Institut für Softwaretechnologie
Abteilung Software Engineering
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3693

Präsentation von Software Repository Mining in Eclipse

Mehmet Fatih Cicek

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Stefan Wagner
Betreuer:	M. Sc. Jasmin Ramadani
Begonnen am:	10. Oktober 2014
Beendet am:	10. April 2015
CR-Klassifikation:	D.2.3, D.2.6

Inhaltsverzeichnis

Danksagung

Zunächst möchte ich mich an dieser Stelle bei all denjenigen bedanken, dich mich während meines Studiums und meiner Diplomarbeit unterstützt und motiviert haben.

Mein besonderer Dank gilt meiner Familie, die mich nicht nur finanziell, sondern auch moralisch immer unterstützt und mir den Rücken gestärkt haben.

Großer Dank gebührt auch meiner Verlobten, die mich immer wieder ermutigte und stets ein offenes Ohr für mich hatte.

Weiterhin bedanke ich mich bei meinem Betreuer M. Sc. Jasmin Ramadani für die hervorragende Betreuung und freundliche Unterstützung während der gesamten Diplomarbeit.

Ebenfalls bedanken möchte ich mich bei meinem Referenten Herrn Prof. Dr. Stefan Wagner, der mir diese Arbeit ermöglicht hat.

Schließlich danke ich auch meinen Freunden und Kommilitonen für die schöne Zeit an der Universität Stuttgart.

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

SWT	Standard Widget Toolkit
GUI	Graphical User Interface
SQL	Structured Query Language
CLI	Command Line Interface
SRM	Software Repository Mining
PDE	Plug-In Development Environment
SPMF	Sequential Pattern Mining Framework
IDE	Integrated Development Environment
GPL	General Public License
EPL	Eclipse Public License

1 Einleitung

Software-Repositories spielen im Software Engeneering eine sehr wichtige Rolle, da sie sämtliche Informationen über die Entwicklung eines Softwaresystems beinhalten und somit eine Informationsquelle für die Softwareentwicklungsanalysen zur Verfügung stellen [1].

Die Ziele dieser Softwareentwicklungsanalysen sind vielfältig. Eines der Ziele ist die Analyse der gekoppelten Dateiänderungen mit Hilfe der Data Mining Technik namens "Frequent Itemset Mining" auf Basis der Software-Historie. Die Ergebnisse dieser Analyse sollen den Entwicklern hinsichtlich ihrer Modifikations- und Bugfixingsaufgaben eine Unterstützung anbieten.

1.1 Motivation

Modifikations- und Bugfixingsaufgaben bilden einen elementaren Bestandteil der Aufgaben eines Entwicklers, mit denen er öfters konfrontiert wird. Ändert beispielweise der Entwickler einen bestimmten Bereich eines Quellcodes oder ein File von einem Softwaresystem, so sind in der Regel weitere Änderungen mit dieser Änderung auch verbunden. Der Entwickler muss in diesem Zusammenhang auch wissen, welche weiteren Bereiche noch zu ändern sind. Vor allem für Entwickler, die neu sind und sich mit dem Softwaresystem nicht gut auskennen, ist es schwer die gekoppelten Dateiänderungen zu entdecken. Es existieren zwar Tools, die die Entwickler in dieser Hinsicht unterstützen sollen. Jedoch sind diese aber nur in der Lage einige interessante Codeänderungen oder Files anzuzeigen und nicht alle relevanten Codes bzw. Files, die geändert werden müssen.

Betrachtet man zum Beispiel gekoppelte Files, die in unterschiedlichen Programmiersprachen implementiert sind, so ist es mit einfachen Tools nicht möglich alle relevanten Files zu erhalten und führt in den meisten Fällen dazu, dass der Entwickler nach den entsprechen gekoppelten Files selber suchen muss oder dass er es vergisst diese File zu ändern. Dies ist eine sehr zeitaufwändige Arbeit für den Entwickler und natürlich auch sehr kostspielig für den Arbeitgeber [2].

1.2 Zielsetzung

Diese Diplomarbeit gliedert sich in zwei Bereiche, einen praktischen Bereich gefolgt von einem theoretischen Bereich.

Das Ziel des praktischen Bereiches ist die Entwicklung eines Eclipse Plug-Ins für die Unterstützung der Entwickler hinsichtlich ihrer Modifikations- und Bugfixingsaufgaben.

Das Plug-In soll die Durchführung der folgenden Punkte ermöglichen:

1. Ausführen der Frequent-Itemset-Analyse.
2. Anzeigen der gekoppelten Dateienänderungen und weiteren Informationen an dem Benutzer.

Der theoretische Teil dieser Diplomarbeit hingegen umfasst die nachfolgenden zwei Punkte:

1. Vergleich von Sequential Pattern Mining und Frequent Itemset Mining.
2. Fertigstellung der Ausarbeitung.

1.3 Gliederung

Kapitel 1 – Einleitung: Einführung in das Themengebiet gefolgt von der Motivation und Zielsetzung.

Kapitel 2 – Grundlagen: Darstellung der grundlegenden Themen, die für das bessere Verstehen der Diplomarbeit von entscheidender Bedeutung sind.

Kapitel 3 – Verwandte Projekte: Vorstellung der verwandten Projekte.

Kapitel 4 – Tools und Algorithmen: Repräsentation und Erläuterung der Tools und Algorithmen, die bei der Entwicklung des Eclipse Plug-Ins eingesetzt wurden.

Kapitel 5 – Anforderungen und Konzept: Festlegung der Anforderungen an das Eclipse Plug-In und Darstellung des Konzeptes.

Kapitel 6 – Implementierung: Implementierung des in Kapitel 5 dargestellten Konzeptes.

Kapitel 7 – Vergleich von Sequential Pattern Mining und Frequent Itemset Mining: Vergleich von diesen beiden Algorithmen auf theoretischer Ebene.

Kapitel 8 – Evaluation: Testen und Bewerten des im Rahmen dieser Diplomarbeit entwickelten Software Repository Mining (SRM) Plug-Ins.

Kapitel 9 – Zusammenfassung und Ausblick: Zusammenfassung der Resultate dieser Diplomarbeit und Vorstellung der Erweiterungspunkte.

2 Grundlagen

In diesem Kapitel findet eine Einführung in die grundlegenden Themen dieser Arbeit statt.

2.1 Software Repository Mining

Unter Software Repository Mining versteht man die Untersuchungen von Software-Repositories. Darin befinden sich sämtliche Daten, die während der Softwareentwicklung erzeugt und archiviert werden und geben somit einen eindeutigen Aspekt über die Art und Weise der Realisierung des Softwaresystems. Diese Daten, deren Existenz sich auf die Gesamtlaufzeit des Projektes beläuft, enthalten Informationen über Änderungen in der Projektentwicklung.

Die Extraktion relevanter Informationen und die Entdeckung der Zusammenhänge zwischen diesen extrahieren Informationen im Rahmen der Softwareentwicklung beruhen auf eine große Palette an Verfahren, die durch die Softwareentwickler entwickelt und experimentiert wurden. Da diese Verfahren sehr viele Parallelen zu Data Mining und Wissensentdeckung (Knowledge Discovery) aufweisen, wird Software Repository Mining gleichgestellt zu Data Mining und Wissensentdeckung (Knowledge Discovery). An dieser Stelle ist aber Aufschluss darüber zu geben, dass Software Repository Mining natürlich nicht auf Data Mining und Wissensentdeckung (Knowledge Discovery) begrenzt ist.

Somit bezweckt die Software Repository Mining die Aufbringung neuer Erkenntnisse in den Prozess der Softwareentwicklung und die Hervorhebung der im Laufe der Zeit aufgetretenen Änderungen. Dies wird durch die Aufdeckung von relevanten Informationen und Relationen zwischen diesen Informationen über einen bestimmten Bereich des Softwaresystems gewährleistet. Forschungsuntersuchungen, die sich mit Mining Techniken auf Software beziehen sind vielfältig [3].

2.2 Change Coupling

Die Forschung der Softwareentwicklung wird dadurch verwirklicht, indem man durch die Anwendung von Software-Historien die aktuellen Probleme des Softwaresystems analysiert, um die Ursachen dieses Problems zu verstehen und die zukünftige Entwicklung des Softwaresystems vorherzusagen [1].

Es existieren neben vielen Informationen auch Informationen über Change Coupling in den Software-Historien eines Softwaresystems. Unter Change Coupling versteht man die gemeinsame Änderung von zwei oder mehreren Software-Artefakten während der

Entwicklung eines Softwaresystems. D.h., Change Coupling zeigt die gekoppelten Dateien, die zusammen geändert wurden, und besagt, dass die in der Software-Historie zusammen geänderten Dateien auch zukünftig zusammen geändert werden müssen. Durch diese Informationen von den Change Coupling können die Entwickler ihre Bugfixings- und Modifikationsaufgaben viel besser und effizienter bewerkstelligen. In der vorliegenden Arbeit wird anstelle des Begriffes „Change Coupling“ öfters auch die deutsche Übersetzung „gekoppelte Dateiänderungen“ benutzt [4].

2.3 Data Mining

Viele Unternehmen haben durch den technischen Forstschritt die Möglichkeit bekommen jede Art von Information, die mit Ihren Unternehmensaktivitäten in Verbindung stehen zu sehr vernünftigen Preisen zu speichern. Dieser Fortschritt bringt aber auch den Nachteil mit sich, dass die Lücke zwischen der Datenerstellung und dem Datenverständnis immer grösser wird, weshalb die Verarbeitung und Interpretation dieser gespeicherten Daten einen sehr hohen Stellenwert in einem Unternehmen bekommt. Die Verarbeitung und Interpretation der Daten sollen es dem Unternehmen ermöglichen, zu neuen Informationen zu gelangen, um einen wirtschaftlichen Vorteil für das Unternehmen zu schaffen [5]. Diesen Prozess der Datenverarbeitung und Interpretation bezeichnet man als Data Mining. Im Allgemeinen geht es beim Data Mining darum Wissen aus großen Datenmengen zu extrahieren.

Abbildung 1: Data Mining: Suche nach Wissen in den Daten [6]

Definition Data Mining:

> *"Data mining is the process of discovering hidden, previously unknown and usable information from a large amount of data. The data is analyzed without any expectation on the result. Data mining delivers knowledge that can be used for a better understanding of the data."* [9]

Diese Definition macht es deutlich, dass es sich hierbei um einen Prozess handelt, dessen Ziel es ist Wissen zu liefern, um die Daten besser zu verstehen. Diese Wissensgewinnung wird dadurch realisiert, indem bekannte und nützliche Informationen aus großen Datenmengen extrahiert, entdeckt und anschließend durch gewisse Verfahren analysiert werden.

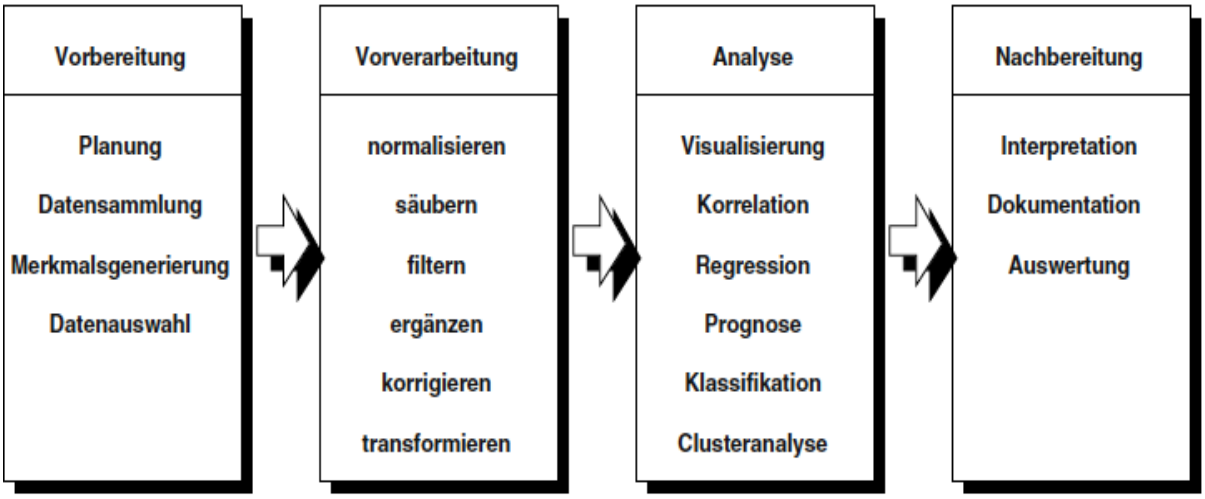

Abbildung 2: Der Data-Mining-Prozess [7]

In der Abbildung 2 wird noch einmal gut veranschaulicht, dass das Data Mining die Extraktion des Wissens, also von interessanten Mustern, aus Datenmengen zu Ziel hat. Bei interessanten Mustern hingegen handelt es sich um Mustern, welche sich durch die Eigenschaften Nichttrivialität, Nützlichkeit, Verständlichkeit und allgemeine Gültigkeit beschreiben lassen. Die Abbildung 2 stellt somit einen Überblick über die einzelnen Phasen, die beim Data-Mining-Prozess nicht zwingend sequentiell durchlaufen werden. Der Prozess besteht aus den Phasen Vorbereitung, Vorverarbeitung, Analyse und Nachbereitung. Ausgangspunkt sind die Rohdaten. Das bedeutet, bevor der Data-Mining-Prozess beginnt, liegen im Regelfall die Rohdaten ungeordnet vor. Zudem sind diese Rohdaten dadurch ausgezeichnet, dass sie unvollständig, teilweise redundant, unwichtig und fehlerhaft sind. Deshalb ist die Notwendigkeit der Vorbereitung und Vorverarbeitung dieser Rohdaten erforderlich, um die Analyse durchzuführen.

In der Vorbereitungsphase ereignet sich somit in erster Linie die Planung und Datensammlung. Da im Regelfall beim Projektanfang erst einmal überhaupt keine Daten vorhanden sind, erfolgen im Vorfeld die Planung und Durchführung der Datensammlung. Im Anschluss daran findet noch in der Vorbereitungsphase die Merkmalsegmentierung und die Datenauswahl statt, bevor diese Rohdaten an die Vorverarbeitungsphase zur Normalisierung, Säuberung, Ergänzung, Korrektur, Filterung und Transformation weitergeleitet werden. Nachdem die Vorverarbeitungsphase auch sein Ende erreicht hat, stehen dann die Daten bereit für die Analyse. Anschließend werden in der Analysephase die vorverarbeiteten Daten mit Hilfe von verschiedenen Methoden analysiert. Welche Methoden angewendet werden sollen, sind davon abhängig, was für ein Wissen man aus diesen Daten herausziehen möchte. Diese verschiedenen Verfahren werden in dem Kapitel 2.4. näher in Betracht gezogen. Aus dieser Analysephase erhält man einige Ergebnisse, die wiederum durch die Nachbereitungsphase herangezogen werden, um das Wissen aus diesen Ergebnissen zu extrahieren. Nach dieser letzten Phase steht das Wissen schließlich für die Verwendung zur Verfügung [7]. Anhand von diesem Blickwinkel kann man dann dem Data Mining System die in der nachfolgenden Abbildung 3 dargestellten Komponenten zuordnen.

