen Baumstruktur. Das Hauptelement ist immer der Typ der Anforderungen, als dessen Kindelemente werden die Bezeichnungen der Anforderungen dargestellt. Eine kurze Beschreibung dieser Anforderungen wird als Kindelement der Anforderungen dargestellt. Die Ansicht der Eclipse IDE sollte nun der folgenden Abbildung 7.5 ähneln.

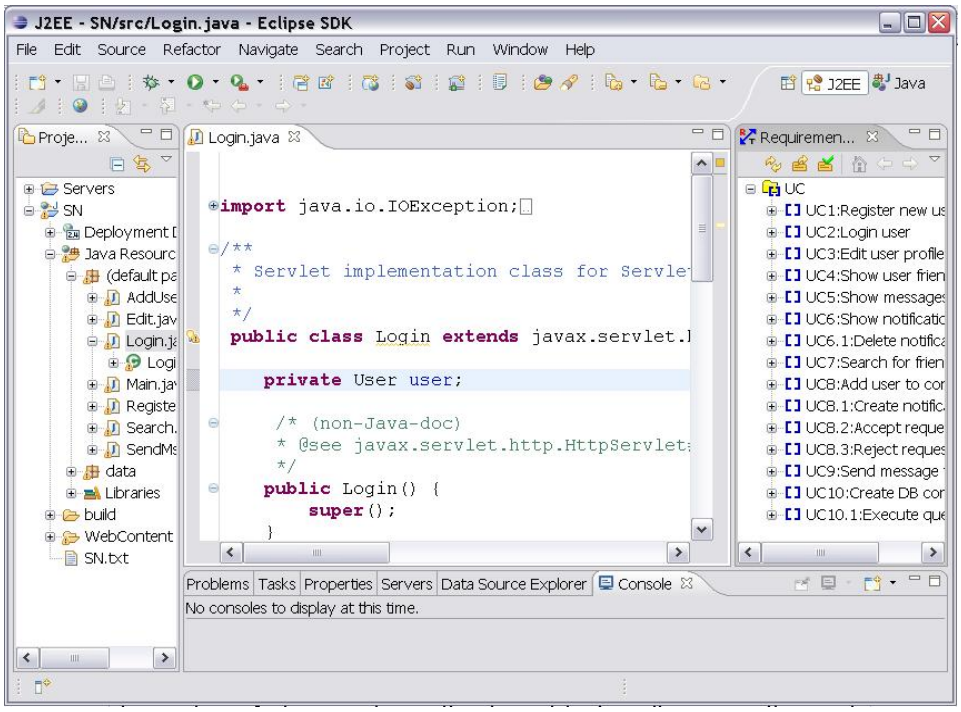

Abbildung 7.5: Eclipse IDE mit geöffneter Klasse und RTV-Fenster

Um nun eine neue Verknüpfung zwischen einer Methode und einer Anforderung herzustellen, muss der Cursor in den Code-Bereich der jeweiligen Methode positioniert werden. Danach kann mit einem Doppelklick auf die zu verknüpfende Anforderung in dem „Requirement Tracing View“ - Fenster die Verknüpfung hergestellt werden. Es wird nun automatisch in den JavaDoc-Bereich der jeweiligen Methode ein Eintrag mit der ID und der Bezeichnung der Anforderung hinzugefügt. Gleichzeitig wird zu der Anforderung die Klasse als Kindelement hinzugefügt.

Die nun erstellte Verknüpfung zeigt die folgende Abbildung 7.6.

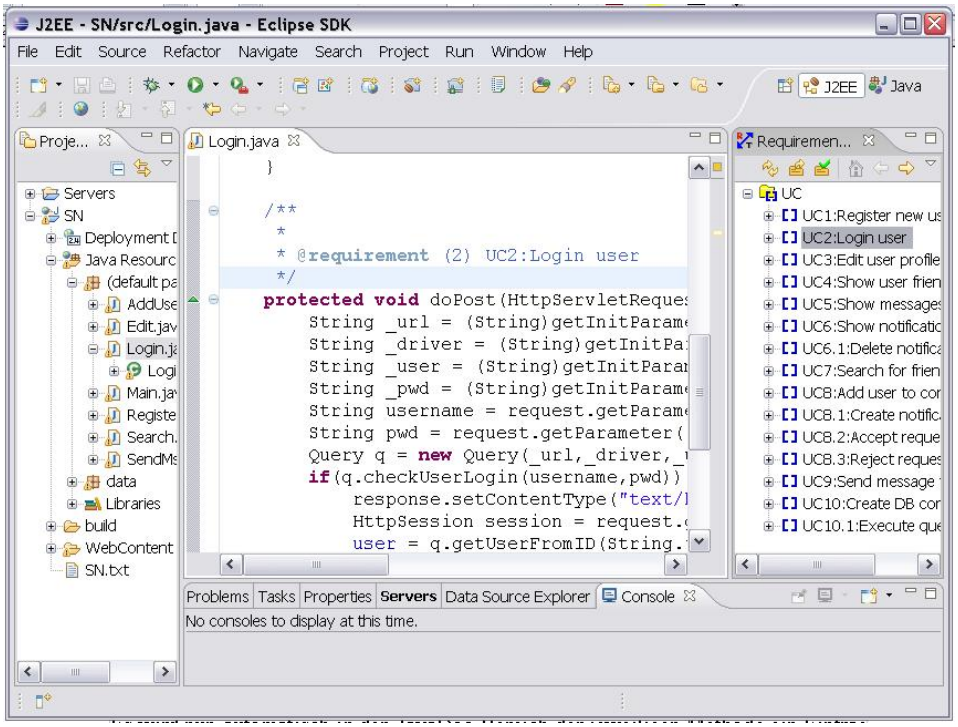

Abbildung 7.6: Eclipse Fenster mit erstellter Verknüpfung

Nun kann die Verknüpfungsinformation durch betätigen des Buttons „Export“ ( ) in der Toolbar des „Requirement Tracing View“ - Fensters exportiert und in RequisitePro re-importiert werden. Der Export und der Re-import geschehen vollkommen automatisch. Der Export der Verknüpfungsinformationen kann auch automatisch bei jedem „Speichern“ - Event durchgeführt werden. Dazu müssen die entsprechenden Einstellungen in den Projekteinstellungen vorgenommen werden. Dies wurde in dem Kapitel 3.3.1 schon beschrieben.

7.1.4 Konsistenzprüfung

Bei der Konsistenzprüfung werden die existierenden Verknüpfungen zwischen den Klassen / Methoden und den Anforderungen der jeweils aktuell geöffneten Klasse mit den bestehenden Verknüpfungsinformationen, die über die definierte Exporterweiterung (in dem Beispiel ist das RequisitePro) importiert werden, verglichen. Wurde eine Anforderung in dem Anforderungsmanagement Tool umbenannt, so wird dies bei der Konsistenzprüfung erkannt und angezeigt. Weiters wird überprüft ob eine Anforderung gelöscht wurde oder eine Verknüpfung gelöscht oder erstellt wurde. Dies wird ebenfalls angezeigt.

Die folgende Abbildung 7.7 zeigt das Dialogfenster wenn Konsistenzprobleme gefunden wurden.

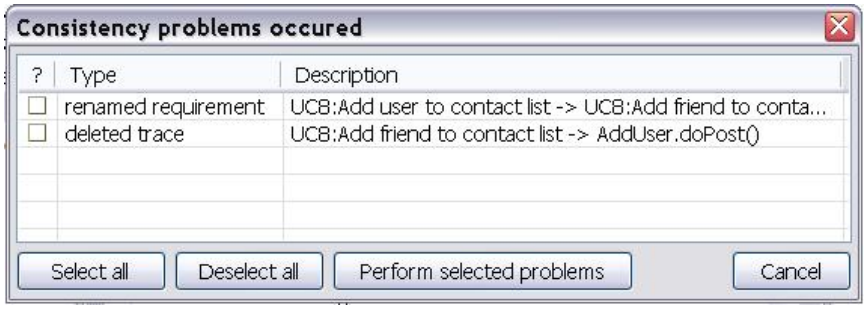

Abbildung 7.7: Dialogfenster der Konsistenzprüfung

Es wurden zwei Probleme gefunden und in einer Tabelle angezeigt. Die Spalte „Type“ zeigt an welcher Art das Problem ist.

Es werden vier Arten von Problemen unterschieden.

- *renamed requirement:* Diese Art gibt an, dass die Anforderung im Anforderungsmanagement Tool umbenannt wurde.
- *deleted requirement:* Diese Art gibt an, dass die Anforderung im Anforderungsmanagement Tool gelöscht wurde.
- *deleted trace:* Diese Art gibt an, dass die Verknüpfung im Anforderungsmanagement Tool gelöscht wurde oder eine neue Verknüpfung im Code erstellt und nicht exportiert wurde.
- *new trace:* Diese Art gibt an, dass eine neue Verknüpfung im Anforderungsmanagement Tool erstellt wurde oder eine Verknüpfung im Code gelöscht und nicht exportiert wurde.

Die Spalte „Description“ gibt die Beschreibung des Problems wieder. Hier wird angeführt, welche Anforderung umbenannt oder welche Verknüpfung neu/gelöscht wurde. Der Benutzer kann dann auswählen welche der Änderungen übernommen werden sollen. Durch Selektion der gewünschten Änderungen und dem Betätigen des Buttons „Perform selected problems“ werden die Änderungen in den Code übernommen.

Diese Konsistenzprüfung ist allerdings nur dann notwendig, wenn die Möglichkeit besteht, dass die Anforderungen im Anforderungsmanagement Tool geändert wurden. Die Konsistenzprüfung macht auch nur dann Sinn, wenn diese gleich nach dem

öffnen einer Klasse durchgeführt wurde. Deshalb ist es möglich die Option der automatischen Konsistenzprüfung bei jedem Öffnen einer Klasse in den projektspezifischen Einstellungen zu aktivieren.

7.1.5 Anzeigen der ausführlichen Beschreibung einer Anforderung

Um eine ausführliche Beschreibung einer Anforderung zu bekommen, kann der Benutzer diese durch einen Doppelklick auf die Kurzbeschreibung der jeweiligen Anforderung im Baum öffnen. Die Abbildung 7.8 zeigt das „Requirements Tracing View“ - Fenster mit den Anforderungen.