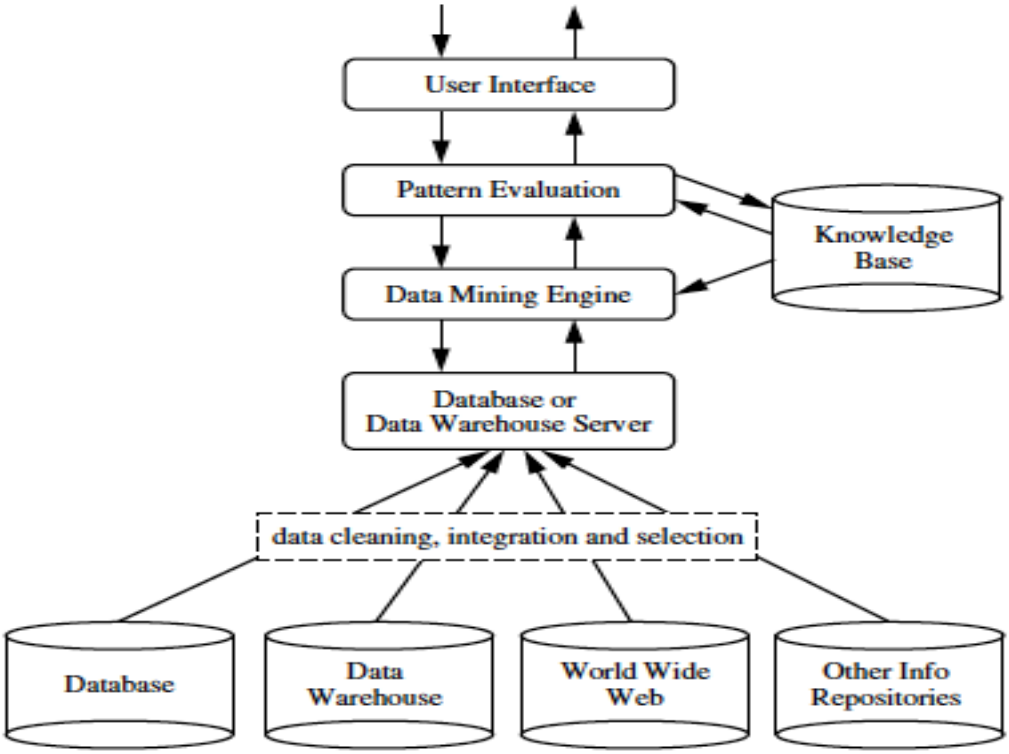

Abbildung 3: Architektur eines Data Mining Systems [6]

So wie in der Abbildung 3 auch zusehen, besteht ein Data Mining Systems aus insgesamt fünf Komponenten, die im Folgenden näher erläutert werden:

1. **Datenbank, World Wide Web, andere Info Repositories, Dara Warehouse:**
   Diese Komponente stellt eine Menge an Datenbanken dar, deren Inhalt später im Data-Mining-Prozess zur Bearbeitung herangezogen wird. D.h., dass die Datenquellen sich in dieser Komponente befinden.

2. **Datenbank oder Data Warehouse Server:**
   Für die Aufnahme der durch den Benutzer abgefragten, relevanten und vorverarbeiteten Daten ist der Datenbank Server oder der Data Warehouse Server verantwortlich.

3. **Data Mining Engine:**
   Diese Komponente bildet sozusagen das Herz eines Data Mining Systems. Hier finden die verschiedenen Verfahren der Data Mining Techniken Anwendung. Die Entscheidung, welche Methode bei welcher Aufgabe anzuwenden ist, ist von den durch die Anwender definierten Aufgabenstellungen abhängig.

4. **Wissensdatenbank (Knowledge Base):**
   Bei dieser Komponente handelt es sich um eine Wissensdatenbank, die eine Orientierung bei der Suche oder der Auswertung nach resultierenden Mustern dient. Darüberhinaus ist eine Wissensdatenbank auch dadurch ausgezeichnet, dass das Wissen in dieser Datenbank in schriftlicher Form vorliegt.

5. **Musterauswertung (Pattern Evaluation):**
   Die Hauptaufgabe von dieser Komponente ist es, mit den Data Mining Modulen zu interagieren und somit den Fokus der Musterauswertung auf interessante Muster zu richten. Die Muster haben große Bedeutung im Data Mining. Das bedeutet, dass die uninteressanten Muster durch einen bestimmten Wert ausgefiltert werden können.

6. **Benutzerschnittstelle (User Interface):**
   Zu guter Letzt existiert auch die User Interface Komponente, welche die Schnittstelle zwischen dem Benutzer und dem Data Mining System darstellt. Durch diese Schnittstelle hat dann der Benutzer die Möglichkeit Anfragen an das Data Mining Systems zu senden, um die relevanten Informationen zu selektieren und sich am Ende auch die Ergebnisse anzuschauen. Allgemein formuliert ermöglicht der User Interface die Interaktion des Benutzers mit dem Data Mining System [6].

2.4 Data Mining Techniken

Dieser Abschnitt beschäftigt sich mit der detaillierten Erläuterung der Data Mining Techniken, wobei das Hauptaugenmerk auf die Assoziationsanalyse und in diesem Zusammenhang auf den FP-Growth Algorithmus gerichtet ist.

2.4.1 Überblick über die Data Mining Techniken

Es existieren vier Data Mining Techniken, die generell akzeptiert werden. Diese sind Assoziationsanalyse, Clusteranalyse, Klassifikationsanalyse und Regressionsanalyse. Bei diesen Techniken handelt es sich wiederum entweder um deskriptive oder prädiktive Techniken. Clusteranalyse und Assoziationsanalyse gehören zu den deskriptiven Techniken, während Klassifikationsanalyse und Regressionsanalyse zu den prädiktiven Techniken zugeordnet werden.

Prädiktive Techniken:

1. **Klassifikationsanalyse:**
   Das Ziel dieses Verfahrens ist die Ermittlung von bestimmten Mustern, die es ermöglichen sollen, mit Hilfe von bereits vorhandenen Informationen, Aussagen über Objekte machen zu können.

   Nach der Festlegung der Kriterien, werden die einzelnen Objekte entsprechend dieser Kriterien klassifiziert. Daraus wird ein Klassifikationsmodell aus diesen Objekten generiert.

   Dieses Verfahren wird in diversen Bereichen eingesetzt. Eines der Anwendungsgebiete von diesem Verfahren ist die Beurteilung der Kreditwürdigkeit der Kunden [8]. In Abbildung 4 ist der Entscheidungsbaum dargestellt, der die Kreditwürdigkeit der Kunden einer Bank repräsentiert. Von diesem Entscheidungsbaum kann man entnehmen, dass wenn ein Kunde nicht berufstätig ist, kein Vermögen hat, Student ist und keine Bürgschaft der Eltern besitzt, folglich auch keinen Anspruch auf Kredit bekommt [8].

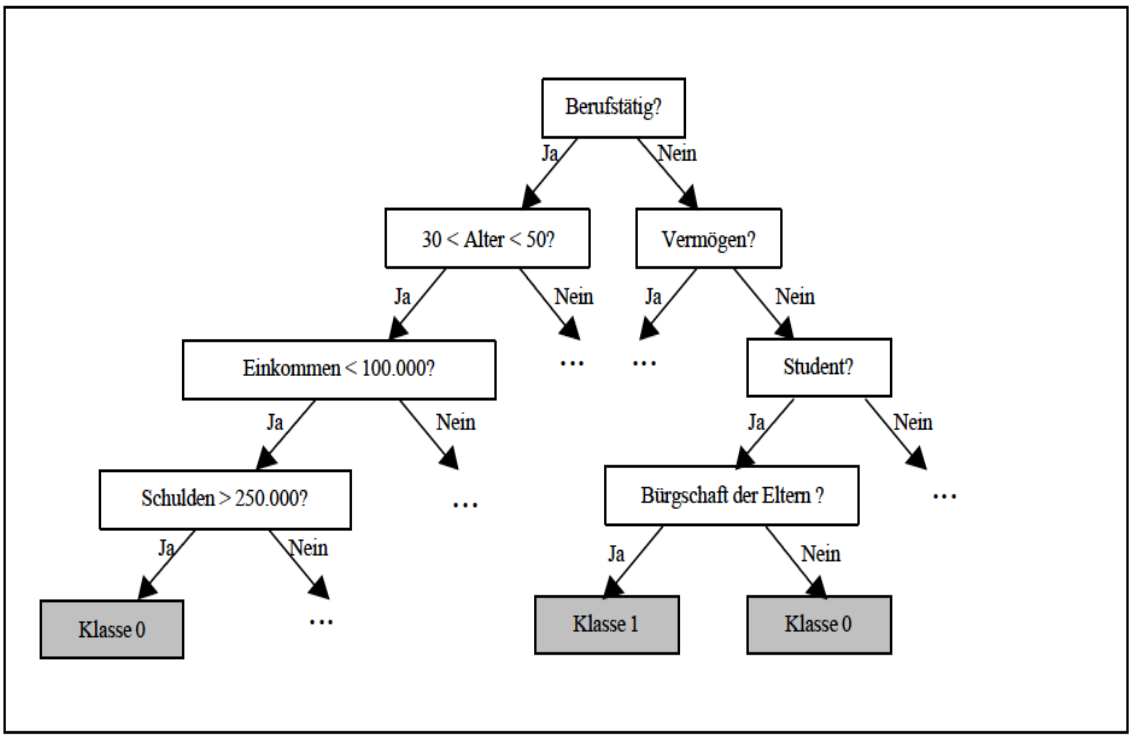

Abbildung 4: Klassifikationsanalyse: Entscheidungsbaum für Kreditwürdigkeit [8]

2. **Regressionsanalyse:**
   Regressionsanalyse weist sehr viele Ähnlichkeiten zu Klassifikationsanalyse auf, mit dem Unterschied des Zielfeldes. Während das Ziel der Klassifikationsanalyse die Vorhersage von Klassenlabels ist, geht es bei der Regressionsanalyse um die Vorhersage von numerischen Werten, weshalb dieser Wert keine Gleichheit mit anderen Werten in dem Modell aufzeigt [9]. Abbildung 5 repräsentiert die Regressionsanalyse.

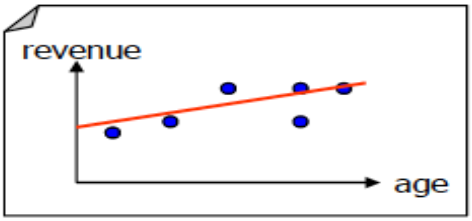

Abbildung 5: Regressionsanalyse [9]

<u>Deskriptive Techniken:</u>

1. **Clusteranalyse:**
   Unter Clusteranalyse versteht man die Unterteilung von einer gewissen Anzahl an Objekten in Gruppen mit Hilfe von gewissen Variablen. Die Gruppen hingegen sind

so aufgegliedert, dass alle Objekte innerhalb der gleichen Gruppe homogene und alle in unterschiedlichen Gruppen heterogene Charaktermerkmale aufweisen. Anwendung findet dieses Verfahren vor allem im Marketing und speziell im Bereich der Marktforschung. Dieser Vorgang der Marktunterteilung wird auch als Marktsegmentierung bezeichnet [10].

Abbildung 6 zeigt wie die Clusteranalyse abläuft. Dieses Beispiel beschreibt wie die Bankkunden anhand von ihren Einkommen und ihren Schulden in unterschiedliche Clustern untergliedert werden. Cluster 1 repräsentiert Bankkunden mit geringen Einkommen und hohen Schulden. Cluster 2 zeigt die Bankkunden mit wenig Einkommen und wenig Schulden. Zuletzt befinden sich in Cluster 3 diejenigen Bankkunden, deren Einkommen über ihre Schulden sind.

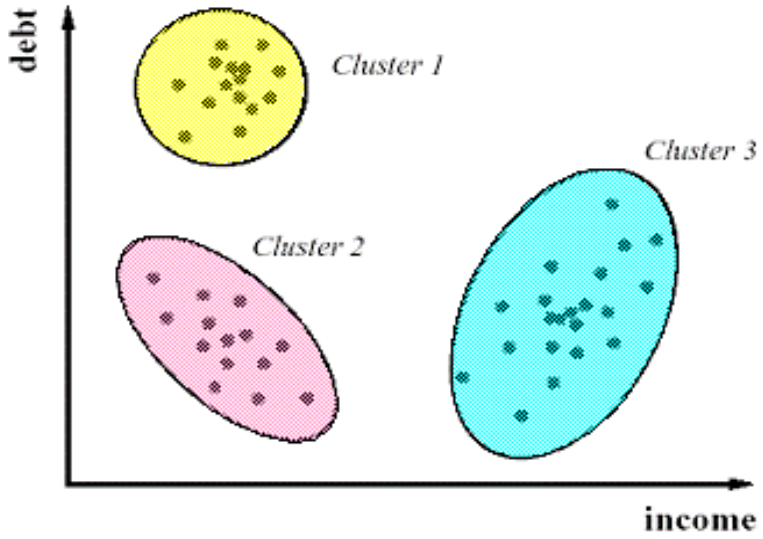

Abbildung 6: Beispiel Clusteranalyse [11]

2. **Assoziationsanalyse:**
   Die Basis der Assoziationsanalyse bildet eine Transaktionsdatenbank. Diese Datenbank besteht aus mehreren Transaktionen. Eine Transaktion hingegen besteht aus einer Menge von Items.

   Das Ziel der Assoziationsanalyse ist es, die Zusammenhänge und die Beziehungen zwischen den Items zu bestimmen und in den Vordergrund zu bringen. Das Ergebnis der Assoziationsanalyse sind Assoziationsregeln. Assoziationsregeln sind Regeln der Form wie beispielweise: Kunden die das Produkt A gekauft haben, haben auch zu 80% das Produkt B und zu 50% das Produkt C gekauft. Dieses Verfahren ermöglicht es den Unternehmen das Kaufverhalten ihrer Kunden zu analysieren und ihr Sortiment diesem Verhalten entsprechend anzupassen [12].

Auf Transaktionsebene betrachtet sind die Assoziationsregeln folgendermaßen zu formulieren: Eine Transaktion, welche das Item A besitzt, enthält auch zu 80% das Item B und zu 50% das Item C. Folglich ist die Hauptaufgabe der Assoziationsanalyse die Entdeckung von Assoziationsregeln in den Transaktionsdatenbanken.

Die Bestimmung von Assoziationsregeln erfolgt in zwei Schritten. Im ersten Schritt findet die Generierung von Frequent Itemsets mit Hilfe eines Data Mining Algorithmus statt. Das bedeutet, es werden die häufig auftretenden Itemmengen ermittelt. Der zweite Schritt erzeugt im Anschluss daran die Assoziationsregeln aus den ermittelten Frequent Itemsets [9]. Die Abbildung 7 zeigt ein Beispiel der Assoziationsanalyse.

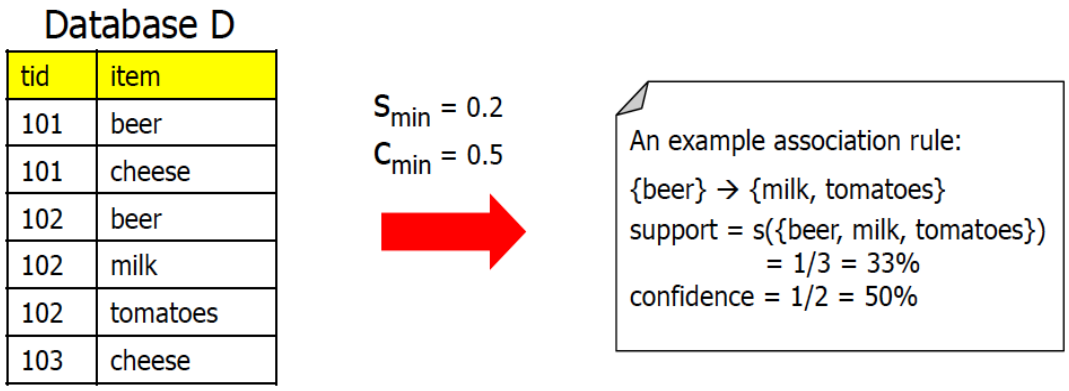

Abbildung 7: Assoziationsanalyse [9]

2.4.2 Ermittlung von Frequent Itemsets in der Assoziationsanalyse

Wie in dem vorherigen Abschnitt auch erläutert, besteht die Assoziationsanalyse aus zwei Schritten. Der erste Schritt ist die Ermittlung von den Frequent Itemsets mit Hilfe eines Data Mining Algorithmus. Der zweite Schritt ist die Erzeugung der Assoziationsregeln aus diesen Frequent Itemsets.