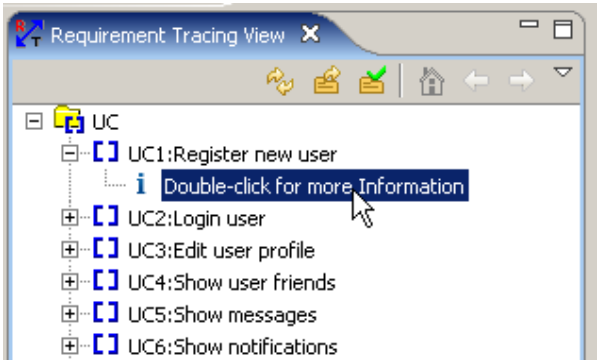

Abbildung 7.8: Requirements Tracing View

Durch einen Doppelklick auf das selektierte Element öffnet sich ein Dialog, in dem die ausführliche Beschreibung und die Attribute und die Historie der Anforderung dargestellt wird. Abbildung 7.9 zeigt diesen Dialog.

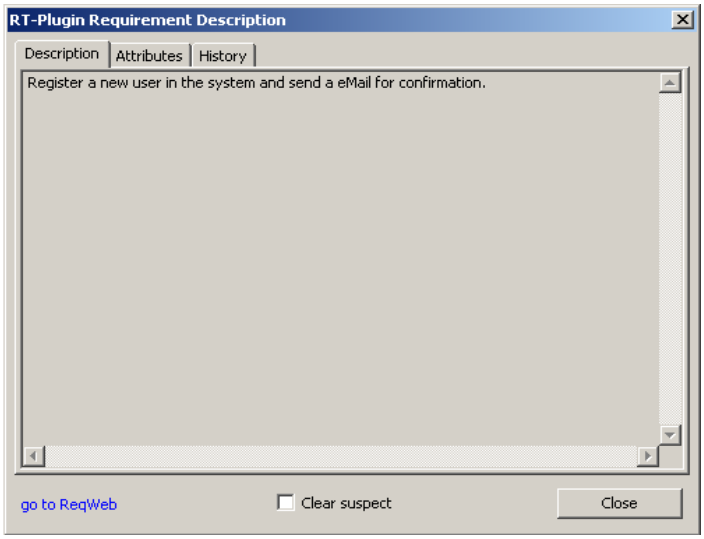

Abbildung 7.9: Description Dialog, Description Tab

In dem Description – Tab (Abbildung 7.9) wird die komplette Beschreibung in einem mehrzeiligen Textfeld dargestellt. In der linken unteren Ecke, in blau geschrieben, ist ein Link, der den definierten ReqWeb Link in einem Browser öffnet. Mit dem Optionsfeld in der Mitte kann das Attribut „Suspect“ gelöscht werden.

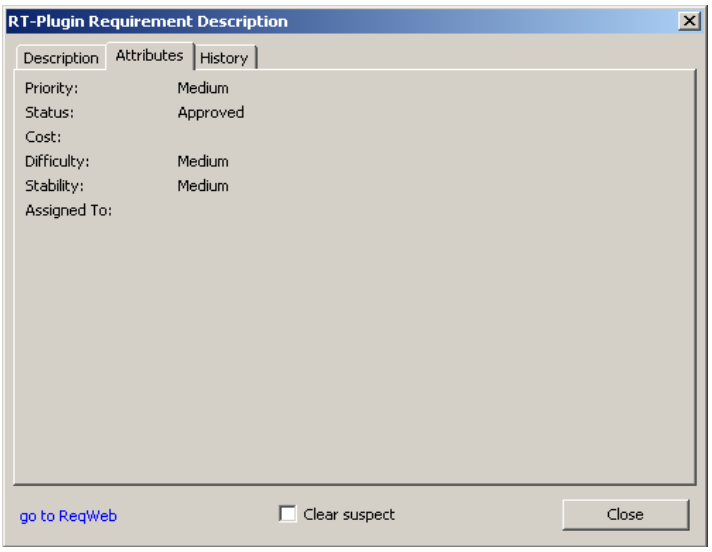

Abbildung 7.10: Description Dialog, Attribute Tab

In dem Attributes – Tab (Abbildung 7.10) werden die Attribute der Anforderung aufgelistet.

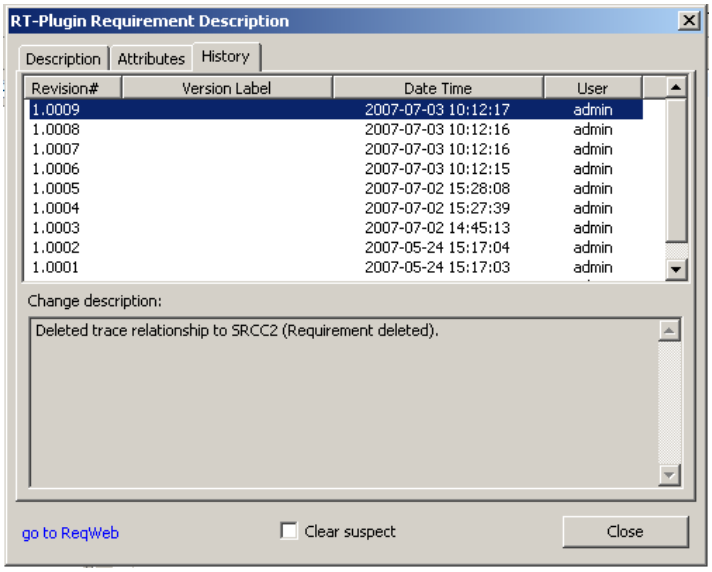

Abbildung 7.11: Description Dialog, History Tab

In dem Hitory – Tab (Abbildung 7.11) werden die Änderungen, die vorgenommen wurden angezeigt. In der Tabelle werden die Änderungen nach deren Revision Nummern sortiert angezeigt. Hier werden zusätzlich das Datum der Änderung und der Benutzername angezeigt. Wird eine Zeile in der Tabelle selektiert, so wird in den unteren Textfeldern die Beschreibung der Änderung angezeigt.

7.1.6 Ergebnis: Traceability Matrix in RequisitePro

Beim Export und dem Re-import der Verknüpfungsinformationen in RequisitePro wird im Projekt ein neues Package “Source Code” erstellt. In diesem Package werden alle Codeelemente in Form von Anforderungen erstellt. Zusätzlich werden die zwei Ansichten, die die Traceability Matrix für die Verknüpfungen zwischen den Klassen und den Anforderungen und die Traceability Matrix für die Verknüpfungen zwischen den Methoden und den Anforderungen zeigen, erstellt.

Die Abbildung 7.12 zeigt die Traceability Matrix der Verknüpfungen zwischen den Klassen und den Anforderungen.

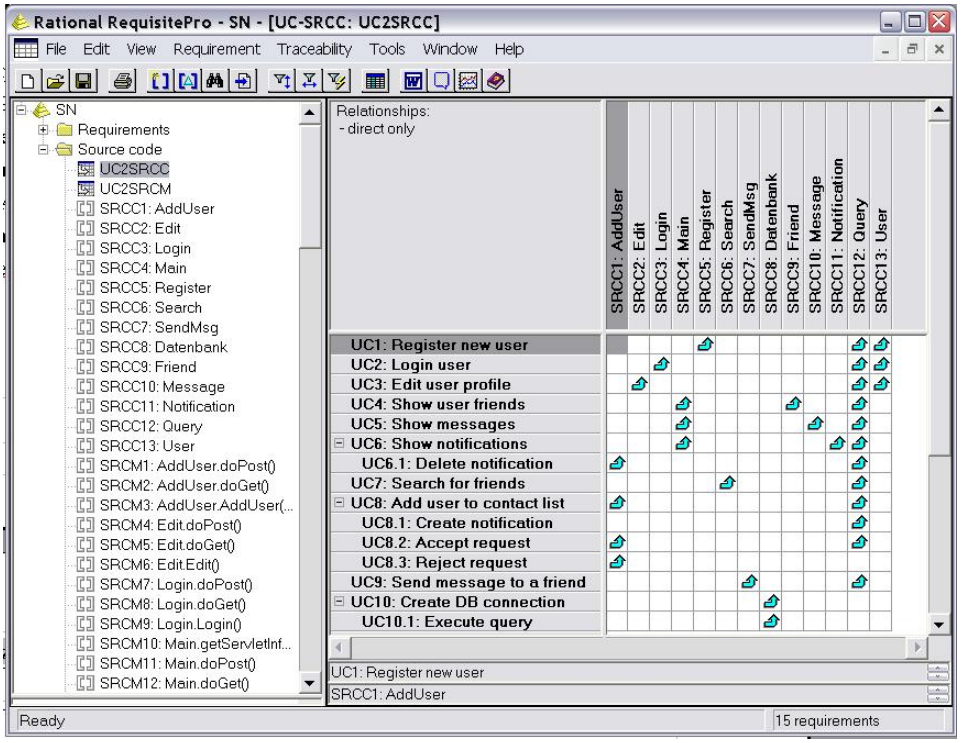

Abbildung 7.12: Traceability Matrix in RequisitePro (Klassen – Anforderungen)

Die Abbildung 7.12 zeigt auf der linken Seite das neu erstellte Package „Source Code“, das die Klassen und Methoden in Form von Anforderungen zeigt. Dabei wird den Klassen der Anforderungstyp „SRCC“ und den Methoden der Anforderungstyp „SRCM“ zugewiesen. Die ersten zwei Einträge in dem Package stellen die Ansichten dar. Auf der rechten Seite der Abbildung wird die Tracability Matrix, die die Verknüpfungsinformationen zwischen den Klassen und den Anforderungen zeigt, dargestellt. Die Anforderungen werden hier vertikal in den einzelnen Zeilen dargestellt. Die Anforderungen, die den Klassen entsprechen, werden horizontal in Spalten dargestellt. Besteht zwischen einer Anforderung und einer Klasse eine Verknüpfung, so wird in der entsprechenden Zelle der Matrix ein Pfeil dargestellt.