Da diese Diplomarbeit aber nur den ersten Schritt der Assoziationsanalyse umfasst, wird hier nur auf diesen einen Teil eingegangen. Für die Ermittlung von den Assoziationsregeln aus den Frequent Itemsets ist das Vorlesungsskript vom PD Dr. rer. nat. habil. Holger Schwarz zu empfehlen [9].

1. Formale Beschreibung:

Transaktionsdatenbank:

Eine Transaktionsdatenbank D setzt sich zusammen aus einer bestimmten Transaktionsaktionsmenge $D = \{T_1, T_2, T_3, \ldots, T_m\}$, wobei jede Transaktion durch ein eindeutiges TID (Transaktionsidentifikation) gekennzeichnet ist. Zudem existieren Items gegeben durch die Menge $I = \{I_1, I_2, I_3, \ldots I_n\}$. Weiterhin gilt für jede Transaktion, dass sie eine Menge von Items beinhaltet und dass sie Teilmenge von der Transaktionsdatenbank D ist, also $T \subseteq D$ gilt. Es ist zusätzlich auch noch eine Menge A definiert, welche daraufhin überprüft wird, ob sie Teilmenge von T ist, also $A \subseteq T$ gilt [13].

Support:

Support oder auch Unterstützungsgrad genannt, gibt einen Wert über die Anzahl der Transaktionen in der Menge A an. Die Formel für die Berechnung von dem Supportwert für die Menge A lautet folgendermaßen [14]:

$$support(A) = \frac{|\{T \in D | A \subseteq T)|}{|D|}$$

Es existiert weiterhin ein Minimumsupportwert s_{min}, wobei $0 \leq s_{min} \leq 1$ gilt. Einen Itemset A nennt man Frequent Itemset, wenn $support(A) \geq s_{min}$ gilt. Unter Frequent Itemset versteht man häufig auftretende Itemmengen. Die Assoziationsregeln hingegen zeichnen sich dadurch aus, dass sie Implikationen aufweisen. Betrachtet man die folgende Assoziationsregel: $A \rightarrow B$. Diese Regel besagt somit: Wenn A ein Teil der Transaktion T ist, dann ist auch B ein Teil der Transaktion T [9]. Die Formel zum Support zu der Assoziationsregeln sieht somit wie folgt aus [13]:

$$support(A \rightarrow B) = support(A \cup B) = \frac{|\{T \in D | A \cup B \subseteq T\}|}{|D|}$$

Konfidenz

Die Konfidenz gibt die relative Häufigkeit der Beispiele an, für die die Assoziationsregel richtig ist. Die Formel für die Berechnung des Konfidenzwertes für die Regel $A \rightarrow B$ sieht wie folgt aus:

$$conf_D(A \rightarrow B) = \frac{|\{T \in D | A \cup B \subseteq T\}|}{|\{T \in D | A \subseteq T\}|}$$

Es existiert weiterhin auch ein Minimumkonfidenzwert c_{min}. Eine Assoziationsregel wird als streng bezeichnet, wenn deren Supportwert und Konfidenzwert größer oder gleich dem Minimumsupportwert und Minimumkonfidenzwert ist [15].

2. Ermittlung von Frequent Itemsets am Beispiel des FP-Growth Algorithmus:

Der FP-Growth Algorithmus ist eine baumbasierte Verarbeitung der Daten von einer Transaktionsdatenbank. Der Unterschied von diesem Algorithmus im Vergleich zum Apriori Algorithmus ist dies, dass er ohne Kandidatengenerierung die Frequent Itemsets bestimmt. [16].

Beim FP-Growth Algorithmus findet die Ermittlung der Frequent Itemsets in zwei Schritten statt. Im ersten Schritt wird der FP-Tree erzeugt. Beim FP-Tree handelt es sich um einen Baum, der eine kompakte Datenstruktur der Transaktionsdatenbank repräsentiert. Um den FP-Tree zu erzeugen, muss die Transaktionsdatenbank zweimal gescannt bzw. durchlaufen werden. Beim ersten Durchlauf werden die Supportwerte der einzelnen Items von der Transaktionsdatenbank ermittelt. D.h., es wird berechnet, in vielen Transaktionen die einzelnen Items vorkommen [17]. Die Tabelle 1 stellt eine Transaktionsaktionsdatenbank D dar, die für die Erzeugung vom FP-Tree zweimal durchlaufen bzw. gescannt wird.

Tabelle 1: Transaktionsdatenbank D [18]

TID	Itemsets
101	{Apfel, Mandarine, Melone}
102	{Apfel, Mandarine, Zitrone}
103	{Apfel, Birne}
104	{Mandarine, Melone, Apfel}
105	{Apfel, Mandarine, Melone, Zitrone}
106	{Birne}

Nachdem die Tabelle 1 einmal gescannt wurde, werden die Ergebnisse, also die Supportwerte von jedem Item, in eine Zwischenergebnisstabelle gespeichert. Die Tabelle 2 repräsentiert die Zwischenergebnisstabelle für die Items. Beispielweise besagt die erste Zeile der Tabelle 2, dass das Item {Apfel} einen Supportwert von 5 hat. D.h., dass das Item {Apfel} in 5 Transaktionen vorkommt. Dies sind in diesem Fall die Transaktionen mit den TIDs {101, 102, 103, 104, 105} [18].

Tabelle 2: Zwischenergebnisstabelle [18]

Item	Support
Apfel	5
Mandarine	4
Melone	3
Zitrone	2
Birne	2

Ist der erste Durchlauf der Transaktionsdatenbank D der Tabelle 1 vollendet, folgt der zweite Durchlauf, um den FP-Tree zu konstruieren. Dazu werden die Transaktionen der Tabelle 1 noch einmal durchlaufen und überprüft, ob die Supportwerte der einzelnen Items größer oder gleich dem Minimumsupportwert sind. Falls $support(Item) \geq s_{min}$ gilt, wird das Item bei der FP-Tree Konstruktion berücksichtigt, andernfalls nicht [19].

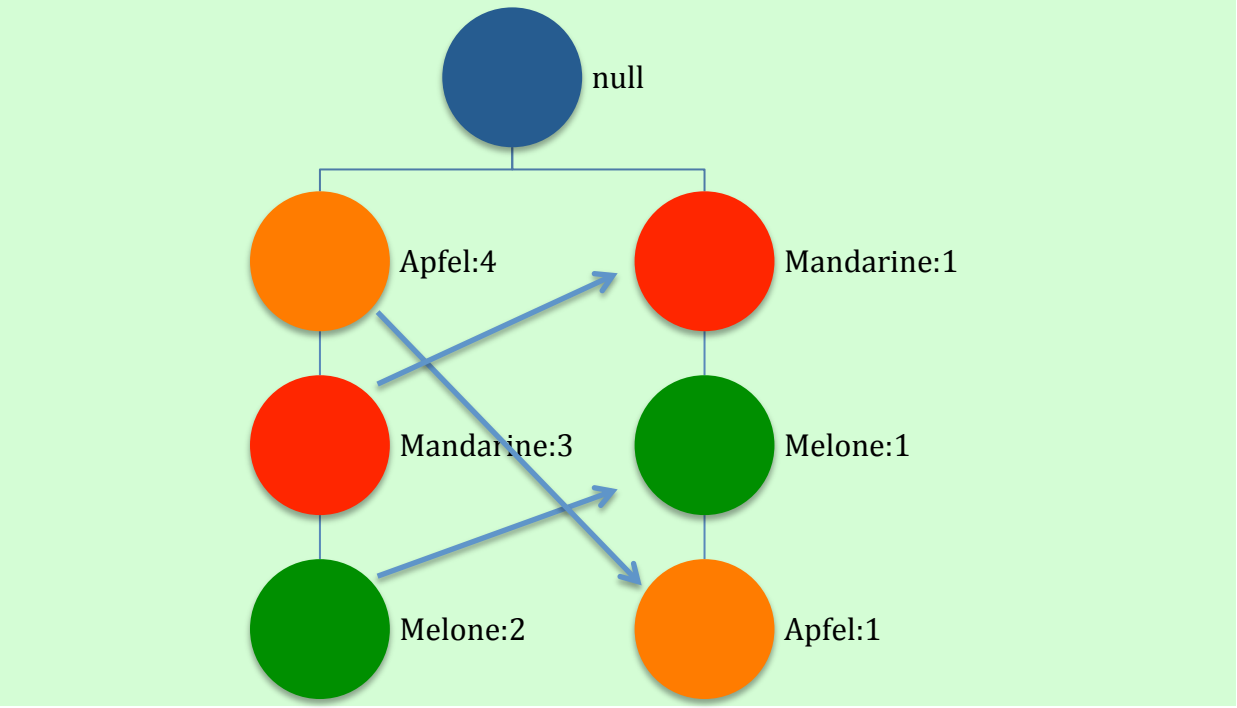

Abbildung 8: FP-Tree

Die Abbildung 8 stellt den FP-Tree dar, der nach dem zweiten Durchlauf der Transaktionsdatenbank D aus der Tabelle 1 und unter Anwendung von einem Minimumsupportwert von 50% ($s_{min} = 50\% = 3$) entstanden ist. Man kann von der Abbildung 8 entnehmen, dass die Items {Zitrone} und {Birne} nicht bei der FP-Tree Konstruktion beachtet wurden, da deren Supportwerte kleiner als dem Minimumsupportwert sind. Somit wird der erste Schritt abgeschlossen [19].

Im zweiten Schritt wird der FP-Tree von den Blättern bis zu dem Wurzel nach dem Bottom-up Prinzip bearbeitet, um somit die Frequent Itemsets zu ermitteln [19]. Der Vorgang erfolgt nach dem Divide and Conquer Verfahren. Zunächst werden die Präfixpfade aus dem FP-Tree erzeugt. Präfixpfade sind Unterbäume (*eng. sub-trees*), die mit einem Item oder Itemset enden. Die Erzeugung der Präfixpfade werden anhand der Links (Pfeile, die in der Abbildung 8 ersichtlich sind) verwirklicht. Die Abbildung 9 zeigt als Beispiel den Präfixpfad mit dem Item {Melone} als Endung. Es werden zwei weitere Präfixpfade erstellt. Einer von denen endet mit dem Item {Mandarine} und der andere mit dem Item {Apfel}. Nachdem die einzelnen Präfixpfade von dem FP-Tree extrahiert sind, beginnt die Ermittlung der Frequent Itemsets von diesen Präfixpfaden [17]. Es wird hier aber nur die Verarbeitung von dem Präfixpfad, der mit dem Item {Melone} endet näher beschrieben. Die Bearbeitung von den anderen Präfixpfaden erfolgt aber nach dem gleichen Prinzip.

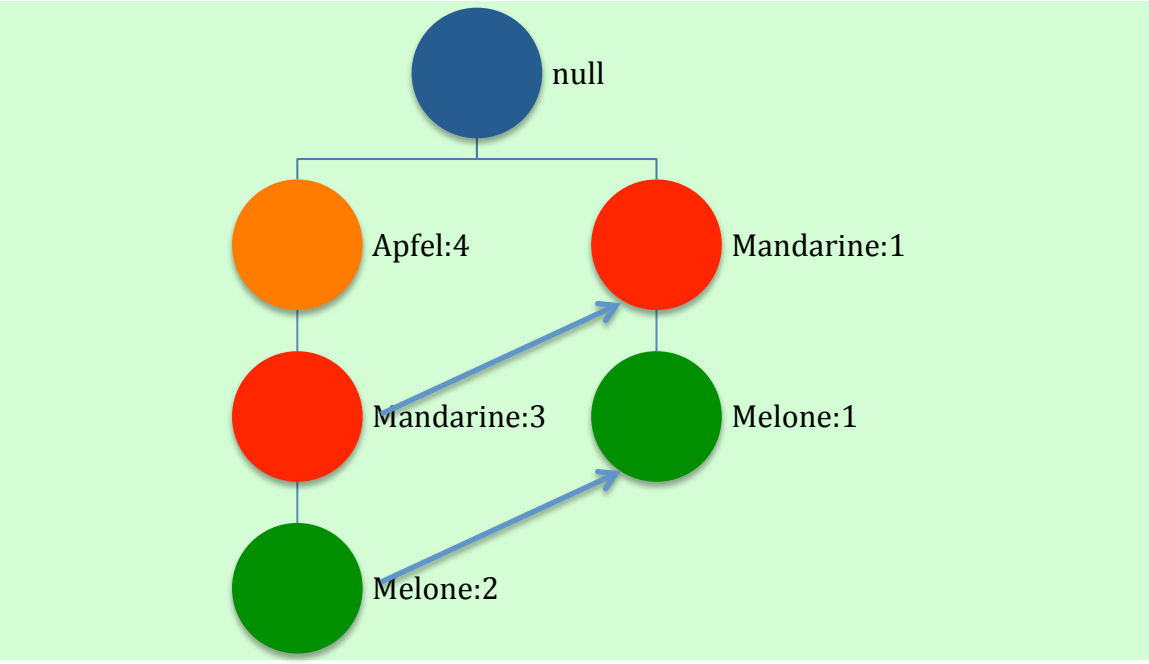

Abbildung 9: Präfixpfad mit der Endung Melone

Die Ermittlung von den Frequent Itemsets erfolgt mit Hilfe der Links, indem alle Supportwerte von einem Item entlang eines Pfades (der Pfad wird durch die Links angegeben) addiert und anschließend mit dem gegebenen Minimumsupportwert verglichen werden. Ist der Supportwert von dem Item oder Itemset größer oder gleich dem Minimumsupportwert, dann handelt es sich bei dem Item(set) um einen Frequent Item(set), andernfalls nicht.

In diesem Beispiel wird von einem Minimumsupportwert von 3 ($s_{min} = 50\% = 3$) ausgegangen. Bei der Ermittlung von den Frequent Itemsets wird anfänglich das letzte Item betrachtet und dessen Supportwert ermittelt. In diesem Fall gilt $support(Melone) = 3 \geq s_{min} = 3$. Dies besagt, dass das Item {Melone} ein Frequent Itemset ist [19].

Nachdem das Item {Melone} als Frequent Itemset ermittelt wurde, wird weiterhin überprüft, ob es sich bei den Itemsets, welche mit dem Item {Melone} enden, ebenfalls um Frequent Itemsets handelt. Folglich wird überprüft, ob die Itemsets {Mandarine, Melone}, {Apfel, Melone}, {Apfel, Mandarine, Melone} auch Frequent Itemsets sind. Um diese Überprüfung durchzuführen wird aus dem Präfixpfad der Conditional FP-Tree ermittelt. Conditional FP-Tree zeichnet sich dadurch aus, dass es das letzte Item nicht beinhaltet und die Supportwerte von den restlichen Items aktualisiert sind.

Der Vorgang zur Ermittlung des Conditional FP-Trees sieht folgendermaßen aus:

1. Die Transaktionsdatenbank D wird noch einmal durchgegangen, und nur diejenigen Transaktionen werden in Betracht gezogen, welche das Item {Melone} am Ende beinhalten.

2. Infolgedessen werden die Supportwerte von den Items in dem entsprechenden Präfixpfad aktualisiert.

3. Zum Schluss wird dann das letzte Item von dem Präfixpfad entfernt. Somit liegt das Conditional FP-Tree bereit.

Diese beschriebenen drei Schritte werden solange wiederholt bis kein Item mehr zu entfernen ist. Das heißt, bis alle möglichen Frequent Itemsets aus dem entsprechenden Präfixpfad ermittelt sind. Abbildung 10 zeigt das entstandene Conditional FP-Tree für das Item {Melone} [17].