Abbildung 7.13 zeigt die Traceability Matrix, die die Verknüpfungsinformationen zwischen den Anforderungen und den Methoden darstellt.

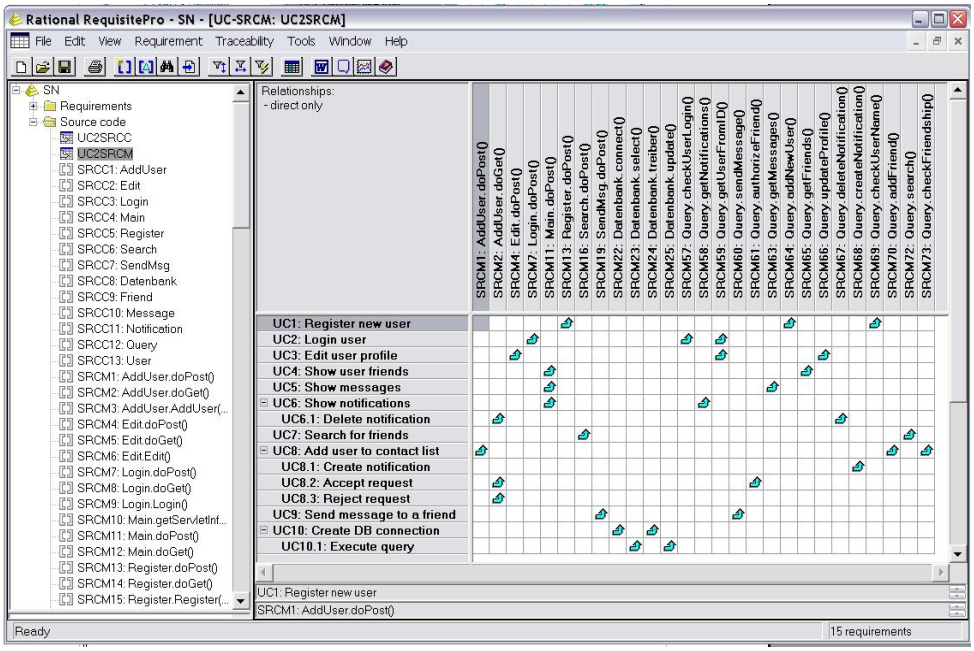

Abbildung 7.13: Traceability Matrix in RequisitePro (Methoden - Anforderungen)

In dieser Ansicht werden ebenfalls die Anforderungen vertikal in den Zeilen dargestellt. Die Anforderungen, die den Methoden entsprechen, werden horizontal in den Spalten dargestellt. Die Verknüpfung zwischen einer Anforderung und einer Methode wird wieder mit einem Pfeil dargestellt.

7.2 Schnittstellenbeschreibung

Das RT-Plugin hat eine Schnittstelle über die der Benutzer mehrere Erweiterungen für den Datenimport und den Datenexport anknüpfen kann. So kann ganz einfach für jedes Anforderungsmanagement Tool eine Erweiterung zu dieser Schnittstelle implementiert werden. Das RT-Plugin kann somit mit allen beliebigen Anforderungsmanagement Tools kommunizieren und kombiniert werden.

Im Folgenden wird diese Schnittstelle beschrieben. Im Abschnitt 7.2.1 wird die Extension Point Definition beschrieben. Im Abschnitt 7.2.2 werden die Interfaces für die Schnittstellenklassen näher beschrieben. Der Abschnitt 7.2.3 beschreibt die Interfaces für die Datenformate des Datenaustausches beschrieben.

Ein Beispiel einer einfachen Erweiterung findet sich im Appendix A.

7.2.1 Extension Point

Wie schon in einem früheren Abschnitt erwähnt werden in Eclipse so genannte „Extension Points“ definiert, zu denen Erweiterungen implementiert werden können. Das RT-Plugin hat auch so einen Extension Point definiert, der in diesem Abschnitt beschreiben wird.

Um eine Erweiterung zu implementieren ist es notwendig in Eclipse ein neues Plugin-Projekt zu erstellen. In diesem Projekt wird von Eclipse automatisch eine „plugin.xml“ Datei erstellt, in der die Einstellungen des Plugins definiert werden. Unter Anderem wird hier auch der Extension Point, an den das Plugin anknüpfen soll, definiert.

Diese Extension Point Definition in der „plugin.xml“ Datei zeigt folgendes Listing.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         id="RTSampleExtension"
         name="RT Sample Extension"
         point="RT.rtextension">
      <rtextension name="Sample Extension">
         <import
               class="rte.RTImport"
               name="RTImport"/>
         <export
               class="rte.RTExport"
               name="RTExport"/>
         <importpropertypage
               class="rte.RTImportPropertyPage"
               name="RTImportPropertyPage"/>
         <exportpropertypage
               class="rte.RTExportPropertyPage"
               name="RTExportPropertyPage"/>
      </rtextension>
   </extension>
</plugin>
```

Listing 1: plugin.xml der Erweiterung

Mit dem Tag `<extension>` wird der Extension Point, den man verwenden will definiert. Hierbei steht das Attribut id für die ID des zu implementierenden Plugins. Das Attribut name definiert den Namen des Plugins. Mit dem Attribut point wird der Name des Extension Points angegeben, der für das RT-Plugin „RT.rtextension“ lautet.

In dem Tag `<rtextension>` werden die schnittstellenspezifischen Einstellungen vorgenommen. Das Attribut name steht für den Namen der Erweiterung, der dann

auch in der Auswahlliste der Import- und Export- Erweiterung in den Projekteinstellungen auf scheint.

Die Subtags definieren dann die Klassennamen, die die von der Schnittstelle definierten Interfaces definieren und auf die das RT-Plugin zugreift. Hierbei wird im `import` Tag die Klasse definiert, die die Importdaten zur Verfügung stellt. Der `export` Tag verarbeitet die Daten über die Verknüpfungsinformationen des RT-Plugins. Der Tag `importpropertypage` erstellt den Inhalt für das „Import“ Register in den Projekteinstellungen, wo dann die erweiterungsspezifischen Einstellungen für den Import vorgenommen werden. Der `exportpropertypage` Tag definiert den Inhalt des „Export“ Registers in den Projekteinstellungen wo ebenfalls die erweiterungsspezifischen Einstellungen für den Export vorgenommen werden.

Damit wären die grundlegenden Einstellungen abgeschlossen. Nun müssen die in der „plugin.xml“ Datei definierten Klassen noch der Schnittstelle entsprechend implementiert werden. Dazu wurden entsprechende Interfaces definiert, die im folgenden Abschnitt beschrieben werden.

7.2.2 Interfaces für die Schnittstellenklassen

In diesem Abschnitt werden die Interfaces, die die Klassen des Erweiterungs - Plugins implementieren müssen, näher beschrieben. Das Beispiel in Appendix zeigt dann wie diese Interfaces eingesetzt werden sollen.

7.2.2.1 IImport

Dieses Interface definiert die Methoden, die jene Klasse die den Datenimport umsetzt implementieren muss. In dem Beispiel der „plugin.xml“ Datei aus Abschnitt 7.2.1 betrifft es die Klasse „rte.RTImport“.

Das folgende Listing zeigt das Interface.

```
public interface IImport {
   public List<IRequirement> getRequirements(
      IProject project) throws Exception;
   public List<IType> getTypes(IProject project)
      throws Exception;
   public Description getDescription(IProject project,
      String ReqID) throws Exception;
   public void clearSuspect(IProject project, String id)
      throws Exception;
}
```

Listing 2: Interface IImport

Die Methode `getRequirements()` stellt eine Liste der Anforderungen zur Verfügung. Mit dem Parameter project wird das aktuelle Projekt übergeben, über welches die Projekteinstellungen ausgelesen werden können.

Die Methode `getTypes()` stellt die Liste der Anforderungstypen, die angezeigt werden sollen zur Verfügung.

Die Methode `getDescription()` liefert ein Description – Objekt, in dem die vollständige Beschreibung der Anforderung gespeichert ist. In diesem Objekt werden die Attribute und die History gespeichert.

Die Methode `clearSuspect()` entfernt das Attribut, das anzeigt, dass sich eine verknüpften Anforderung geändert hat.

7.2.2.2 IExport

Dieses Interface definiert die Methoden, die jene Klasse die den Datenexport umsetzt implementieren muss. In dem Beispiel der „plugin.xml“ Datei aus Abschnitt 7.2.1 betrifft es die Klasse „rte.RTExport“.

Das folgende Listing zeigt das Interface.

```
public interface IExport {
   public void exportInformation(
      List<ITraceabilityInformation> tilist,
      Map<String,ICodeElement> clist,
      Map<String,IRequirement> rlist,
      Map<String,IType> tlist, IProject project)
      throws Exception;
   public TransferTraceInformation getTraceabilityInformation(
      IProject project, String classname) throws Exception;
   public Map<String,List> getTraceInformationForRequirements(
      IProject project, Object[] reqIDs) throws Exception;
}
```

Listing 3: Interface IExport

Dieses Interface hat nur die Methode `exportInformation()` der alle Informationen über die Verknüpfungen und den beteiligten Objekten übergeben werden.

Der Parameter `tilist` stellt eine Liste aller Verknüpfungsinformationen dar.

Der Parameter `clist` stellt eine Liste der Code Elemente (Methoden und Klassen) dar.

Der Parameter `rlist` stellt eine Liste aller importierten Anforderungen dar.

Der Parameter `tlist` stellt eine Liste aller Anforderungstypen dar.

Der Parameter `project` stellt das aktuelle Projekt dar, über den man die Projekteinstellungen auslesen kann.

Die Methode `getTraceabilityInformation()` liefert die schon vorhandenen Verknüpfungsinformationen für die mit dem Parameter classname übergebenen Klasse.

Die Methode `getTraceInformationForRequirements()` liefert die Verknüpfungsinformationen zu allen Anforderung. In der Map werden zu jeder Anforderungs ID eine Liste der Klassennamen die mit der Anforderung verknüpft sind gespeichert.

7.2.2.3 IImportPropertyPage

Dieses Interface definiert die Methoden, die jene Klasse implementieren muss, die den Inhalt des „Import“-Registers der Projekteinstellungen erstellt. In dem Beispiel der „plugin.xml“ Datei aus Abschnitt 7.2.1 betrifft es die Klasse „rte.RTImportPropertyPage“.

Das folgende Listing zeigt das Interface.