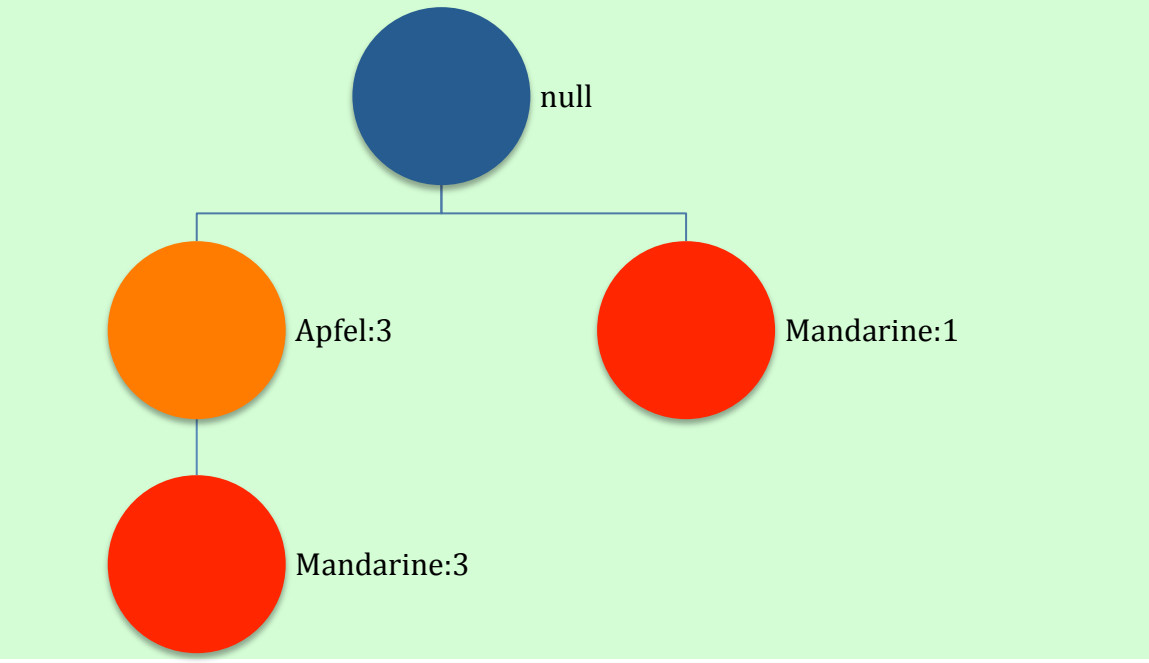

Abbildung 10: Conditional FP-Tree für das Item Melone

Somit werden alle Frequent Itemsets ermittelt. Die Tabelle 3 präsentiert die ganzen Frequent Itemsets, die nach dem FP-Growth Algorithmus ermittelt wurden.

Tabelle 3: Tabelle der Frequent Itemsets

Suffix	**Frequent Itemsets**
Melone	{Melone}, {Apfel, Melone}, {Mandarine, Melone}, {Apfel, Mandarine, Melone}
Mandarine	{Mandarine}, {Apfel, Mandarine}
Apfel	{Apfel}

2.5 Eclipse Plattform

Wie bereits im Kapitel 1 erwähnt, ist das Ziel dieser Diplomarbeit die Entwicklung eines Eclipse Plug-Ins. Aus diesem Grund werden im Folgenden auf die Grundlagen des Eclipse Plattforms eingegangen.

2.5.1 Was ist Eclipse?

Die Eclipse Entwicklungsumgebung ist ein Programmierwerkzeug, der sich dadurch auszeichnet, dass es java-basiert, erweiterbar und quelloffen ist [20]. Eclipse wurde vom IBM im Jahre 1998 entwickelt mit dem Ziel jede Art der Softwareentwicklung zu unterstützen. Ursprünglich wurde Eclipse für die Sprache Java entwickelt, wobei aber im Laufe der Zeit die Eclipse Entwicklungsumgebung auch für die anderen Sprachen erweitert wurde. Darüberhinaus ist die Eclipse Architektur auch so konstruiert, dass es auch den Drittanbietern die Möglichkeit gibt, die Eclipse Entwicklungsumgebung zu erweitern, um die Integration von externen Tools ins Eclipse Plattform zu gewährleisten [21]. Die Erweiterung der Eclipse Entwicklungsumgebung wird durch die Plug-In Entwicklung realisiert, wofür in Eclipse die Plug-In Development Environment (PDE) vorgesehen ist [20].

2.5.2 Eclipse Plattform Übersicht

Die Eclipse Architektur basiert hauptsächlich auf die Einbettung von verschiedenen Plug-Ins in die Laufzeitumgebung. Die Abbildung 11 zeigt den Überblick über den Eclipse Plattform. In diesem sind außer der Laufzeitumgebung (Runtime Environment) alle anderen Komponenten Plug-Ins. Die Laufzeitumgebung ist dafür verantwortlich, alle notwendigen Plug-Ins beim Start von Eclipse zu laden [22].

Wie es auch in der Abbildung 11 ersichtlich ist, besteht die Eclipse Plattform aus den Plug-Ins Workspace Plug-In, Workbench UI Plug-In, Help Plug-In und dem Team Plug-In. Im Folgenden werden nur die ersten zwei Plug-Ins kurz oberflächlich dargelegt, da sie die Hauptkomponenten des Eclipse Plattforms bilden.

- **Workspace:**
  Dieses Plug-In ist für Verwaltung von Ressourcen verantwortlich, weshalb man es auch als Ressource Management Plug-In bezeichnet. Um die Verwaltung von Ressourcen zu verwirklichen, definiert es ein Ressourcenmodell.

- **Workbench UI:**
  Dieses Worbench UI ist für die Darstellung der Workbench Benutzerschnittstelle verantwortlich. Um die Workbench Benutzerschnittstelle so darzustellen, werden mehrere Workbench UI Plug-Ins implementiert. Darüberhinaus ist die Benutzerschnittstelle so konzipiert, dass es individuell durch Drittpersonen auch erweiterbar ist [22]. Ein Workbench UI kann sich aus mehreren Perspektiven zusammensetzen, wobei jede Perspektive wiederum aus Views, Editors, usw. besteht [23].

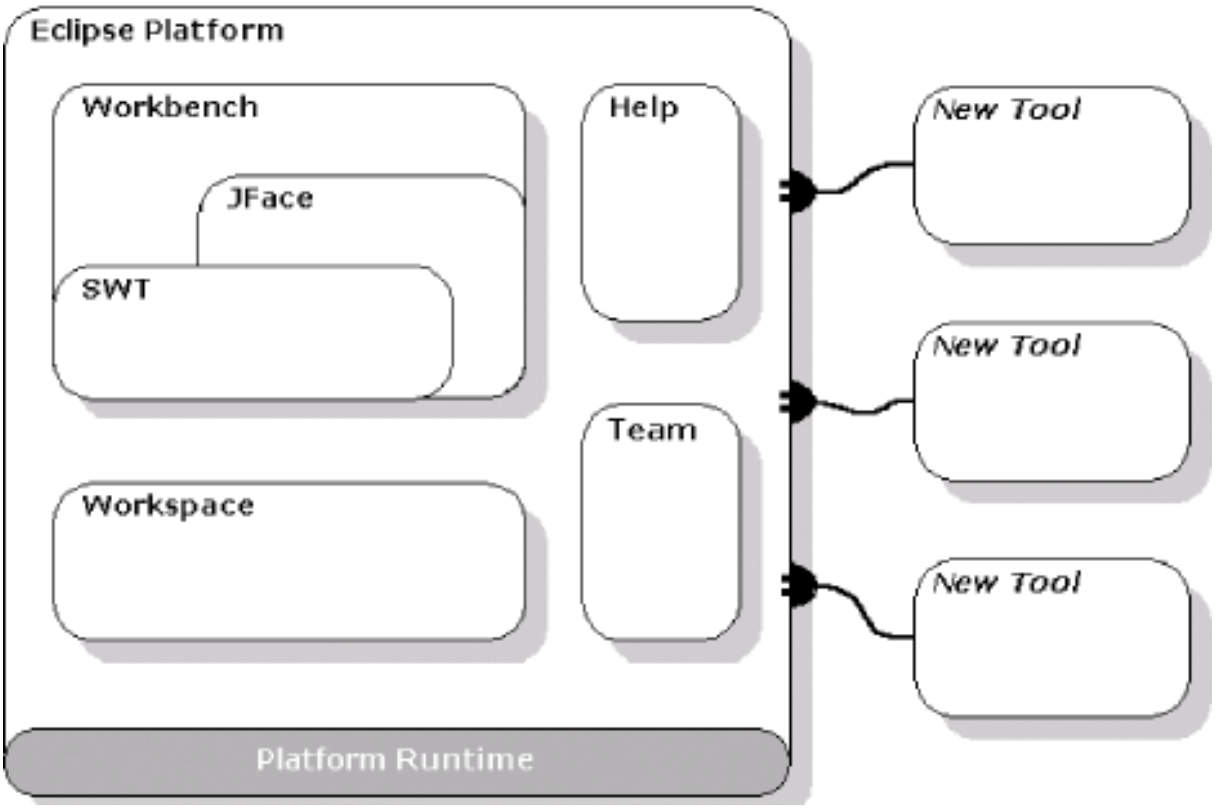

Abbildung 11: Eclipse Plattform Übersicht [23]

2.5.3 Eclipse Plug-In Mechanismus

Bei Plug-Ins handelt es sich um Komponenten, die bei der Erweiterung eines Programmes ihren Einsatz finden [24]. Ein Plug-In setzt sich zusammen aus den Dateien plugin.xml und MANIFEST.MF. Die Datei MANINFEST.MF enthält Informationen über das Plug-In. Das sind Informationen wie zum Beispiel der Name des Plug-Ins, Version, Classpath usw. Die plugin.xml Datei dagegen enthält die Erweiterungen (Extensions) und Erweiterungspunkte (Extension Points) [21].

Die Abbildung 12 repräsentiert die Beziehung zwischen Erweiterungspunkten (Extension Points) und Erweiterungen (Extensions). Wenn beispielweise ein Plug-In, seine eigenen Funktionalitäten erweitern lassen möchte, definiert er in der Regel in seiner plugin.xml Datei einen Erweiterungspunkt, der dann den anderen Plug-Ins erlaubt, diesen Plug-In zu erweitern. Der Erweiterungspunkt ist eine Art Kontrakt. Die anderen Plug-Ins, die das Plug-In erweitern möchten, definieren Erweiterungen. Diese Erweiterungen müssen mit dem Erweiterungspunkt des Plug-Ins übereinstimmen, damit man das Plug-In überhaupt erweitern kann. Es existiert weiterhin auch noch eine Activator Klasse, die für das Starten und Beenden des Plug-Ins zuständig ist [25].

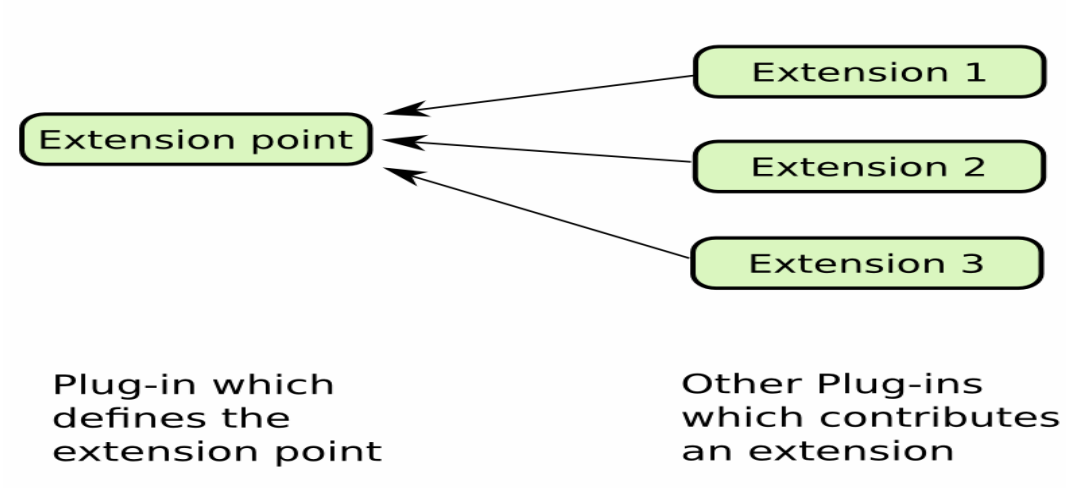

Abbildung 12: Beziehung zwischen Erweiterungspunkt (Extension Point) und Erweiterungen (Extensions) in Eclipse Plug-In [26]

3 Verwandte Projekte

Im Folgenden werden zwei Projekte vorgestellt, die sehr viele Parallelen zu dem im Rahmen dieser Diplomarbeit entwickelten Tools aufweisen.

1. Ansatz von Zimmermann et al: Das ROSE Tool:

Viele die in Amazon.de eine Bestellung gemacht haben, wissen, dass in einem unteren Feld eine Meldung mit der Nachricht erscheint: *"Kunden, die dieses Artikel ... gekauft haben, kauften auch das Artikel .."*. Solche Arten von Informationen werden durch Data Mining erhalten. Wenn jemand beispielweise ein Artikel bestellt, dann überprüft Amazon.de mit Hilfe von Data Mining Techniken, welche anderen Artikeln zusammen mit diesem Artikel bestellt wurden, und schlägt dem Kunden diese Artikel vor.

Ähnlich dem Prinzip von Amazon.de arbeitet auch das ROSE Tool. Hierbei wird Data Mining aber auf Version-Historien angewendet. Wenn der Programmierer eine Änderung vornimmt, dann erscheint ihm eine Meldung *"Programmer who changed ..., also changed ..." (Programmierer, die ... geändert haben, haben auch ... geändert)* [27]. Die Abbildung 13 zeigt die zwei Files *ComparePreferencePage.java* und *plugin.properties* und die Anzahl darüber, wie oft die zwei Files zusammen (gekoppelt) geändert wurden. Aus dieser Abbildung 13 ist ersichtlich, dass die Funktion *fKeys[]* mit der Funktion *initDefaults()* 11 mal und mit dem File *plugin.properties* 10 mal zusammen geändert wurde [28]. Ändert der Programmierer jetzt einen bestimmten File, erscheinen die Vorschläge des ROSE Tools über die anderen Files, die noch geändert werden müssen, in einem anderen Fenster. Abbildung 14 zeigt den Aufbau und die Funktionsweise des ROSE Plug-Ins. Wenn der Programmierer also beispielweise eine Änderung an der Funktion *fKeys[]* in dem Quellcode vornimmt, so schlägt das ROSE Plug-In in dem unteren Fenster alle weiteren Funktionen und Files vor, die gemeinsam mit dieser Funktion *fKeys[]* geändert wurden [27].