```
public interface IImportPropertyPage {
   public void addPropertyPage(IResource resource,
      Composite parent);
   public void performDefault(IResource resource);
   public boolean performOk(IResource resource);
}
```

Listing 4: Interface IImportPropertyPage

Die Methode `addPropertyPage()` fügt die Eingabefelder, mit denen die erweiterungsspezifischen Einstellungen im „Import“-Register der Projekteinstellungen vorgenommen werden, hinzu. Über den Parameter resource kann auf die Projektdatei zugegriffen werden und so auf die schon existierenden Einstellungen zugreifen. Der Parameter composite stellt das übergeordnete Composite Object dar, dem die einzelnen Eingabefelder (Widgets) hinzugefügt werden können.

In der Methode `performDefault()` werden die Eingabefelder auf die Anfangseinstellung zurückgesetzt.

In der Methode `performOk()` werden die in die Eingabefelder eingegebenen Werte in der Projektdatei gespeichert. Dies geschieht durch Verwendung des im Parameter resource übergebenen IResource Objekts.

7.2.2.4 IExportPropertyPage

Dieses Interface definiert die Methoden, die jene Klasse implementieren muss, die den Inhalt des „Export“-Registers der Projekteinstellungen erstellt. In dem Beispiel der „plugin.xml“ Datei aus Abschnitt 7.2.1 betrifft es die Klasse „rte.RTExportPropertyPage“.

Das folgende Listing zeigt das Interface.

```
public interface IExportPropertyPage {
   public void addPropertyPage(IResource resource,
      Composite parent);
   public void performDefault(IResource resource);
   public boolean performOk(IResource resource);
}
```

Listing 5: Interface IExportPropertyPage

Die Methode `addPropertyPage()` fügt die Eingabefelder, mit denen die erweiterungsspezifischen Einstellungen im „Export“-Register der Projekteinstellungen vorgenommen werden, hinzu. Über den Parameter resource kann auf die Projektdatei zugegriffen werden und so auf die schon existierenden Einstellungen zugreifen. Der Parameter composite stellt das übergeordnete Composite Objekt dar, dem die einzelnen Eingabefelder (Widgets) hinzugefügt werden können.

In der Methode `performDefault()` werden die Eingabefelder auf die Anfangseinstellung zurückgesetzt.

In der Methode `performOk()` werden die in die Eingabefelder eingegebenen Werte in der Projektdatei gespeichert. Dies geschieht durch Verwendung des im Parameter resource übergebenen IResource Objekts.

7.2.3 Interfaces für den Datenaustausch

In diesem Abschnitt werden die Interfaces, die das Datenformat für den Datenaustausch definieren, beschrieben.

7.2.3.1 IRequirement

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die eine Anforderung abbildet.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface IRequirement extends Serializable {
   public String getTypeId();
   public String getName();
   public String getPrefix();
   public String getId();
   public String getDescription();
   public boolean isSuspect();
   public boolean hasDocument();
   public String getDocKey();
   public String getBookmark();
   public String getReqWebLink();
}
```

Listing 6: Interface IRequirement

Die Methode `getTypeId()` gibt die ID des Anforderungstyps als String zurück.

Die Methode `getName()` gibt den Namen/Bezeichnung der Anforderung zurück.

Die Methode `getPrefix()` gibt den Präfix der Anforderung zurück. Der Präfix ist bei der RequisitePro Erweiterung das Kürzel des Anforderungstyps kombiniert mit einer fortlaufenden Nummer.

Die Methode `getId()` gibt die eindeutige ID der Anforderung zurück. Bei der RequisitePro Erweiterung ist dies die ID des Datenbankeintrages.

Die Methode `getDescription()` gibt die Beschreibung der Anforderung zurück.

Die Methode `isSuspect()` gibt an ob sich bei der Anforderung etwas geändert hat.

Die Methode `hasDocument()` gibt an ob die Anforderung einen Verweis zu einem Dokument hat.

Die Methode `getDocKey()` liefert den Schlüssel des Dokumentes zurück.

Die Methode `getBookMark()` liefert die genaue Bookmark und somit die genaue Position der Anforderung im Dokument zurück.

Die Mehtode `getReqWebLink()` liefert die Linkadresse zu der Webseite, auf der die Anforderungsbeschreibungen gespeichert sind.

7.2.3.2 IType

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die den Anforderungstyp beschreibt.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface IType extends Serializable {
   public String getName();
   public String getPrefix();
   public String getId();
}
```

Listing 7: Interface IType

Die Methode `getName()` gibt den Namen/Bezeichnung des Anforderungstyps zurück.

Die Methode `getPrefix()` gibt den Präfix (das Kürzel) des Anforderungstyps zurück.

Die Methode `getId()` gibt die eindeutige ID des Anforderungstyps zurück.

7.2.3.3 ICodeElement

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die ein Code Element (Methode oder Klasse) beschreibt.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface ICodeElement extends Serializable {
   public String getName();
   public boolean isClass();
   public boolean isMethod();
   public String getId();
   public Object getIJavaElement();
   public String getPackageName();
}
```

Listing 8: Interface ICodeElement

Die Methode `getName()` gibt den Namen der Methode oder Klasse zurück. Hierbei setzt sich der Name wie folgt zusammen. Der Klassenname entspricht dem Namen. Bei den Methoden einer Klasse setzt sich der Name aus dem Klassennamen und dem Methodennamen zusammen. Dies entspricht dann diesem Format: „Klasse.methode()“. Bei einer Innerclass wird der Klassenname der äußeren Klasse vor den Namen der Innerclass und deren Methoden gesetzt. Dies geschieht in folgendem Format: „Klasse#Innerclass.methode()“.

Die Methode `isClass()` gibt true zurück wenn es sich um eine Klasse handelt.

Die Methode `isMethod()` gibt true zurück wenn es sich um eine Methode handelt.

Die Methode `getId()` gibt die intern vergebene ID der Code Elemente zurück. Diese wird für die Zuweisung bei den Verknüpfungen benötigt, da dort nur die verknüpften ID’s gespeichert werden.

Die Methode `getIJavaElement()` gibt das zugehörige IJavaElement der Methode oder Klasse zurück. Das IJavaElement ist im Package org.eclipse.jdt.core.IJavaElement definiert. Dies wird benötigt um schon vorhandene Verknüpfungen zu ändern oder neue zu erstellen. Für den Datenimport oder -export wird dieses Objekt allerdings nicht verwendet.

Die Methode `getPackageName()` liefert den Package Namen der Klasse.

7.2.3.4 ITraceabilityInformation

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die eine Verknüpfungsinformation beschreibt.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface ITraceabilityInformation extends Serializable
{
   public String getCodeElementId();
   public String getRequirementId();
}
```

Listing 9: Interface ITraceabilityInformation

Die Methode `getCodeElementId()` gibt die ID des Code Elements (Methode oder Klasse) zurück.

Die Methode `getRequirementId()` gibt die ID der Anforderung zurück.

7.2.3.5 ITransferTraceInformation

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die die vorhandenen Verknüpfungsinformationen speichert.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface ITransferTraceInformation extends Serializable
{
   public List<TraceabilityInformation>
      getTraceabilityInformation();
   public List<TransferCodeElement> getCodeElements();
}
```

Listing 10: Interface ITransferTraceInformation

Die Methode `getTraceabilityInformation()` gibt eine Liste der im Anforderungsmanagement Tool vorhandenen Verknüpfungsinformationen zurück.

Die Methode `getCodeElements()` gibt eine Liste der Methoden und Klassen zurück, die eine Verknüpfungsinformation haben.

7.2.3.6 ITransferCodeElement

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die die vorhandenen Codeelemente speichert.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface TransferCodeElement extends Serializable {
   public String getId();
   public String getName();
   public boolean isClass();
   public boolean isMethod();
   public String getPackageName();
}
```

Listing 11: Interface ITransferCodeElement

Die Methode `getId()` gibt die ID des Code-Elements zurück.

Die Methode `getName()` gibt den Namen des Code-Elements zurück.

Die Methode `isClass()` gibt `true` zurück, wenn das Code-Element eine Klasse ist.

Die Methode `isMethod()` gibt `true` zurück, wenn das Code-Element eine Methode ist.

Die Methode `getPackageName()` gibt den Namen des Packages zurück.

7.2.3.7 IDescription

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die die vorhandenen Beschreibungsinformationen speichert.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface IDescription extends Serializable {
   public String getBookmark();
   public String getDockey();
   public String getReqweblink();
   public String getReqID();
   public List<Attribute> getAttributes();
   public List<History> getHistories();
   public String getDescription();
}
```

Listing 12: Interface IDescription

Die Methode `getBookmark()` gibt den Bookmark der Anforderung in einem Dokument zurück.

Die Methode `getDockey()` gibt den Schlüssel des Dokumentes in dem die Anforderung beschrieben ist zurück.

Die Methode `getReqweblink()` gibt den Link zu der Webseite zurück wo die Anforderung beschrieben ist.

Die Methode `getReqID()` gibt die ID der Anforderung zurück.

Die Methode `getAttributes()` gibt eine Liste der Attribute der Anforderung zurück.

Die Methode `getHistories()` gibt eine Liste der History – Einträge zurück.

Die Methode `getDescription()` gibt die ausführliche Beschreibung der Anforderung zurück.

7.2.3.8 IAttribute

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die die vorhandenen Attribute speichert.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface IAttribute extends Serializable {
   public String getName();
   public String getValue();
}
```

Listing 13: Interface IAttribute

Die Methode `getName()` gibt den Attributnamen zurück.

Die Methode `getValue()` gibt den Wert des Attributes zurück.

7.2.3.9 IHistory

Dieses Interface beschreibt die Methoden, die jene Klasse implementieren muss, die die vorhandenen History - Einträge speichert.

Das folgende Listing zeigt das Interface mit seinen Methoden.

```
public interface IHistory extends Serializable {
   public String getChangeDescr();
   public String getDate();
   public String getRequirementText();
   public String getRevision();
   public String getUser();
   public String getVersionLabel();
}
```

Listing 14: Interface IHistory

Die Methode `getChangeDescr()` gibt die Beschreibung der Änderung zurück.

Die Methode `getDate()` gibt das Datum der Änderung zurück.

Die Methode `getRequirementText()` gibt die Beschreibung der Anforderung zurück.

Die Methode `getRevision()` gibt die Revisionsnummer der Änderung zurück.

Die Methode `getUser()` gibt den Usernamen zurück, der die Änderung vorgenommen hat.

Die Methode `getVersionLabel()` gibt die Versionsbezeichnung zurück.

8 Zusammenfassung

Requirements Engineering ist eine Disziplin, die sich mit der Erhebung und Verwaltung von Anforderungen an ein Softwareprodukt beschäftigt. Auf Grund von sich häufig ändernden Anforderungen ist deren Verfolgung (Requirements Tracing) ein wichtiges Instrument um Auswirkungen dieser Änderungen effizient abschätzen zu können. Außerdem ist Requirements Tracing von verschiedenen Standards, wie zum Beispiel CMMI vorgeschrieben und stellt vor allem in sicherheitskritischen Projekten ein Muss dar.

Requirements Tracing ist eine große Unterstützung für viele Aufgaben in der Softwareentwicklung. Change Impact Analysen sind ein Beispiel dafür. Dabei muss der Projektmanager alle von einem Änderungswunsch betroffenen Artefakte identifizieren und den Aufwand für dessen Umsetzung kalkulieren. Liegen die Informationen über alle Abhängigkeiten aller Artefakte vor, so ist es ein Leichtes die betroffenen Dokumente zu identifizieren. Es gibt jedoch einige Herausforderungen beim Requirements Tracing. Ansätze und Methoden, die eine automatisierte Erstellung dieser Abhängigkeitsinformationen beschreiben, haben den Weg in die Praxis noch nicht gefunden. Bislang werden die Abhängigkeiten manuell mit diversen Tools erstellt. Dies ist meist mit enormem Aufwand verbunden, weshalb oft auf das Requirements Tracing verzichtet wird.

In meiner Arbeit habe ich ein in die Entwicklungsumgebung integriertes Plugin entwickelt, dass es dem Entwickler ermöglicht, die Abhängigkeiten zwischen den Anforderungen und den Source Code Elementen mit einem einfachen Doppelklick zu erstellen. Dies bringt folgende Vorteile:

- *Aufwandsreduktion bei der Erstellung von Abhängigkeiten:*
  Durch das integrierte Plugin kann der Entwickler die Abhängigkeiten gleich während der Implementierung in seiner gewohnten Entwicklungsumgebung mit einem einfachen Doppelklick erstellen.

- *Erhöhter Nutzen für die Entwickler:*
  Die Features des RT-Plugins bieten dem Entwickler eine Vielzahl von Informationen direkt in seine Entwicklungsumgebung. So kann er sich eine ausführliche Beschreibung jeder Anforderung anzeigen lassen. Er sieht auf einem Blick welche Anforderung wo implementiert wurde, da bei jeder Anforderung die Klassennamen jener Klassen angezeigt werden, die mit der Anforderung verknüpft sind. Es wird optisch dargestellt, wenn sich eine Anforderung im Anforderungsmanagement Tool geändert hat, usw.
- *Erhöhte Vollständigkeit und Korrektheit der erstellten Abhängigkeiten:*
  Durch die Aufwandsreduktion und den zusätzlichen Nutzen, die das RT-Plugin dem Entwickler bietet, wird die Motivation und die Bereitschaft der Entwickler gesteigert. Aus der gesteigerten Bereitschaft die Abhängigkeiten zu erstellen folgt ein höherer Grad der Vollständigkeit und der Korrektheit der erstellten Abhängigkeiten.

Das RT-Plugin wurde bei Siemens PSE von Requirements Engineering Experten begutachtet und evaluiert. Dabei ergaben sich einige Verbesserungsvorschläge, um die das RT-Plugin erweitert wurde. Eine Erstevaluierung zeigte, dass durch das RT-Plugin der Aufwand für die Erstellung der Verknüpfungen auf ein Sechstel reduziert werden kann. Ein detailliertes Fallstudien Projekt bei Siemens PSE, deren Evaluierungskonzept in meiner Arbeit beschrieben wurde, wird diese Ergebnisse überprüfen und das RT-Plugin noch genauer evaluieren.

Basierend auf den bisherigen Ergebnissen der Erstevaluierung und der Requirements Engineering Experten der Siemens PSE kann der plugin-basierte Ansatz und das RT-Plugin als eine viel versprechende Lösung im Requirements Tracing angesehen und weiter verfolgt werden.

9 Ausblick

Die Erstevaluierung zeigte eine enorme Aufwandsreduktion in der Erstellung von Verknüpfungen und zeigt, dass der plugin-basierte Ansatz und das RT-Plugin ein viel versprechender Ansatz des Requirements Tracings ist. Die umgesetzten Verbesserungen und hinzugefügten Funktionen führten zu einem Tool, das bereit für den praktischen Einsatz und der Durchführung einer umfassenden und detaillierten Fallstudie ist. Ein Evaluierungskonzept für diese Fallstudie, die im Rahmen eines Projektes bei Siemens PSE durchgeführt werden soll, ist in Abschnitt 4.3 beschrieben. Dabei kann die Aufwandsreduktion und der Grad der Vollständigkeit und Korrektheit der erstellten Verknüpfungen konkret gemessen und mit anderen Tools und Methoden verglichen werden.

Ein weiterer Schritt ist die Weiterentwicklung des RT-Plugins. Ein wichtiger Punkt hierfür ist die Automatisierung der Aktualisierung der Ansicht und somit der Anzeige der aktuellen Suspect-Markierung. Dies ist besonders in der verteilten Entwicklung notwendig, da so der Entwickler gleich über jede Änderung informiert wird und sich diese Änderungen ansehen kann. Eine Möglichkeit dies zu realisieren ist ein automatischer Anstoß der Aktualisierung durch einen Timer. Dann wird in bestimmten definierbaren Abständen die Aktualisierung durchgeführt. Eine andere Möglichkeit wäre durch ein eigenes Notifizierungssystem, indem bei jeder Änderung eine Notifizierung an alle Beteiligten Tool versendet wird. Diese Tools können dann entsprechend auf diese Nachrichten reagieren und zum Beispiel ein erneutes Laden der Anforderungen anstoßen.

In der Siemens PSE wird gerade eine Erweiterung des RT-Plugins entwickelt, mit der die Ergebnisse der JUnit-Tests ebenfalls in das Anforderungsmanagement Tool RequisitePro importiert werden kann. Dadurch wird der Kreislauf des Tracings beginnend mit der Definition der Benutzeranforderung, die Erstellung der Entwicklungsanforderugen, die Verknüpfung mit den Source Code und den Testfällen bis hin zu deren Ergebnissen möglich.

Literaturliste

[1] S. Ahn, K, Chong, "A Feature-Oriented Requirements Tracing Method: A Study of Cost-benefit Analysis", 2006 International Conference on Hybrid Information Technology,

[2] J. Cleland-Huang, G. Zemont, W. Lukasik, "A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability", Proceedings of the 12th IEEE International Requirements Engineering Conference,

[3] A. Egyed, "A Scenario-Driven Approach to Traceability", Proceedings of the 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, pp. 123-132,May 2001

[4] A. Egyed, P Grünbacher, “A Scenario-Driven Approach to Trace Dependency Analysis”, Transactions on software engineering 116-132, 2003

[5] A. Egyed, P. Grünbacher, "Automating Requirements Traceability: Beyond the Record & Replay Paradigm", Proceedings 17th International Conference on Automated Software Engineering, ASE 2002, pp. 163-171.Edinburgh, IEEE Computer Society,

[6] M.W. Evans, "The Software Factory", John Wiley and Sons, 1989

[7] B.Freimut, T.Punter, S.Biffl, M.Ciolkowski, "State-of-the-Art in Empirical Studies", VISEK Technical Report 007/E, 2002

[8] A Q Gates, O Mondragon, “FasTLInC: a constraint-based tracing approach”. Journal of Systems and Software 7, 2002

[9] O. C. Z. Gotel, A. C. W. Finkelstein, "An analysis of the requirements traceability problem", 1st International Conference on Requirements Engineering, pp.94-101, 1994

[10] J. Huffman Hayes, A. Dekhtyar, S. K.Sundaram, “Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods”. Trans-

actions on Software Engineering 4-19, 2006

[11] J. Huffman Hayes, A. Dekhtyar, S. K. Sundaram, S. Howard, “Helping Analysts Trace Requirements: An Objective Look”, IEEE International Conference on Requirements Engineering, 249-259, 2004

[12] M.Heindl, S.Biffl, "A Case Study on Value-based Requirements Tracing", Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 2005

[13] M.Heindl, S.Biffl, F.Reinisch, "Integrated Developer Tool Support for More Efficient Requirements Tracing and Change Impact Analysis", Technical Report TU Wien Quality Software Engineering, 2007

[14] J. Jackson, "A Keyphrase Based Traceability Scheme", IEE Colloquium on Tools and Techniques for MaintainingTraceability during Design, pp.2-1-2/4, 1991

[15] H. Kaindl, "The Missing Link in Requirements Engineering", ACM SigSoft Software Engineering Notes, vol. 18, no. 2, pp. 30-39,1993

[16] G. Kotonya, I. Sommerville, "Requirements Engineering Processes and Techniques", John Wiley & Sons Ltd, The Atrium, Southern Gate, 1998

[17] M. Lefering, "An Incremental Integration Tool between Requirements Engineering and Programming in the Large", Proceedings of the IEEE International Symposiumon Requirements Engineering, San Diego, California,Jan. 4-6, pp. 82-89,1993

[18] J Lin, C Chou Lin, J Cleland-Huang, Raffaella Settimi, et al., “Poirot: A Distributed Tool Supporting Enterprise-Wide Automated Traceability”, IEEE International Conference on Requirements Engineering, 2006

[19] I Macfarlane, I Reilly, “Requirements Traceability in an Integrated Development Environment”, International Conference on Requirements Engineering 116-127, 1995

[20] C. Neumüller, P. Grünbacher, "Automating Software Traceability in Very Small Companies: A Case Study and Lessons Learned", 21st IEEE International Conference on Automated Software Engineering (ASE'06), 2006

[21] D Chi-Liang Ni, “Enumeration and traceability tools for UNIX™ and WINDOWS™ environments”, Journal of Systems and Software 39, 1997

[22] F.A.C. Pinheiro, J. A.Goguen, "An Object-Oriented Tool for Tracing Requirements", IEEE Software 13(2), 52-64, 1996

[23] B. Ramesh, T. Powers, C. Stubbs, M. Edwards, “Implementing Requirements Traceability; A Case Study”, IEEE International Conference on Requirements Engineering 89-99, 1995

[24] Rational RequisitePro, http://www-306.ibm.com/software/awdtools/reqpro/, 2006.

[25] X Song, B Hasling, G Mangla, B Sherman, “Lessons learned from building a web-based requirements tracing system”, International Conference on Requirements Engineering, 1998

[26] Telelogic product DOORS, http://www.telelogic.com/products/doorsers/doors/index.cfm, 2005.

Appendix

A Beispiel einer Schnittstelle

In diesem Abschnitt wird die Erstellung einer einfachen Erweiterung an Hand eines Beispiels, der RTSampleExtension Erweiterung, beschreiben. Dies soll den Einsatz und die Art wie die in den vorigen Abschnitten beschriebenen Interfaces zu verwenden sind.

A.1 Erstellung eines Plugin Projektes

Um eine Erweiterung zu dem RT-Plugin zu entwickeln muss der Entwickler zuerst ein neues Plugin Projekt in Eclipse erstellen. Dies kann mit Hilfe des Assistenten, den Eclipse zur Verfügung stellt durchgeführt werden. Dabei erstellt Eclipse die notwendige Projektstruktur und die Dateien für die Plugin Einstellungen. Bevor die Einstellungen vorgenommen werden können muss sichergestellt sein, dass das RT-Plugin installiert ist oder es zu den Bibliotheken in den Projekteinstellungen hinzugefügt wurde. Wenn dies geschehen ist, dann kann der Entwickler die Plugin Einstellungen vornehmen. Dazu öffnet er die „plugin.xml“ Datei, die in der Projektstruktur enthalten ist. Eclipse bietet einen eigenen Editor für die Bearbeitung. In diesem Editor gibt es mehrere Register auf denen verschiedene Einstellungen vorgenommen werden können. Auf dem ersten Register „Overview“ werden die grundlegenden Einstellungen, wie der Name die ID und die Version definiert. Auf dem zweiten Register „Dependencies“ werden die Bibliotheken (Libraries), die das Plugin benötigt und verwendet aufgelistet. Auf der linken Seite, wo die „Required Plug-ins“ eingetragen werden, muss auch das RT-Plugin hinzugefügt werden. Auf dem Register „Extensions“ können dann die Einstellungen der Extension Points vorgenommen werden. Hier kann der Entwickler den Extension Point des RT-Plugins aus einer Liste auswählen und hinzufügen. Wenn die Einstellungen richtig vorgenommen wurden, sollte das Register „plugin.xml“ den folgenden Inhalt aufweisen.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         id="RTSampleExtension"
         name="RT Sample Extension"
         point="RT.rtextension">
      <rtextension name="Sample Extension">
         <import
               class="rte.RTImport"
               name="RTImport"/>
         <export
               class="rte.RTExport"
               name="RTExport"/>
         <importpropertypage
               class="rte.RTImportPropertyPage"
               name="RTImportPropertyPage"/>
         <exportpropertypage
               class="rte.RTExportPropertyPage"
               name="RTExportPropertyPage"/>
      </rtextension>
   </extension>
</plugin>
```