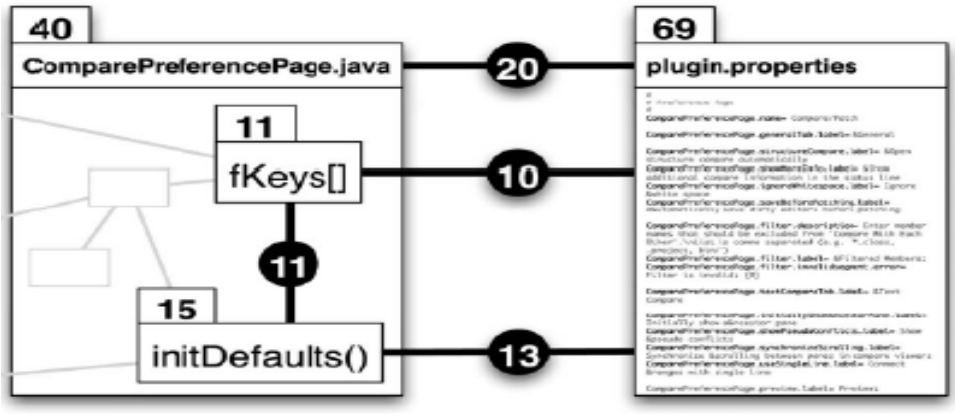

Abbildung 13: Gekoppelte Dateiänderungen zwischen zwei Files [28]

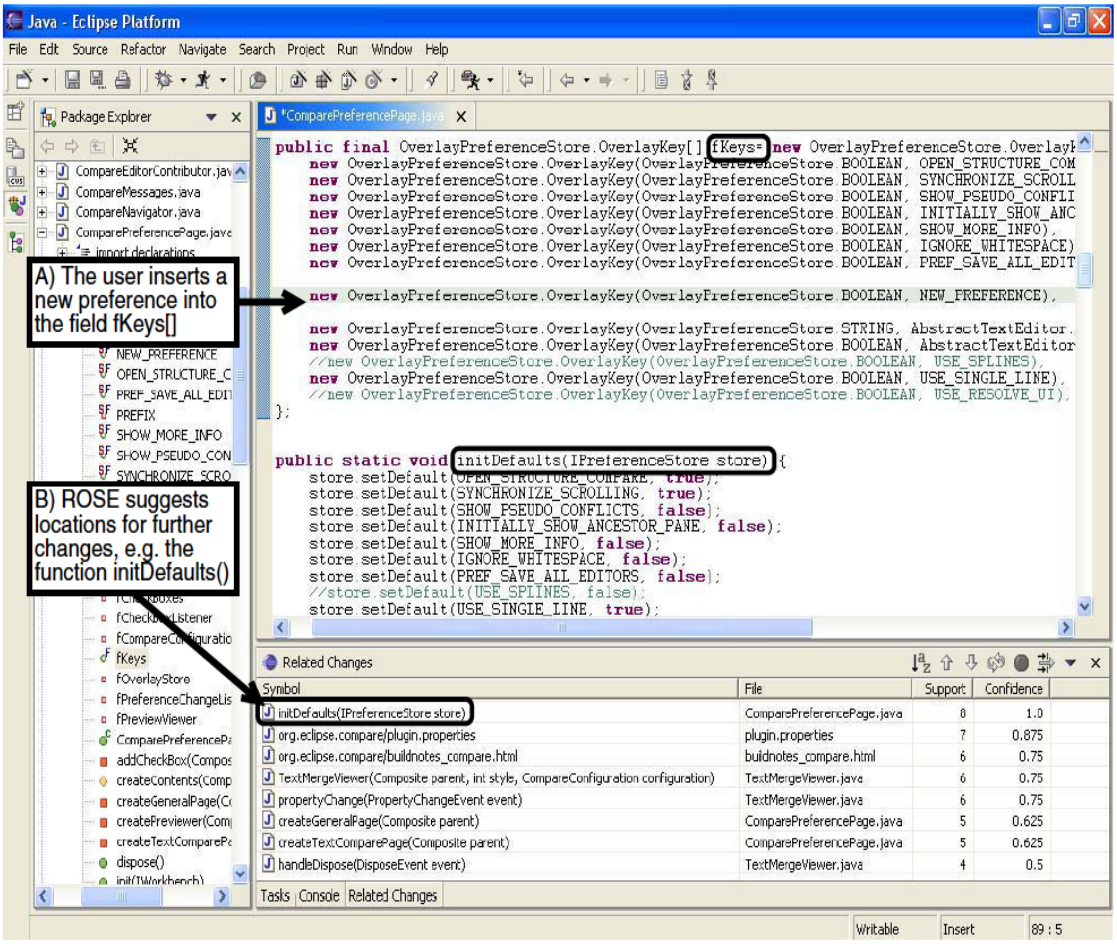

Abbildung 14:ROSE Plug-In [27]

Nachdem die Funktionsweise des ROSE Tools erläutert wurde, wird schließlich auch noch auf die Struktur des Datenflusses in dem ROSE Tool eingegangen. Die Abbildung 15 zeigt den Datenfluss in dem ROSE Tool. Der Datenfluss beginnt mit dem ROSE Server. Der Rose Server liest zu Beginn die Files aus dem Version Archive und gruppiert die Änderungen in Transaktionen. Daraufhin werden auf diese ermittelten Transaktionen Data Mining Verfahren angewendet, um daraus gewisse Menge an Regeln zu generieren. Die Regeln haben bezogen auf das Beispiel in der Abbildung 13 die Form wie: *"Wenn das File ComparePreferencePage.java geändert wird, dann wird das File plugin.properties auch geändert"*. Diese ganzen Regeln werden dann in die Rule Set Datenbank gespeichert.

Die Rose Eclipse Client bildet die Schnittstelle zu dem Benutzer (*eng. User*). Wenn der Benutzer eine Änderung vornimmt, stellt die Rule Application eine Anfrage an den Rule Set, und verlangt alle Regeln, die mit der Anfrage übereinstimmen. Nach dem die Rule Application von dem Rule Set alle übereinstimmenden Regeln erhalten hat, schlägt sie diese Regeln dem Benutzer vor. Wenn der Benutzer also bezogen auf das Beispiel von vorhin das

File *ComparePreferencePage.java* geändert hat, dann schlägt die Rule Application nach der Interaktion mit dem Rule Set dem Benutzer das File *plugin.properties* vor [27].

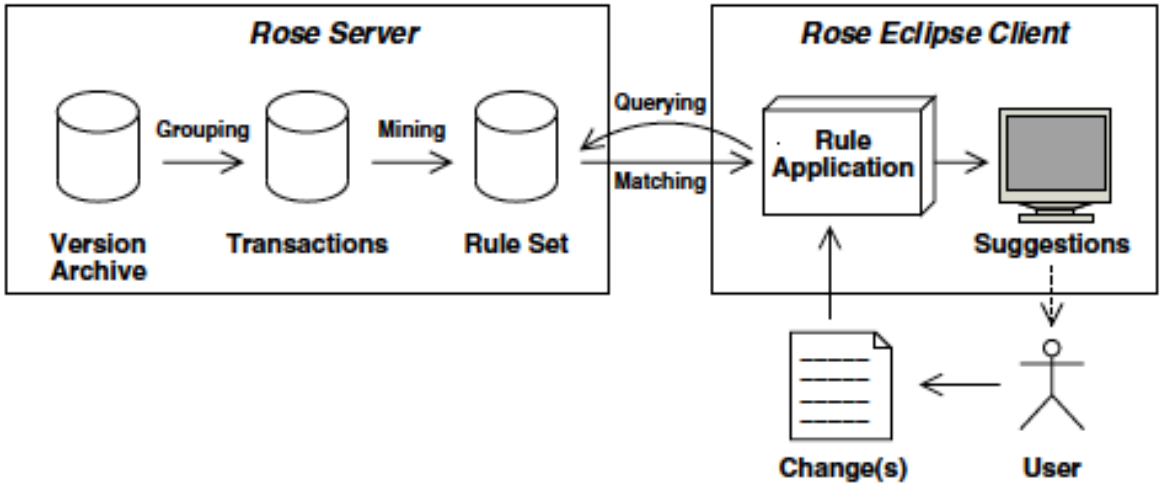

Abbildung 15: Datenfluss in dem ROSE Tool [28]

2. Ansatz von Ying et al.

Dieser Ansatz bezweckt auch die Unterstützung der Entwickler hinsichtlich in ihren Modifikations- und Bugfixingsaufgaben, indem es dem Entwickler alle relevante Quellcodes oder Files zur Änderung vorschlägt.

Dabei liegt der Schwerpunkt dieses Ansatzes in der Ermittlung von Änderungsmustern (Change Patterns) durch die Anwendung von Assoziationsanalysen. Unter Änderungsmustern versteht man jene Files, welche in der Entwicklungshistorie eines Softwaresystems oft zusammen geändert wurden. Aufbauend auf diesen Änderungsmustern (Change Patterns) erfolgen dann die Vorschläge über die relevanten Files. Ändert der Entwickler einen File f_S, so wird durch diesen Ansatz dem Entwickler alle möglichen Files f_R vorgeschlagen, die mit dem File f_S häufig geändert wurden [2].

Dieser Ansatz setzt sich zusammen aus drei Phasen, die in der Abbildung 16 dargestellt sind. In der ersten Phase findet die Extraktion der Daten aus einem Softwarekonfigurationssystem (SCM) statt. Im Anschluss daran erfolgt die Vorverarbeitung dieser extrahierten Daten, um diese dann dem Data Mining Algorithmus als Eingabe zur Verfügung zu stellen.

Die Ermittlung von Änderungsmustern tritt in der zweiten Phase auf. Hierbei wird der Algorithmus der Assoziationsanalyse auf die extrahierten Daten angewendet.

In der letzten Phase werden dem Entwickler alle relevanten Quellcodes vorgeschlagen, indem eine Abfrage auf die Änderungsmustern gemacht wird [29].

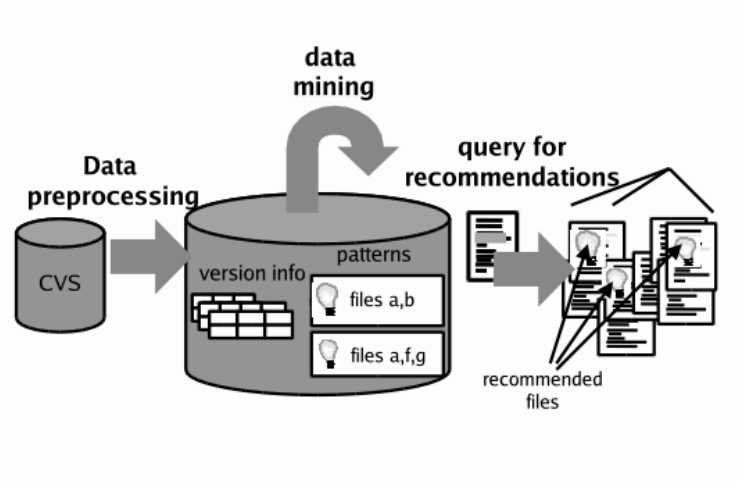

Abbildung 16: Die drei Phasen des Ansatzes von Ying et al [29]

4 Tools und Algorithmen

4.1 Sequential Pattern Mining Framework (SPMF)

Dieser Abschnitt umfasst die Erläuterung des SPMF Data Mining Frameworks und seiner Algorithmen.

4.1.1 Aufbau und Funktionsweise des SPMF

Sequential Pattern Mining Framework oder auch kurz SPMF ist ein Data Mining Framework, dessen Schwerpunkt im Bereich der Frequent Pattern Mining liegt. Es existieren zudem eine große Breite an Data Mining Algorithmen, die sowohl bei den Transaktionsdatenbanken als auch bei den Sequenzdatenbanken zur Ermittlung von Mustern eingesetzt werden. Darüberhinaus ist das SPMF ein java–basiertes und quelloffenes Data Mining Framework. Das SPMF besitzt sowohl einen GUI (siehe Abbildung 17) als auch einen CLI (siehe Abbildung 18) als Schnittstelle mit dem Benutzer [30].

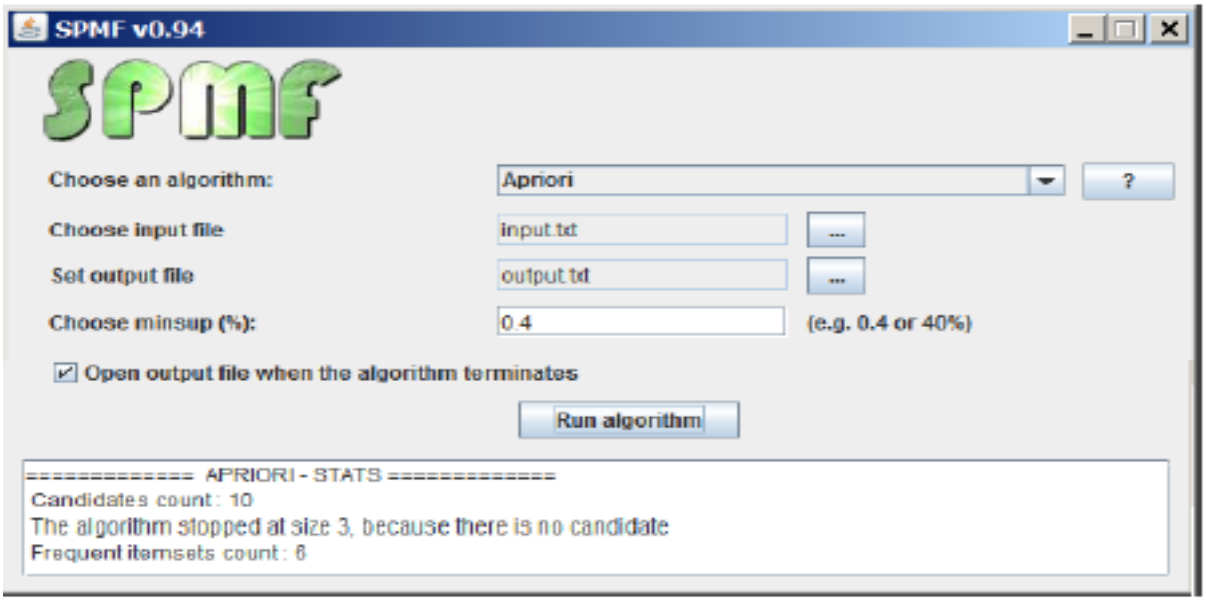

Abbildung 17: SPMF GUI [30]

```
java -jar spmf.jar run Apriori input.txt output.txt 0.4
```

Abbildung 18: SPMF CLI [30]

Die Abbildungen 17 zeigt das Beispiel, wie das Ausführen des Apriori Algorithmus im SPMF GUI aussieht. Der Benutzer selektiert den Apriori Algorithmus in dem Combobox.

Je nachdem, welcher Data Mining Algorithmus selektiert wird, erscheinen oder verschwinden einige Eingabefelder. Die Inputs und Outputs des Sequential Pattern Mining Frameworks sind Text Files. Die Transaktionsdatenbanken und die Sequenzdatenbanken werden den Data Mining Algorithmen als Text Files zur Verfügung gestellt. Der Benutzer selektiert jetzt als Input den Input.txt File und setzt den Namen des Output.txt Files als Output. Weiterhin gibt der Benutzer einen Minimumsupportwert ein und betätigt den „Run algorithm“ Button. Dieser gesamte Ablauf kann auch durch den SPMF CLI (Command Line Interface) ausgeführt werden, wie es in der Abbildung 18 auch zu sehen ist. Somit liest der Apriori Algorithmus die Daten aus dem Input.txt File, bearbeitet diese und schreibt anschließend die Ergebnisse in den Output.txt File [30]. Sämtliche weitere Informationen über das SPMF kann man von der offiziellen Webseite des Sequential Pattern Mining Frameworks erhalten [31].

4.1.2 SPMF Data Mining Algorithmen

In diesem Kapitel wird anhand eines Beispiels die Funktionsweise des Algorithmus *FPGrowth_Itemsets_with_Strings* näher unter die Lupe genommen. Die Erläuterung von diesem Algorithmus hat in Rahmen dieser Diplomarbeit eine sehr wichtige Bedeutung, zumal die Integration des Sequential Pattern Mining Frameworks in die Eclipse Entwicklungsumgebung mit Hilfe von diesem Algorithmus realisiert ist.

FPGrowth_Itemsets_with_strings:
Der FP-Growth Algorithmus wurde bereits in dem Kapitel 2.4 sehr detailliert erläutert. Aus diesem Grund wird hier auf die detaillierte Funktionsweise des FP-Growth Algorithmus verzichtet. Es wird lediglich auf die Vorgehensweise des *FPGrowth_Itemsets_with_Strings* in dem SPMF eingegangen.

Um die Funktionsweise und das Ergebnis von diesem Algorithmus zu verstehen, wird der Inhalt der Transaktionsdatenbank D aus der Tabelle 1 dem *FPGrowth_Itemsets_with_Strings* als Input.txt File zur Verfügung gestellt.

```
Apfel Mandarine Melone
Apfel Mandarine Zitrone
Apfel Birne
Mandarine Melone Apfel
Apfel Mandarine Melone Zitrone
Birne
```

Abbildung 19: Input.txt File von *FPGrowth_Itemsets_with_Strings* [32]

Die Abbildung 19 repräsentiert den Input.txt File für den Algorithmus. Jede Transaktion ist hier durch eine Zeile dargestellt und besteht aus einer Menge von Items, welche durch ein Leerzeichen voneinander getrennt sind. Das Ende einer Zeile verweist dementsprechend auf das Ende einer Transaktion. Der Input.txt File enthält somit insgesamt 6 Transaktionen (T1, T2, T3, T4, T5, T6) und 5 Items (Apfel, Melone, Mandarine, Zitrone, Birne). Beispielweise repräsentiert die fünfte Zeile in dem Input.txt File die Transaktion T5 und enthält die Items {Apfel, Mandarine, Melone, Zitrone} [32]. Geht man zudem von einem Minimumsupportwert von 0,5 ($s_{min} = 0{,}5 = 50\%$) aus, dann produziert dieser Algorithmus das in Abbildung 20 dargestellte Ergebnis in dem Output.txt File [32]. Jede Zeile in dem Output.txt File steht für einen Frequent Itemset. In jeder Zeile werden als Erstes die Frequent Itemsets aufgelistet, die durch Leerzeichen voneinander getrennt sind. Danach steht das Zeichen ":" und es folgt darauf ein Integerwert, der den Supportwert von dem Frequent Itemset angibt. Die dritte Zeile in dem Output.txt File drückt aus, dass das Frequent Itemset aus den Items {Apfel, Mandarine, Melone} besteht und einen Support von 3 Transaktionen besitzt [32].

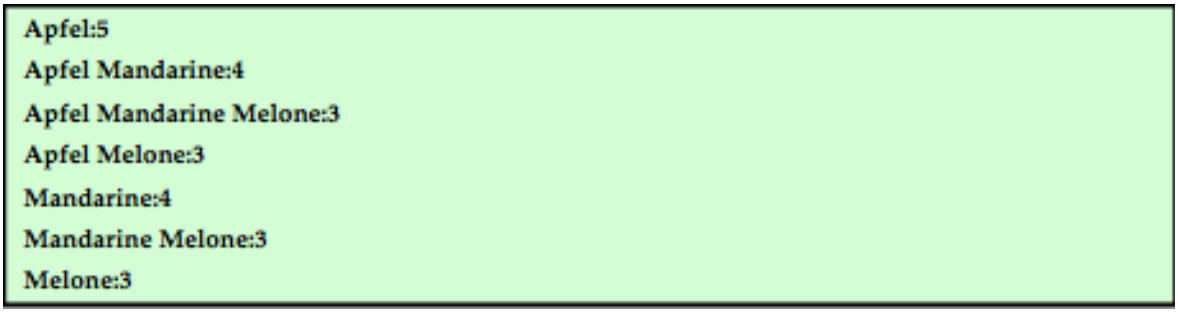

Abbildung 20: Output.txt File von FPGrwoth_Itemsets_with_Strings [32]

4.2 Windowbuilder

Beim Windowbuilder handelt es sich um einen GUI Builder, der dem Entwickler ermöglicht, Benutzeroberflächen sehr leicht zu erstellen ohne dabei unnötige Zeit mit dem Schreiben von Quellcodes zu verbringen. Die Steuerelemente können dabei einfach durch Drag and Drop Methode in das GUI eingefügt werden [33]. Die Abbildung 21 zeigt die Oberfläche des Windowbuilder Plug-Ins, welches aus den Hauptkomponenten Source View, Design View, Component Tree, Property Pane, Palette, Wizards, Toolbars & Context Menus besteht. Eine sehr wichtige Eigenschaft von dem Windowbuilder ist die bi-direktionale Codegenerierung. Fügt der Entwickler ein Steuerelement in der Design Sicht (View), wird in der Source Sicht (View) der entsprechende Quellcode automatisch generiert und umgekehrt. Die Abbildung 22 stellt die Eigenschaft bi-direktionale Codegenerierung von Windowbuilder graphisch dar [34].

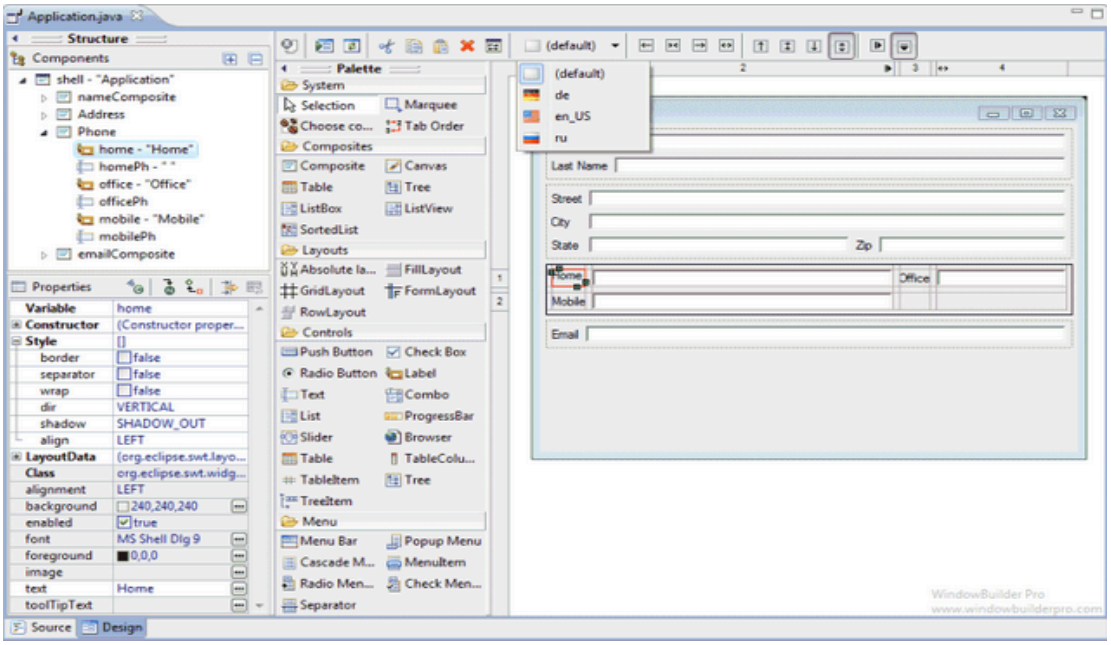

Abbildung 21: Benutzeroberfläche des Windowbuilder Plug-Ins [33]

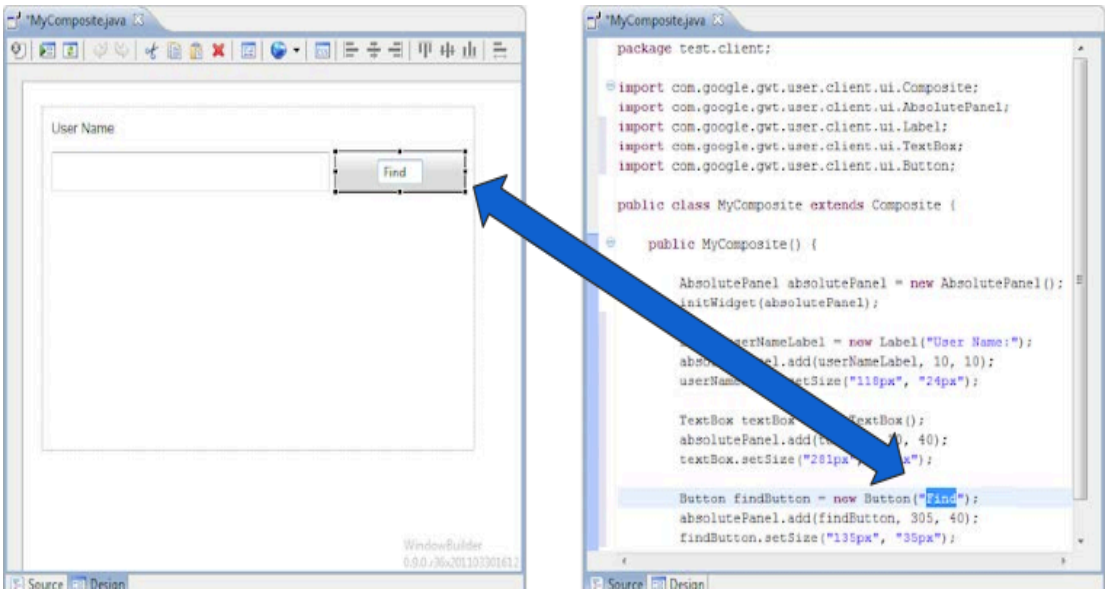

Abbildung 22: Bi-direktionale Codegenerierung in Windowbuilder [34]

5 Anforderungen und Konzept

In diesem Kapitel wird das Software Repository Mining Plug-In (SRM Plug-In) eingeführt und näher erläutert. Zunächst werden die Anforderungen an das Plug-In kurz vorgestellt. Hinterher wird das Konzept gefolgt durch die Struktur und Workflow präsentiert. Zuletzt findet dann eine detaillierte Erläuterung der Korrelation der einzelnen Komponenten des Plug-Ins anhand eines Beispiels statt.

5.1 Anforderungen

Hier werden die Anforderungen an das SRM Plug-Ins festgelegt. In erster Linie erfolgt die Erläuterung über die Überarbeitung des SPMF Data Mining Algorithmus, der innerhalb des SRM Plug-Ins für die Durchführung der Frequent-Itemset-Analyse angewendet wird. Im Anschluss daran wird die Integration der Data Mining Analyse ins SRM Plug-In beschrieben.

5.1.1 Überarbeitung von SPMF Data Mining Algorithmus

Da das SRM Plug-In für die Durchführung der Frequent-Itemset-Analyse den Algorithmus *FPGrowth_Itemsets_with_Strings* von dem SPMF Data Mining Framework benutzt, ist in erster Linie die Überarbeitung und Anpassung von diesem Algorithmus erforderlich. Wie im Abschnitt 4.1.2 bereits geschildert ist, kann der Algorithmus *FPGrowth_Itemsets_with_Strings* Daten nur von Text Files lesen und die Ergebnisse in Text Files schreiben.

Die erste Anforderung an das SRM Plug-In bezüglich der Durchführung der Frequent-Itemset-Analyse ist dies, dass der Algorithmus, welcher die Frequent-Itemset-Analyse durchführen soll, Daten von einer Datenbanktabelle liest, und die Ergebnisse in eine andere Datenbanktabelle speichert.

Eine weitere Anforderung an das SRM Plug-Ins bezieht sich auf die Ergebnisse, die durch den Algorithmus *FPGrowth_Itemsets_with_Strings* erzeugt werden. Betrachtet man beispielweise die zweite Zeile des Output.txt Files aus der Abbildung 20, so kann man die Ausgabe (Apfel Mandarine: 4) erkennen. Diese Ausgabe besagt, dass das Frequent Itemset {Apfel Mandarine} einen Support von 4 hat und somit in vier Transaktionen auftritt. Das SRM Plug-In aber benötigt für die weitere Verarbeitung der aus der Frequent-Itemset-Analyse resultierten Daten bzw. Frequent Itemsets, Informationen darüber, in welchen Transaktionen diese Frequent Itemsets überhaupt auftreten. Bezogen auf das Beispiel von vorhin, muss in der Outputdatenbanktabelle der Frequent-Itemset-Analyse des SRM Plug-Ins eine Spalte existieren, die diese vier Transaktionen des Frequent Itemsets {Apfel Mandarine} beinhaltet.

5.1.2 Integration von Data Mining ins Eclipse Tool

Die Durchführung der Frequent-Itemset-Analyse bildet den einen Teil des SRM Plug-Ins. Ein weiterer und somit auch der wichtigste Teil der Anforderungen an das Plug-In ist die Integration von Data Mining Analysen in das SRM Plug-In.

Nachdem die Frequent-Itemset-Analyse durchgeführt wurde und die Ergebnisse sich in der Outputdatenbanktabelle befinden, erfolgt dann die Bereitstellung dieser Ergebnisse an den Entwickler in dem SRM Plug-In. Aus diesem Grund geht es hier darum zu bestimmen, welche Informationen aus der Outputdatenbanktabelle dem Entwickler bereitgestellt werden sollen.

Die Anforderungen an dem SRM Plug-In sind dann wie folgt:

1. Der Entwickler soll die Frequent-Itemset-Analyse von dem SRM Plug-In aus starten können.

2. Nachdem Selektieren bzw. Ändern eines Files durch den Entwickler, sollen die gekoppelten Dateiänderungen (Coupled Changes) und die Transaktionen jeweils separat dem Entwickler vorgeschlagen werden.

3. Weiterhin soll der Entwickler auch die Möglichkeit haben, eine Transaktion auszuwählen, um sich die entsprechenden Einträge zu der selektierten Transaktion separat anzeigen zulassen.

5.2 Architektur SRM Plug-In

Hier erfolgt die Darstellung der Architektur des SRM Plug-Ins. Die Abbildung 23 stellt diese Architektur dar. Eigentlich befinden sich alle dargestellten Komponenten in dem Plug-In. Für ein besseres Verständnis wurden diese Komponenten hier getrennt dargestellt.

Zusätzlich ist noch zu erwähnen, dass es sich bei dem FPGA.jar File um den *FPGrowth_Itemsets_with_Strings* Algorithmus handelt. Das SPMF ist eine unter der GPL (General Public License) lizenzierte Software und somit auch deren Algorithmen. Das SRM Plug-In ist unter der EPL (Eclipse Public License). Da diese beiden Lizenzen inkompatibel sind, darf der Quellcode vom *FPGrowth_Itemsets_with_Strings* Algorithmus nicht Teil vom SRM Plug-In sein, sondern darf nur als externer Jar File in das Plug-In integriert werden [41].

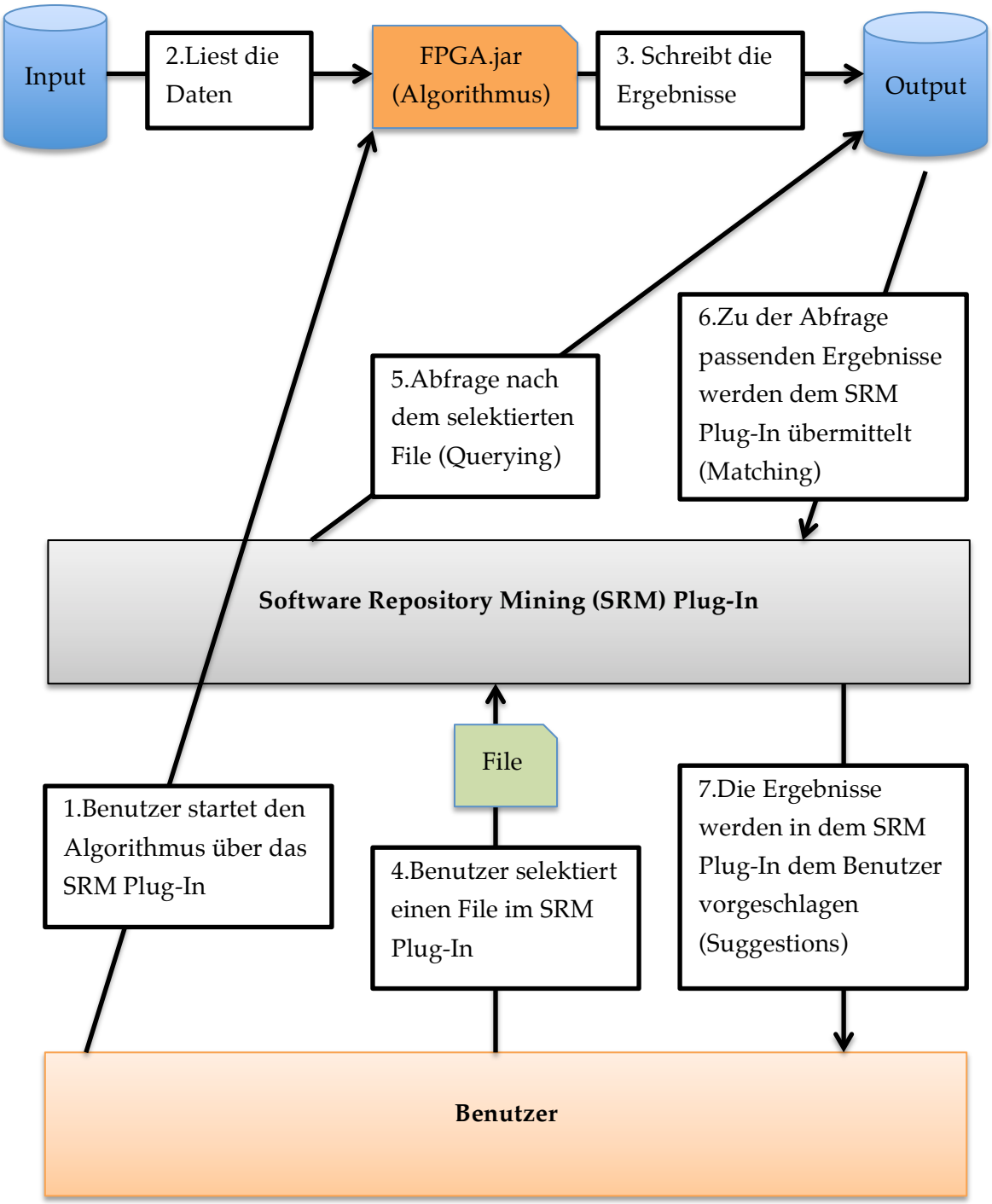

Abbildung 23: Architektur SRM Plug-In

5.3 Struktur und Workflow des SRM Plug-Ins

Dieser Abschnitt umfasst die Struktur und den Workflow des SRM Plug-Ins.