Listing 15: plugin.xml der RTSampleExtension

Die Klassennamen können natürlich abweichend definiert werden. Wichtig ist nur dass die Struktur der Tags übereinstimmt und dass im Tag `<extension>` das Attribut point gleich RT.rtextension ist. Danach kann mit der Implementierung der hier definierten Klassen begonnen werden.

A.2 Implementierung der Daten - Klassen

In diesem Abschnitt werden die Datenklasse der RTSampleExtension Erweiterung angeführt. Diese Klassen werden für den Datenaustausch zwischen dem RT-Plugin und der Erweiterung, die die Informationen über die Anforderungen und den bestehenden Verknüpfungen aus dem Anforderungsmanagement Tool bereitstellt und die neuen Verknüpfungen in das Anforderungsmanagement Tool importiert.

Requirement

Die Klasse Requirement speichert die Informationen einer Anforderung und muss das Interface IRequirement implementieren. Das folgende Listing zeigt diese Klasse mit ihren Methoden.

```
package rt.data;

import rt.data.interfaces.IRequirement;

public class Requirement implements IRequirement{

  private static final long serialVersionUID = 9;
  private String description;
```

```
private String name;
private String prefix;
private String id;
private Object element;
private String typeid;
private String newname = "";
private String newprefix = "";
private boolean suspect = false;
private boolean hasDocument;
private String docKey;
private String bookmark;
private String reqweblink;


public Requirement(){

}
public Requirement(int _typeid,int _id,String _name,
     String _desc,String _prefix){
   typeid = String.valueOf(_typeid);
   id = String.valueOf(_id);
   name = _name;
   description = _desc;
   prefix = _prefix;
}
public String getDescription() {
   return description;
}

public String getId() {
   return id;
}
public String getName() {
   return name;
}
public String getPrefix() {
   return prefix;
}
public String getTypeId() {
   return typeid;
}
public Object getIJavaElement() {
   return element;
}
public void setDescription(String description) {
   this.description = description;
}
public void setId(String id) {
   this.id = id;
}
public void setName(String name) {
   this.name = name;
}
public void setIJavaElement(Object el) {
   element = el;
}
public void setPrefix(String prefix) {
   this.prefix = prefix;
}
public void setTypeid(String typeid) {
   this.typeid = typeid;
}
```

```
    public void setNewName(String n){
      newname = n;
    }
    public void setIsSuspect(boolean _suspect){
      suspect = _suspect;
    }
    public String getNewName(){
      return newname;
    }
    public void setNewPrefix(String p){
      newprefix = p;
    }
    public String getNewPrefix(){
      return newprefix;
    }
    public boolean isSuspect(){
      return suspect;
    }
    public boolean hasDocument(){
      return hasDocument;
    }
    public void setDocument(boolean val){
      hasDocument = val;
    }
    public void setDocKey(String val){
      docKey = val;
    }
    public String getDocKey(){
      return docKey;
    }
    public void setBookmark(String val){
      bookmark = val;
    }
    public String getBookmark(){
      return bookmark;
    }
    public String getReqWebLink(){
      return reqweblink;
    }
    public void setReqWebLink(String l){
      reqweblink = l;
    }
}
```

Listing 16: Klasse Requirement

TransferCodeElement

Die Klasse TransferCodeElement speichert die Informationen einer Methode oder einer Klasse und muss das Interface ITransferCodeElement implementieren. Das folgende Listing zeigt diese Klasse mit ihren Methoden.

```
package rt.data;
import rt.data.interfaces.ITransferCodeElement;

public class TransferCodeElement
    implements ITransferCodeElement{
  private static final long serialVersionUID = 8;
  private String id;
```

```
    private String name;
    private boolean isClass;
    private boolean isMethod;
    private String packagename;

    public String getId() {
      return id;
    }
    public String getName() {
      return name;
    }
    public boolean isClass() {
      return isClass;
    }
    public boolean isMethod() {
      return isMethod;
    }
    public void setId(String id) {
      this.id = id;
    }
    public void setIsClass(boolean isClass) {
      this.isClass = isClass;
      this.isMethod = !isClass;
    }
    public void setIsMethod(boolean isMethod) {
      this.isMethod = isMethod;
      this.isClass = !isMethod;
    }
    public void setName(String name) {
      this.name = name;
    }
    public void setPackageName(String pn){
      packagename = pn;
    }
    public String getPackageName(){
      return packagename;
    }
}
```

Listing 17: Klasse TransferCodeElement

Type

Die Klasse Type speichert die Informationen eines Anforderungstypen und muss das Interface IType implementieren. Das folgende Listing zeigt diese Klasse mit ihren Methoden.

```
package rt.data;
import rt.data.interfaces.IType;

public class Type implements IType{
  private static final long serialVersionUID = 12;
  private int key;
  private String name;
  private String prefix;

  public Type(int key, String name, String prefix) {
    this.key = key;
    this.name = name;
```

```
        this.prefix = prefix;
    }
    public int getKey() {
        return key;
    }
    public String getId(){
        return String.valueOf(key);
    }
    public void setKey(int key) {
        this.key = key;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPrefix() {
        return prefix;
    }
    public void setPrefix(String prefix) {
        this.prefix = prefix;
    }
}
```

Listing 18: Klasse Type

TraceabilityInformation

Die Klasse TraceabilityInformation speichert die Informationen einer Verknüpfung und muss das Interface ITraceabilityInformation implementieren. Das folgende Listing zeigt diese Klasse mit ihren Methoden.

```
package rt.data;
import rt.data.interfaces.ITraceabilityInformation;

public class TraceabilityInformation
    implements ITraceabilityInformation{

    private static final long serialVersionUID = 10;
    private String codeid;
    private String reqid;

    public String getCodeElementId() {
        return codeid;
    }
    public String getRequirementId() {
        return reqid;
    }
    public void setCodeElementId(String id){
        codeid = id;
    }
    public void setRequirementId(String id){
        reqid = id;
    }
}
```

Listing 19: Klasse TraceabilityInformation der RTSampleExtension

TransferTraceInformation

Die Klasse TransferTraceInformation speichert die schon vorhandenen Verknüpfungen des Anforderungsmanagement Tools und muss das Interface ITransferTraceInformation implementieren. Das folgende Listing zeigt diese Klasse mit ihren Methoden.

```
package rt.data;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import rt.data.interfaces.ITransferTraceInformation;

public class TransferTraceInformation
  implements ITransferTraceInformation {

  private static final long serialVersionUID = 123;
  private List<TransferCodeElement> codeelements =
    Collections.synchronizedList(
      new ArrayList<TransferCodeElement>());
  private List<TraceabilityInformation> traces =
    Collections.synchronizedList(
      new ArrayList<TraceabilityInformation>());

  public List<TransferCodeElement> getCodeElements() {
    return codeelements;
  }
  public List<TraceabilityInformation>
    getTraceabilityInformation() {
    return traces;
  }
  public void addCodeElement(TransferCodeElement el){
    codeelements.add(el);
  }
  public void addTraceabilityInformation(
      TraceabilityInformation ti){
    traces.add(ti);
  }
}
```

Listing 20: Klasse TransferTraceInformation

A.3 Implementierung der Schnittstellenklassen

In diesem Abschnitt werden die Klassen beschreiben, die mit dem RT-Plugin kommunizieren. Das heißt das RT-Plugin ruft die Methoden, die in den Interfaces beschrieben sind, auf und bekommt so die Daten für die Anzeige und die Erstellung von neuen Verknüpfungen sowie für die Konsistenzprüfung.