5.3.1 Struktur SRM Plug-In

Das Plug-In besteht insgesamt aus fünf Views und einem Editor. Neben den Standardkomponenten der Eclipse IDE wurden im Rahmen dieser Diplomarbeit vier weitere Views hinzugefügt. Die Abbildung 24 zeigt die Struktur des SRM Plug-Ins. Installiert der Benutzer den SRM Plug-In, erscheint ihm diese Startseite. Die einzelnen Komponenten des SRM Plug-Ins werden im Folgenden aufgelistet und beschrieben:

1. **Execution View**: Durch diesen View startet der Benutzer die Frequent-Itemset-Analyse.

2. **Project Explorer:** Der Benutzer selektiert einen File in seinem Project Explorer.

3. **Coupled Changes:** Die gekoppelten Dateiänderungen werden dem Benutzer in diesem View vorgeschlagen.

4. **Commit View:** Die Transaktionen, welche das selektierte File und seine gekoppelten Dateiänderungen beinhalten, werden in diesem View angezeigt.

5. **Commit Message View:** Die Einträge zu dem in der Commit View durch den Benutzer selektierte Transaktion wird in diesem View dargestellt.

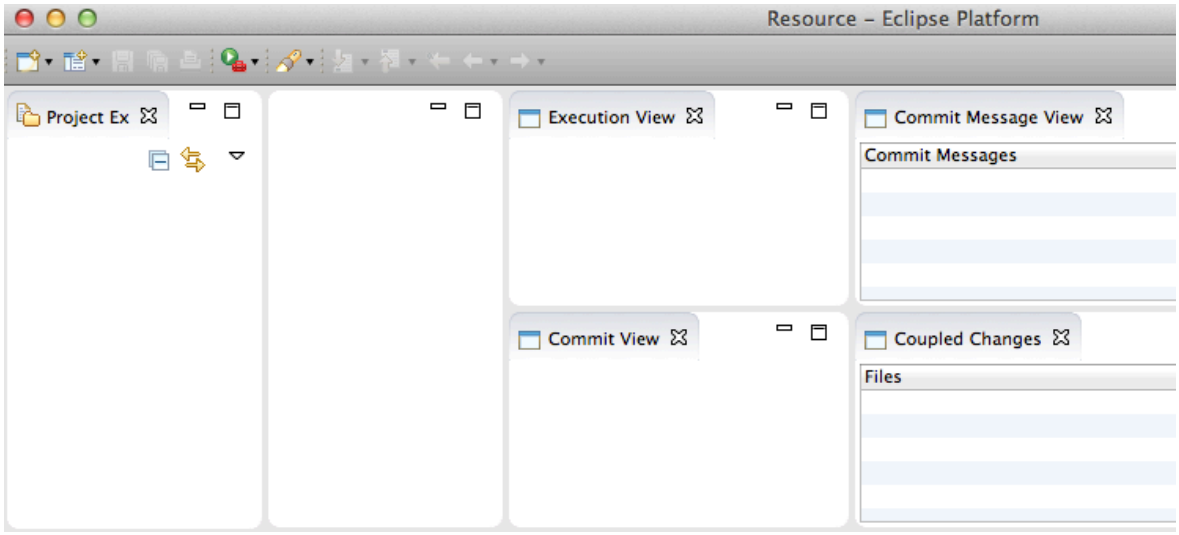

Abbildung 24: Struktur SRM Plug-In

5.3.2 Workflow SRM Plug-In

Dieses Kapitel stellt den Workflow des SRM Plug-Ins dar. Dabei versteht man unter Workflow den Datenfluss in dem Plug-In.

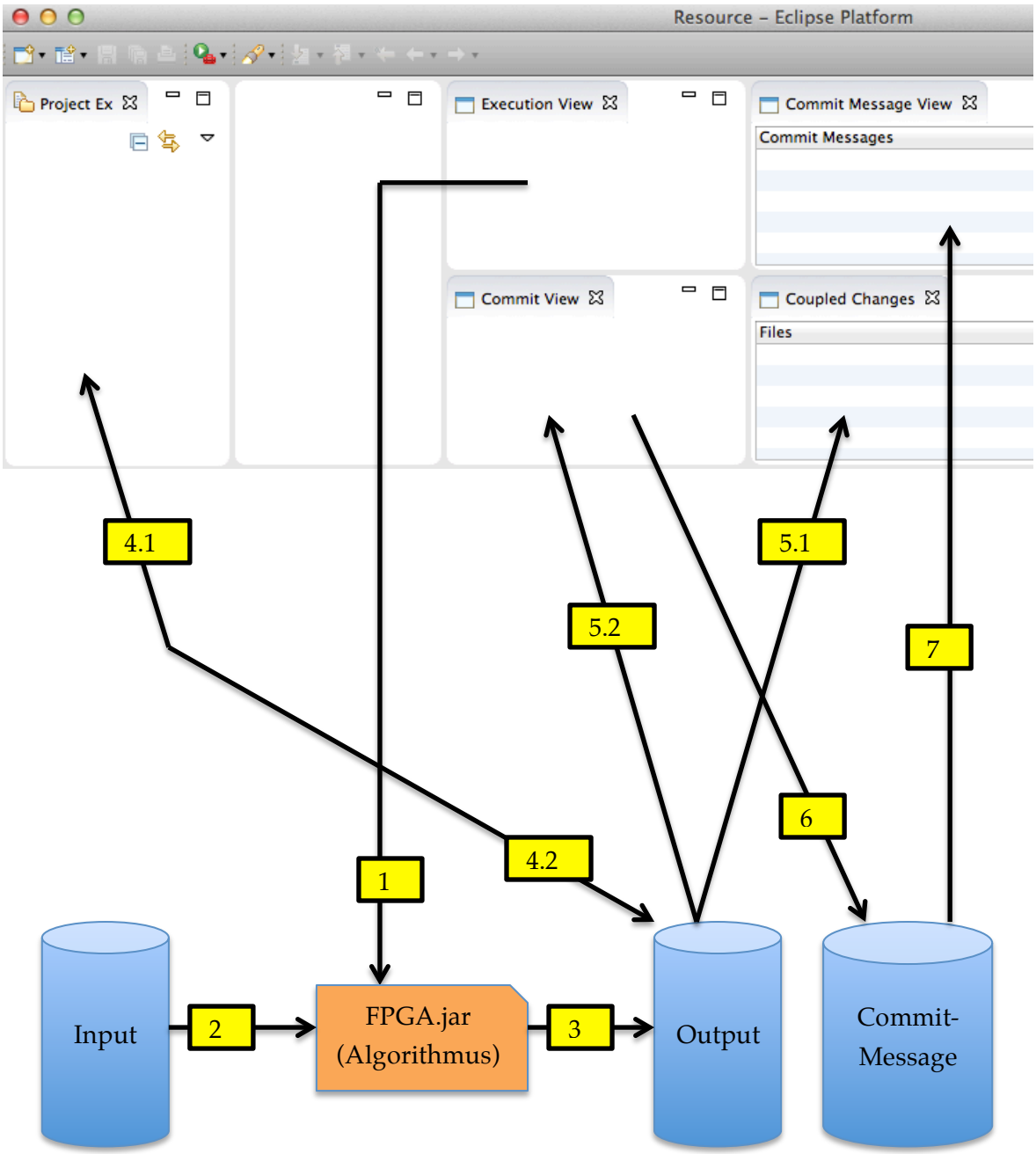

Abbildung 25: Workflow SRM Plug-In

Im Folgenden werden die einzelnen Schritte des Workflows beschrieben:

1. Der Benutzer gibt im Execution View einen Minimumsupportwert ein und betätigt den „Start Execution“ Button. Somit wird der FPGA.jar File aufgerufen und ihm der Minimumsupportwert übergeben.

2. Der FPGA.jar File liest die Daten aus der Inputdatenbanktabelle und führt die Frequent-Itemset-Analyse durch.

3. Danach schreibt der FPGA.jar File die Ergebnisse der Frequent-Itemset-Analyse in die Outputdatenbanktabelle.

4.1. Der Benutzer selektiert einen File in dem Project Explorer.

4.2. Das selektierte File wird mit dem Inhalt der Outputdatenbanktabelle verglichen.

5.1. Die gekoppelten Dateiänderungen werden im Coupled Changes angezeigt.

5.2. Die CommitIDs werden in dem Commit View angezeigt.

6. Der Benutzer selektiert einen CommitID und es wird in der Commit Message Tabelle nach Einträgen zu dem selektierten CommitID gesucht.

7. Die Einträge zu dem im Punkt 6 selektierten CommitID werden dann in dem Commit Message View angezeigt.

5.4 Korrelation der einzelnen SRM Plug-In Komponenten

Die Relation der einzelnen SRM Plug-In Komponenten zueinander werden in diesem Textabschnitt anhand von Beispielwerten beschrieben.

5.4.1 Execution View

Execution View ist der Bereich des Plug-Ins, womit die Frequent-Itemset-Analyse gestartet wird. Es hat einen Textfeld und einen Button. In das „Choose minsupp“ Textfeld kann der Benutzer einen Wert in dem Intervall [0.1, 1.0] eingeben. Betätigt der Benutzer danach den „Start Execution“ Button, dann startet die Frequent-Itemset-Analyse. Die Abbildung 26 zeigt den Execution View mit Beispielwerten.

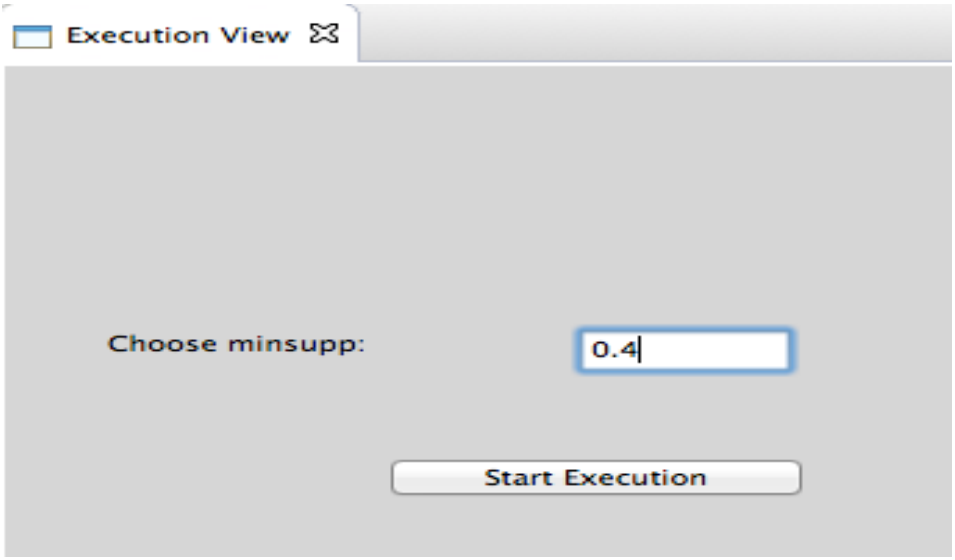

Abbildung 26: Execution View mit Beispielwerten

5.4.2 FPGA.jar File

Das FPGA.jar File ist für die Durchführung der Frequent-Itemset-Analyse verantwortlich. Es liest die Daten aus einer Inputdatenbanktabelle, führt die Frequent-Itemset-Analyse durch und schreibt die Ergebnisse in eine Outputdatenbanktabelle. Die Tabelle 4 repräsentiert die Beispielinputtabelle des FPGA.jar Files.

Tabelle 4:Beispielinputtabelle für das FPGA.jar File

CommitID (PK)	Item1	Item2	Item3
123	Project/src/example/ Test1.java	Project/src/example/ Test2.java	Project/src/example/ Test3.java
124	Project/src/example/ Test3.java	Project/src/example/ Test4.java	Project/src/example/ Test5.java
125	Project/src/example/ Test3.java	Project/src/example/ Test5.java	Project/src/example/ Test6.java
126	Project/src/example/ Test1.java	Project/src/example/ Test2.java	Project/src/example/ Test3.java

Die Beispielinputtabelle besitzt vier Spalten. Die erste Spalte ist die CommitID Spalte. Diese Spalte beinhaltet die CommitIDs von den Transaktionen. In dieser Beispielinputtabelle existieren insgesamt vier Transkationen (123, 124, 125, 126). Jede Zeile stellt eine Transaktion dar, die aus einer Menge von Items besteht. Die Items sind als Pfade dargestellt. In einer Transaktion darf ein Item nur einmal vorkommen. Diese Daten der Beispielinputtabelle werden von dem FPGA.jar File Transaktion für Transaktion gelesen, analysiert und die Ergebnisse in die Outputtabelle geschrieben.

Tabelle 5: Beispieloutputtabelle des FPGA.jar Files

O_ID (PK)	CommitIDs	Support	Item1	Item2	Item3
1	124:125	2	Project/src /example/ Test5.java	Project/src /example/ Test3.java	*NULL*
2	123:126	2	Project/src /example/ Test2.java	Project/src /example/ Test3.java	*NULL*
3	123:126	2	Project/src /example/ Test2.java	Project/src /example/ Test3.java	Project/src /example/ Test1.java
4	123:126	2	Project/src /example/ Test2.java	Project/src /example/ Test1.java	*NULL*
5	123:126	2	Project/src /example/ Test1.java	Project/src /example/ Test3.java	*NULL*

Die Tabelle 5 zeigt die Beispieloutputtabelle, die nach der Frequent-Itemset-Analyse entstanden ist. Da der Minimumsupportwert = 0.4 = 2 Transaktionen ist, werden bei der Frequent-Itemset-Analyse nur jene Itemsets in die Beispieloutputtabelle geschrieben, welche in mindestens zwei Transaktionen vorkommen.

Beispielsweise besagt die dritte Zeile aus der Tabelle 5, dass die Files {*Project/src/example/Test2.java;Project/src/example/Test3.java;Project/src/example/Test1.java*} einen Supportwert von 2 haben und in den Transaktionen mit den CommitIDs {*123* und *126*} auftreten.

5.4.3 Project Explorer und Editor

An dieser Stelle kommen die Standardkomponenten des Eclipse IDE zum Einsatz. Die Entwicklung eines Plug-Ins zeichnet sich wie im Kapitel 2.5 auch dargelegt, durch die Erweiterung der bereits existierenden Komponenten des Eclipse IDE aus. Im Rahmen dieser Diplomarbeit wurden diese Eclipse IDE Komponenten durch die Views (Execution View, Coupled Changes, Commit View und Commit Message View) erweitert. Die Abbildung 27 repräsentiert die Übersicht von den Standardkomponenten der Eclipse Workbench und untergliedert diese in vier Bereiche, die im Folgenden detaillierter erläutert werden.

Komponenten der Eclipse Workbench:

1. **Project Explorer View:**
   Dieses View ist für das Verwalten von Projekten zuständig. Importieren und Erstellen von Projekten, Klassen usw. sind durch dieses Project Explorer View möglich.

2. **Editor:**
   Dieser Beriech ermöglicht das Schreiben und Anzeigen von Quellcodes des in dem Project Explorer View erstellten bzw. selektierten Files.