RTImportPropertyPage

Diese Klasse erweitert das Register „Import“ in den Projekteinstellungen mit den Eingabefeldern, die für die Einstellung der Erweiterung benötigt werden. In diesen Ein-

gabefeldern werden zum Beispiel die Informationen für den Zugriff auf das jeweilige Anforderungsmanagement Tool definiert. Die Klasse muss das Interface IImportPropertyPage implementieren. Bei der RTSampleExtension Erweiterung wird hier nur ein Eingabefeld für die Datei, deren Daten importiert werden sollen, erstellt. Das folgende Listing zeigt diese Klasse und deren Methoden.

```
package rte;

import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IConfigurationElement;
import org.eclipse.core.runtime.IExecutableExtension;
import org.eclipse.core.runtime.QualifiedName;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Text;
import rt.extension.interfaces.IImportPropertyPage;

public class RTImportPropertyPage implements
     IImportPropertyPage,IExecutableExtension{

  private static final String PROJECT_TITLE = "Import File:";
  public static final String PROJECT_PROPERTY = "IMPORTFILE";
  private static final String PROJECT_DEFAULT = "";
  private static final int PROJECT_FIELD_WIDTH = 200;
  private Text projText;
  private String proj;

  public void setInitializationData(
    IConfigurationElement member, String propertyName,
    Object data) throws CoreException {

  }
  public void addPropertyPage(IResource resource,
    Composite parent){
    Composite composite = new Composite(parent,SWT.NONE);
    GridLayout gl = new GridLayout();
    gl.numColumns = 3;
    composite.setLayout(gl);

    Label projLabel = new Label(composite, SWT.NONE);
    projLabel.setText(PROJECT_TITLE);
    projText = new Text(composite,
                              WT.SINGLE | SWT.BORDER);
    GridData gd =
      new GridData(GridData.HORIZONTAL_ALIGN_FILL);
    gd.widthHint = PROJECT_FIELD_WIDTH;
    projText.setLayoutData(gd);

    try {
      proj = resource.getPersistentProperty(
        new QualifiedName("", PROJECT_PROPERTY));
```

```
            projText.setText(
               (proj != null) ? proj : PROJECT_DEFAULT);
         } catch (CoreException e) {
            projText.setText(PROJECT_DEFAULT);
         }
         final FileDialog fd = new FileDialog(parent.getShell(),
                                                    SWT.OPEN);
         String[] filetype = new String[]{"*.txt"};
         fd.setFilterExtensions(filetype);
         final Button button = new Button(composite,SWT.PUSH);
         button.setText("Browse");

         Listener listener = new Listener() {
            public void handleEvent(Event event) {
               if (event.widget == button) {
                  String file = fd.open();
                  if(file != null){
                   projText.setText(file);
                   projText.update();
                   proj = projText.getText();
                  }
               }
            }
         };
         button.addListener(SWT.Selection, listener);
         composite.pack();
      }

      public void performDefault(IResource resource){
         projText.setText(PROJECT_DEFAULT);
      }

      public boolean performOk(IResource resource){
         try {
            resource.setPersistentProperty(
               new QualifiedName("", PROJECT_PROPERTY),
               projText.getText());

         } catch (CoreException e) {
            return false;
         }
         return true;
      }
   }
```

Listing 21: Klasse RTImportPropertyPage der RTSampleExtension

In der Methode `addPropertyPage()` wird ein Eingabefeld erstellt, wo die genaue Pfadangabe der Datei eingegeben werden kann, welche die Importdaten speichert. Um dies nicht nur manuell zu gestalten wird auch ein Button erstellt, der ein Dialogfenster öffnet in dem der Benutzer nach der Datei suchen und auswählen kann.

Die Methode `performDefault()` wird das Eingabefeld auf den Default Wert zurückgestellt.

In der Methode `performOk()` wird der Wert des Eingabefeldes in den Projekteinstellungen gespeichert.

RTExportPropertyPage

Diese Klasse erweitert das Register „Export“ in den Projekteinstellungen mit den Eingabefeldern, die für die Einstellung der Erweiterung benötigt werden. In diesen Eingabefeldern werden zum Beispiel die Informationen für den Zugriff auf das jeweilige Anforderungsmanagement Tool für den Export der Daten definiert. Die Klasse muss das Interface IExportPropertyPage implementieren. Bei der RTSampleExtension Erweiterung wird hier nur ein Eingabefeld für die Datei, in die die Verknüpfungsinformationen exportiert werden sollen, erstellt. Das folgende Listing zeigt diese Klasse und deren Methoden.

```
package rte;

import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IConfigurationElement;
import org.eclipse.core.runtime.IExecutableExtension;
import org.eclipse.core.runtime.QualifiedName;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Text;
import rt.extension.interfaces.IExportPropertyPage;

public class RTExportPropertyPage implements
     IExportPropertyPage,IExecutableExtension{

  private static final String PROJECT_TITLE = "Export File:";
  public static final String PROJECT_PROPERTY = "EXPORTFILE";
  private static final String PROJECT_DEFAULT = "";
  private static final int PROJECT_FIELD_WIDTH = 200;
  private Text projText;
  private String proj;

  public void setInitializationData(
     IConfigurationElement member,
     String propertyName, Object data)
     throws CoreException {
  }

  public void addPropertyPage(IResource resource,
     Composite parent){
     Composite composite = new Composite(parent,SWT.NONE);
     GridLayout gl = new GridLayout();
     gl.numColumns = 3;
```

```
        composite.setLayout(gl);

        Label projLabel = new Label(composite, SWT.NONE);
        projLabel.setText(PROJECT_TITLE);

        projText = new Text(composite,
             SWT.SINGLE | SWT.BORDER);
        GridData gd =
          new GridData(GridData.HORIZONTAL_ALIGN_FILL);
        gd.widthHint = PROJECT_FIELD_WIDTH;
        projText.setLayoutData(gd);

        try {
          proj = resource.getPersistentProperty(
            new QualifiedName("", PROJECT_PROPERTY));
          projText.setText(
            (proj != null) ? proj : PROJECT_DEFAULT);
        } catch (CoreException e) {
          projText.setText(PROJECT_DEFAULT);
        }
        final FileDialog fd =
          new FileDialog(parent.getShell(), SWT.OPEN);
        String[] filetype = new String[]{"*.*"};
        fd.setFilterExtensions(filetype);
        final Button button = new Button(composite,SWT.PUSH);
        button.setText("Browse");

        Listener listener = new Listener() {
          public void handleEvent(Event event) {
            if (event.widget == button) {
              String file = fd.open();
              if(file != null){
               projText.setText(file);
               projText.update();
               proj = projText.getText();
              }
            }
          }
        };
        button.addListener(SWT.Selection, listener);
        composite.pack();
      }

      public void performDefault(IResource resource){
        projText.setText(PROJECT_DEFAULT);
      }

      public boolean performOk(IResource resource){
        try {
          resource.setPersistentProperty(
            new QualifiedName("", PROJECT_PROPERTY),
            projText.getText());
        } catch (CoreException e) {
          return false;
        }
        return true;
      }
    }
```

Listing 22: Klasse RTExportPropertyPage

In der Methode `addPropertyPage()` wird ein Eingabefeld erstellt, wo die genaue Pfadangabe der Datei eingegeben werden kann, in die die Verknüpfungsinformationen gespeichert werden soll. Um dies nicht nur manuell zu gestalten wird auch ein Button erstellt, der ein Dialogfenster öffnet in dem der Benutzer nach einer Datei suchen und auswählen kann.

Die Methode `performDefault()` wird das Eingabefeld auf den Default Wert zurückgestellt.

In der Methode `performOk()` wird der Wert des Eingabefeldes in den Projekteinstellungen gespeichert.

RTImport

In dieser Klasse werden die Daten für die Anzeige der Anforderungen und der Konsistenzprüfung aufbereitet. Dazu werden diese aus dem jeweiligen Anforderungsmanagement Tool ausgelesen und in das den Interfaces entsprechende Datenformat konvertiert. Die Klasse muss das Interface IImport implementieren. Bei der RTSampleExtension Erweiterung werden die Daten der Anforderungen, der Anforderungstypen und der bestehenden Verknüpfungsinformationen aus einer Datei ausgelesen. Das folgende Listing zeigt die Klasse mit ihren Methoden.

```
package rte;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IConfigurationElement;
import org.eclipse.core.runtime.IExecutableExtension;
import org.eclipse.core.runtime.QualifiedName;
import rt.extension.interfaces.IImport;
import rt.data.interfaces.IRequirement;
import rt.data.interfaces.ITraceInformation;
import rt.data.interfaces.IType;

public class RTImport implements IImport,IExecutableExtension {

  public ArrayList<IRequirement> getRequirements(
  IProject project) throws Exception{
    String filePath = null;
    try{
      if(project != null){
        filePath = project.getPersistentProperty(
          new QualifiedName("",
          RTImportPropertyPage.PROJECT_PROPERTY));
      }
    }catch(CoreException e){
      throw new Exception("Can't get requirements! "
```

```
            +e.getMessage());
        }
        if(filePath != null){
          ArrayList<IRequirement> reqList =
            new ArrayList<IRequirement>();
          File file = new File(filePath);
          if(file.exists()){
            BufferedReader br = new BufferedReader(
              new FileReader(file));
            String line = "";
            while((line=br.readLine())!=null){
              if(line.contains("BEGIN REQUIREMENTS")){
               while((line=br.readLine())!=null){
                     if(!line.contains("END REQUIREMENTS")){
                           String[] req = line.split(";");
                           IRequirement r = new Requirement(
                                 req[0],req[1],req[2],
                                 req[3],req[4]);
                           reqList.add(r);
                     }else{
                           break;
                     }
               }
              }
            }
            br.close();
          }
          return reqList;
        }else{
          return null;
        }
      }

      public ArrayList<IType> getTypes(IProject project)
        throws Exception{
        String filePath = null;
        try{
          if(project != null){
            filePath = project.getPersistentProperty(
              new QualifiedName("",
              RTImportPropertyPage.PROJECT_PROPERTY));
          }
        }catch(CoreException e){
          throw new Exception("Can't get requirements! "
            +e.getMessage());
        }
        if(filePath != null){
          ArrayList<IType> typeList =
            new ArrayList<IType>();
          File file = new File(filePath);
          if(file.exists()){
            BufferedReader br = new BufferedReader(
              new FileReader(file));
            String line = "";
            while((line=br.readLine())!=null){
              if(line.contains("BEGIN TYPES")){
               while((line=br.readLine())!=null){
                     if(!line.contains("END TYPES")){
                           String[] type = line.split(";");
                           IType t = new Type(type[0],
                                 type[2],type[1]);
                           typeList.add(t);
```

```
                    }else{
                          break;
                    }
                 }
               }
            }
            br.close();
          }
          return typeList;
        }else{
          return null;
        }
      }
      public Description getDescription(IProject project,
        String ReqID) throws Exception {
        return null; //TODO return Description Object
      }

      public void clearSuspect(IProject proj, String ReqID)
        throws Exception {
        //TODO clear Suspect
      }
      public void setInitializationData(IConfigurationElement
        member, String propertyName, Object data)
        throws CoreException {
      }
    }
```

Listing 23: Klasse RTImport der RTSampleExtension

Die Methode `getRequirements()` liest den Dateinamen der Importdatei aus den Projekteinstellungen des übergebenen Projektes aus. Dann wird die Datei geöffnet und es werden die Anforderungen ausgelesen und in einer Liste gespeichert, die als Rückgabewert der Methode dient.

Die Methode `getTypes()` liest die Typen aus der Importdatei und speichert diese ebenfalls in einer Liste, die als Rückgabewert dient.

Die Methode `getDescription()` soll die vollständige Beschreibung inklusive der Attribute und der History zurückgeben.

Die Methode `clearSuspect()` wird in der SampleExtension Erweiterung nicht verwendet. Sonst löscht diese Methode das Attribut, dass anzeigt, dass sich eine verknüpfte Anforderung geändert hat.

RTExport

Diese Klasse übernimmt den Export der Verknüpfungsinformationen. Es werden hier also die übergebenen Verknüpfungen in das ausgewählte Anforderungsmanagement Tool übertragen, wo diese dann angezeigt werden können. Bei der RTSampleExtension werden diese Verknüpfungsinformationen einfach in einer Datei gespeichert. Die

Klasse muss das Interface IExport implementieren. Das folgende Listing zeigt diese Klasse und deren Methoden.

```
package rte;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.HashMap;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IConfigurationElement;
import org.eclipse.core.runtime.IExecutableExtension;
import org.eclipse.core.runtime.QualifiedName;
import rt.data.interfaces.ICodeElement;
import rt.extension.interfaces.IExport;
import rt.data.interfaces.IRequirement;
import rt.data.interfaces.ITraceabilityInformation;
import rt.data.interfaces.IType;

public class RTExport implements IExport,IExecutableExtension{

   public void exportInformation(
     ArrayList<ITraceabilityInformation> tilist,
     HashMap<String,ICodeElement> clist,
     HashMap<String,IRequirement> rlist,
     HashMap<String,IType> tlist, IProject project)
     throws Exception {
     if(project != null){
       String filename = null;
       try{
         filename = project.getPersistentProperty(
           new QualifiedName("",
           RTExportPropertyPage.PROJECT_PROPERTY));
         if(filename != null){
           File file = new File(filename);
           if(file.exists()){
            file.delete();
           }
           file.createNewFile();
           BufferedWriter bw = new BufferedWriter(
            new FileWriter(file));
           bw.write("BEGIN TYPES\n");
           Iterator it = tlist.keySet().iterator();
           while(it.hasNext()){
            IType type = tlist.get(it.next());
            bw.append(type.getId()+";"+type.getPrefix()+";"
                  +type.getName()+"\n");
           }
           bw.append("END TYPES\n\n");
           bw.append("BEGIN REQUIREMENTS\n");
           Iterator iter =    rlist.keySet().iterator();
           while(iter.hasNext()){
            IRequirement req = rlist.get(iter.next());
            bw.append(req.getId()+";"+req.getPrefix()+";"
                  +req.getName()+";"+req.getTypeId()+";"
                  +req.getDescription()+"\n");
           }
           bw.append("END REQUIREMENTS\n\n");
           bw.append("BEGIN CODEELEMENTS\n");
           Iterator ite = clist.keySet().iterator();
```

```
            while(ite.hasNext()){
             ICodeElement el = clist.get(ite.next());
             bw.append(el.getId()+";"+el.getName()+";"
                   +el.isClass()+"\n");
            }
            bw.append("END CODEELEMENTS\n\n");
            bw.append("BEGIN TRACES\n");
            for(int i=0;i<tilist.size();i++){
             ITraceabilityInformation ti = tilist.get(i);
             String codeid = ti.getCodeElementId();
             String reqid =    ti.getRequirementId();
             bw.append(reqid+";"+codeid+"\n");
         }
            bw.append("END TRACES\n");
            bw.flush();
            bw.close();
         }
      }catch(Exception e){
         throw new Exception("Error during export! "
           +e.getMessage());
      }

   }
}

public TransferTraceInformation getTraceabilityInformation(
   IProject project, String classname) throws Exception{
   TransferTraceInformation traceInfo =
     new TransferTraceInformation();
   String filePath = null;
   try{
      if(project != null){
         filePath = project.getPersistentProperty(
           new QualifiedName("",
           RTExportPropertyPage.PROJECT_PROPERTY));
      }
   }catch(CoreException e){
      throw new Exception("Can't get requirements! "
        +e.getMessage());
   }
   if(filePath != null){
      File file = new File(filePath);
      if(file.exists()){
         BufferedReader br = new BufferedReader(
           new FileReader(file));
         String line = "";
         while((line=br.readLine())!=null){
            if(line.contains("BEGIN CODEELEMENTS")){
             while((line=br.readLine())!=null){
                   if(!line.contains("END CODEELEMENTS")){
                         String[] code =   line.split(";");
                         TransferCodeElement ce =
                               new TransferCodeElement();
                         ce.setId(code[0]);
                         ce.setName(code[1]);
                         ce.setIsClass(Boolean.parseBoolean(
                               code[2]));
                         traceInfo.  addCodeElement(ce);
                   }else{
                         break;
                   }
             }
            }
```

```
            }
            br.close();
            br = new BufferedReader(new FileReader(file));
            line = "";
            while((line=br.readLine())!=null){
               if(line.contains("BEGIN TRACES")){
                while((line=br.readLine())!=null){
                      if(!line.contains("END TRACES")){
                            String[] str = line.split(";");
                            TraceabilityInformation trace =
                                  new   TraceabilityInformation();
                            trace.setRequirementId(str[0]);
                            trace.setCodeElementId(str[1]);
                            traceInfo.addTraceabilityInformation(
                                  trace);
                      }else{
                            break;
                      }
                }
               }
            }
            br.close();
         }
      }
      return traceInfo;
   }
   public Map<String,List> getTraceInformationForRequirements(
      IProject proj, Object[] reqIDs) throws Exception{
      return null;
   }
   public void setInitializationData(IConfigurationElement
      member, String propertyName, Object data)
      throws CoreException {
   }
}
```

Listing 24: Klasse RTExport der RTSampleExtension

In der Methode `exportInformation()` werden die Verknüpfungsinformationen in einer Datei gespeichert. Der Parameter tilist enthält eine Liste von ITraceabilityInformation Objekten. Diese Objekte stehen je für eine Verknüpfung und speichern somit die ID der Anforderung und die ID des Code Elements. Wurde eine Anforderung mit einer Methode verknüpft, so wird zusätzlich eine Verknüpfung zu der zugehörigen Klasse der Methode erstellt.

Der Parameter `clist` enthält eine Liste von ICodeElement Objekten, die die Informationen zu den Code Elementen speichern.

Der Parameter `rlist` enthält eine Liste von IRequirement Objekten, die die Informationen der Anforderungen speichern.

Der Parameter `tlist` enthält eine Liste von IType Objekten, die die Informationen der Anforderungstypen speichern.

Die Methode `getTraceabilityInformation()` liest die vorhandenen Code Elemente (Klassen und Methoden) aus der Importdatei aus und speichert diese in der Liste des TransferTraceInformation Objektes. Weiters werden die vorhandenen Verknüpfungsinformationen ausgelesen und ebenso in einer Liste des TransferTraceInformation Objektes gespeichert. Dieses TransferTraceInformation Objekt, welches alle Informationen über die Verknüpfungen und den beteiligten Code Elementen beinhaltet, wird als Rückgabewert übergeben.

Die Methode `getTraceInformationForRequirements()` soll für jede Anforderung eine Liste mit vollständigen Namen der verknüpften Klassen zurückliefern.