3. **Outline View:**
   In diesem Bereich werden wichtige Informationen über die Klassen des Project Explorer Views angezeigt. Diese Informationen sind beispielsweise die Methoden und Attribute zu den entsprechenden Files aus dem Project Explorer View. Dieser View erleichtert die Arbeit des Entwicklers, da es viele verschiedene Möglichkeiten anbietet, um einen besseren Überblick über die Methoden und Attribute der implementierten Klassen zu bekommen. Einer der Vorteile, welcher diese Outline View mit sich bringt, ist die Tatsache, dass es den Entwicklern die Möglichkeit gibt, durch einen Klick auf eine Methode auf den entsprechenden Quellcodeabschnitt, wo die Methode implementiert ist, zu gelangen. Dadurch wird die aufwendige Suche nach den Methoden in Klassen mit großem Umfang vermieden.

4. Diese View stellt die Konsolenausgaben dar. Hier werden unter anderem Informationen ausgegeben, welche die Fehler beim Kompilieren eines laufenden Programmes anzeigen [35].

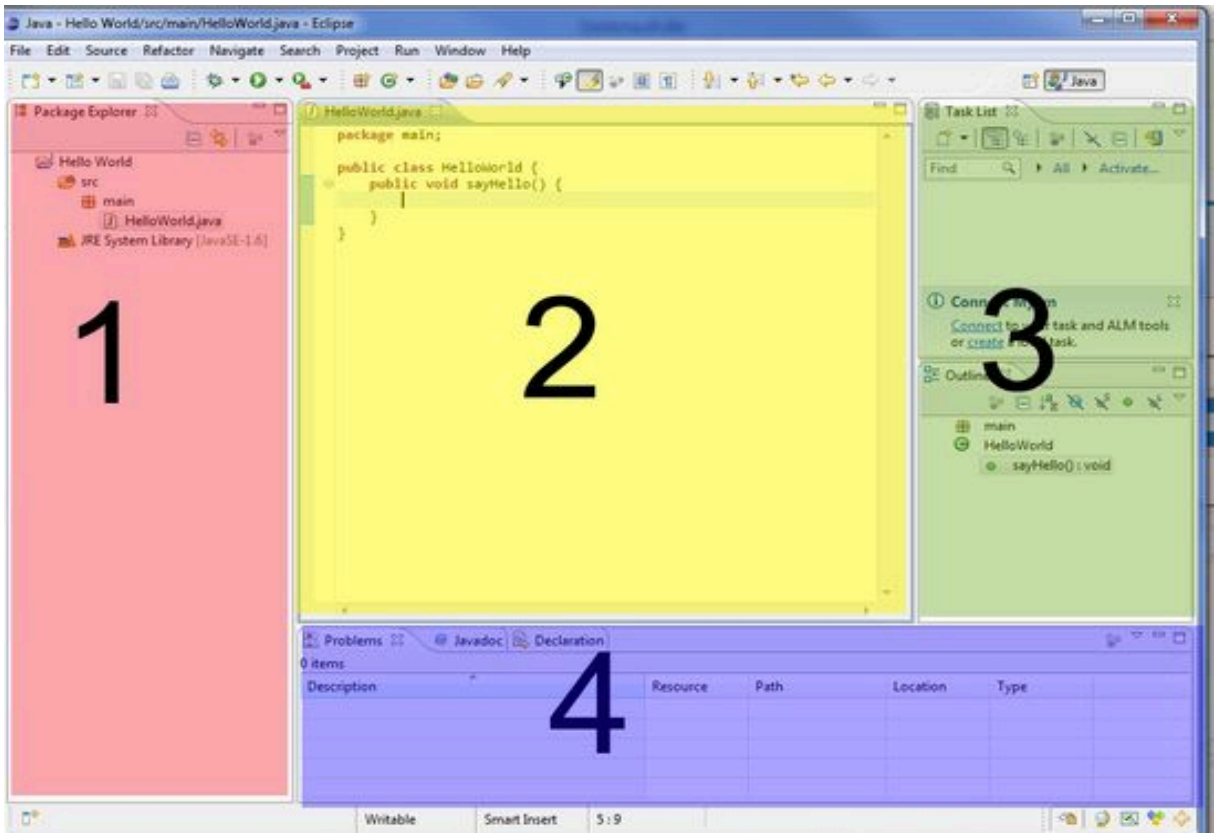

Abbildung 27: Übersicht über den Eclipse Workbench [35]

Abbildung 28 zeigt den Project Explorer mit einem Beispielprojekt. Der Benutzer importiert ein Eclipse Projekt namens „*Project*“, das einige Klassen beinhaltet. Klickt der Benutzer auf den File *Test1.java*, dann wird der Inhalt bzw. Quellcode von diesem File in dem Editor nebenan dargestellt.

Weiterhin wird durch den Klick auf das File der Pfad von diesem selektierten File ermittelt, transformiert und mit dem in der Tabelle 5 dargestellten Beispieloutputtabelle verglichen. Bei dem Vergleich wird in der Outputtabelle nach dem selektierten File gesucht und die entsprechenden Bereiche, die das selektierte File beinhalten mit samt seinen CommitIDs und seinen gekoppelten Dateiänderungen aus der Outputtabelle selektiert und anschließend für die Weiterverarbeitung an die anderen Klassen übergeben. Diese Klassen verarbeiten dann diese selektierten Bereiche so, indem sie die gekoppelten Dateiänderungen von ihren CommitIDs trennen, um diese dann an die entsprechenden Views weiterzuleiten. Dabei

werden die gekoppelten Dateiänderungen an das View Coupled Changes und die CommitIDs dem Commit View übergeben.

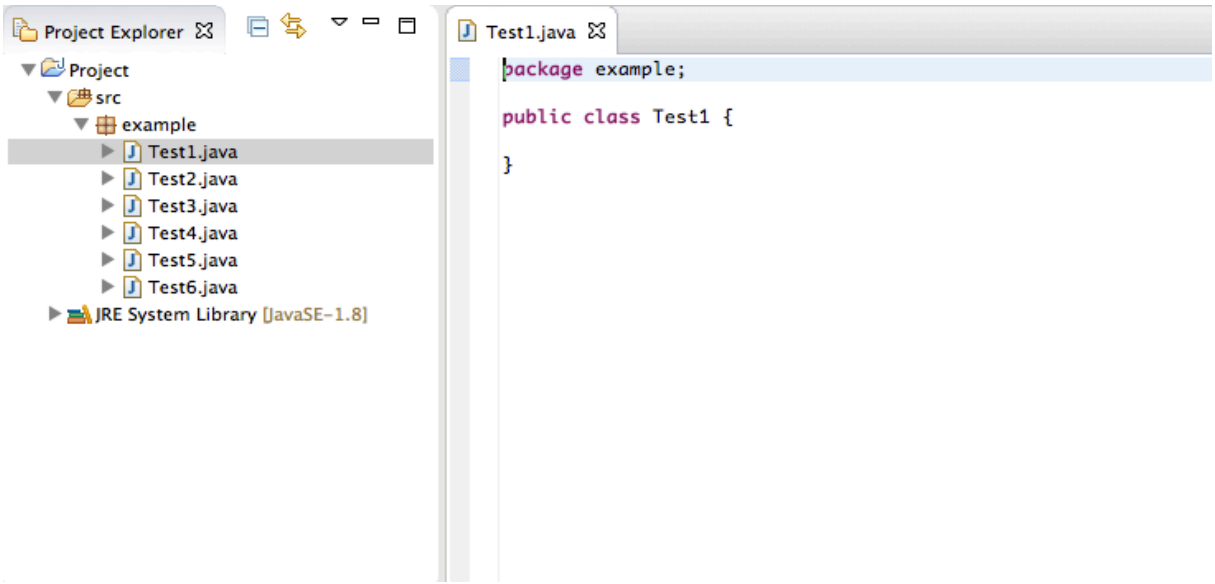

Abbildung 28: Project Explorer mit einem Beispielprojekt

5.4.4 Coupled Changes

Dieses View listet die gekoppelten Dateiänderungen nach absteigenden Supportwerten auf. Selektiert der Benutzer in dem Project Explorer einen File, wird überprüft, ob das selektierte File in der Outputtabelle vorhanden ist. Wenn es der Fall ist, dann werden die gekoppelten Dateiänderungen des selektierten Files in dem View Coupled Changes dargestellt. Die Abbildung 29 repräsentiert die gekoppelten Dateiänderungen des Files *Test5.java* mit dem Pfad: *Project/src/example/Test5.java*.

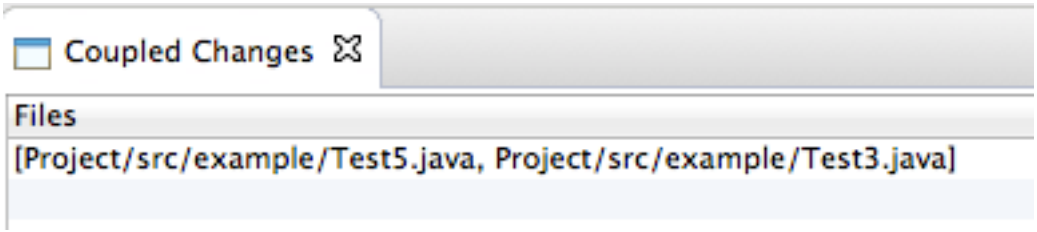

Abbildung 29: Coupled Changes mit Beispielwerten

Klickt der Benutzer hingegen auf einen File, welcher nicht in der Outputtabelle vorhanden ist, dann erscheint die folgenden Meldung im Coupled Changes: „The selected File does not exist in the Outputtable“ („Das selektierte File existiert nicht in der Outputtabelle“). Die Abbildung 30 zeigt diese Fehlermeldung.

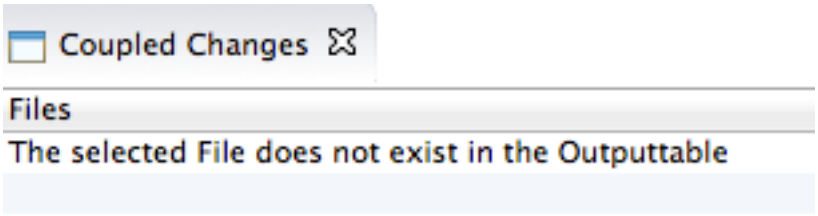

Abbildung 30: Fehlermeldung im Coupled Changes

5.4.5 Commit View

In diesem View werden die CommitIDs, welche das selektierte File und seine gekoppelten Dateiänderungen besitzen in einem Combobox aufgelistet.

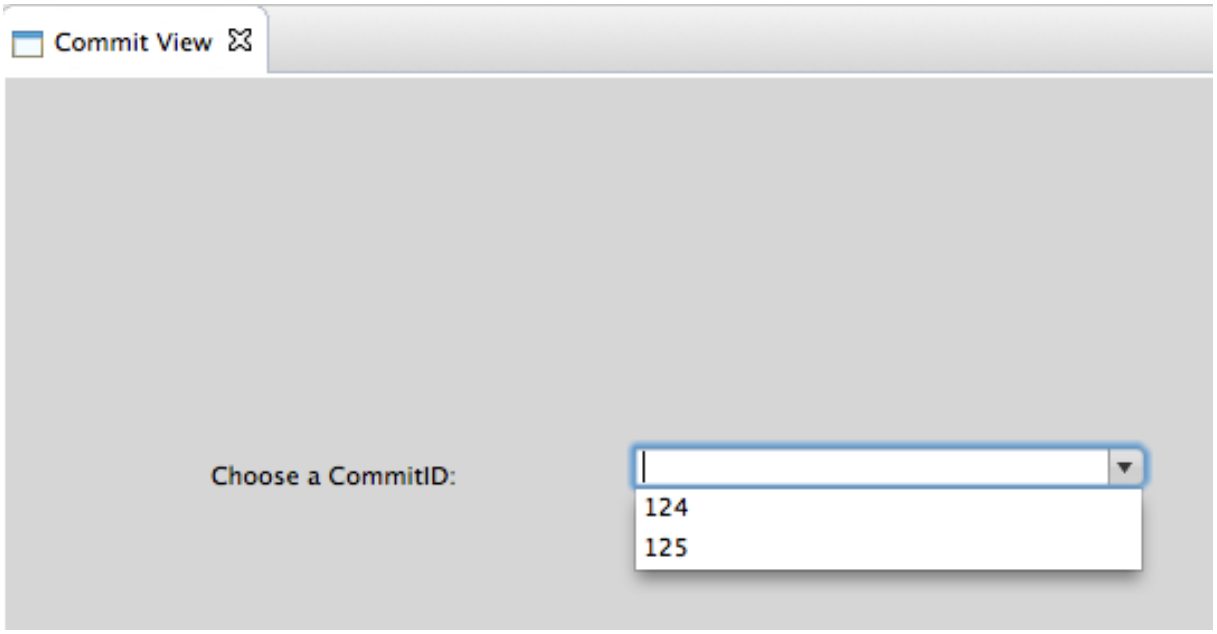

Abbildung 31: Auflistung der CommitIDs im Commit View mit Beispielwerten

In Abbildung 31 wird das Commit View mit Beispielwerten angezeigt. Diese Werte besagen, dass die gekoppelten Dateiänderungen aus der Abbildung 29 in den Transaktionen {*124* und *125*} der Outputtabelle (Tabelle 5) auftreten.

Der Benutzer hat die Möglichkeit eines von den CommitIDs in diesem Commit View auszuwählen, um sich die entsprechenden Einträge aus der Commit Message Tabelle in dem Commit Message View anzeigen zu lassen. Die Tabelle 6 repräsentiert die Commit Message Tabelle.

Tabelle 6: Commit Message Tabelle

Commit MessageID (PK)	Commit_ID	Commit Message
1	123	Die Arbeit ist erledigt
2	124	auch während der Eingabe
3	125	während die anderen Klassen
4	126	nicht vorhanden ist

5.4.6 Commit Message View

Nachdem der Benutzer in dem vorherigen Schritt einen CommitID ausgewählt hat, erscheinen die Einträge zu dem selektierten CommitID in dem Commit Message View des SRM Plug-Ins. Selektiert beispielweise der Benutzer die CommitID *125*, dann erscheint in dem Commit Message View der Eintrag *„während die anderen Klassen“*, wie es auch in der Abbildung 32 zu sehen ist.

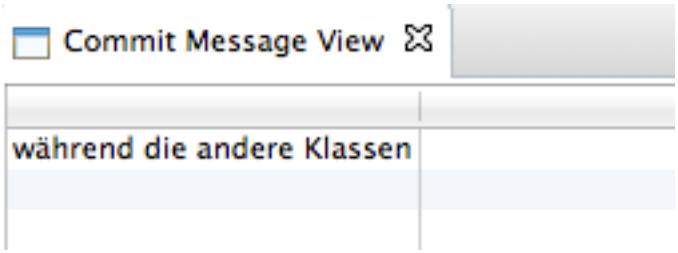

Abbildung 32: Commit Message View mit Beispielwerten

6 Implementierung

Zielrichtung dieses Kapitels ist die Umsetzung des SRM Plug-In Konzeptes.

6.1 Entwicklungsumgebung

Das SRM Plug-In wurde mit Hilfe der Entwicklungsumgebung Eclipse Mars als eine reine Java Anwendung realisiert. Das Plug-In wurde dabei neben dem Eclipse SDK auch noch mit dem Einsatz des PDE (Plug-In Development Environment) entwickelt. Zuletzt wurden die GUIs durch die Nutzung des Windowbuilder Tools mit Hilfe der Graphikbibliothek SWT erstellt.

6.2 Registrierung des SRM Plug-Ins

In erster Linie wird ein Plug-In Projekt mit dem Namen *de.uni_stuttgart.srmplugin* erstellt. Im Anschluss daran wird das bereits vorhandene Eclipse IDE um die benötigten Views (Execution View, Coupled Changes, Commit View und Commit Message View) erweitert. Abbildung 33 zeigt, wie die Views dem SRM Plug-In hinzugefügt werden.

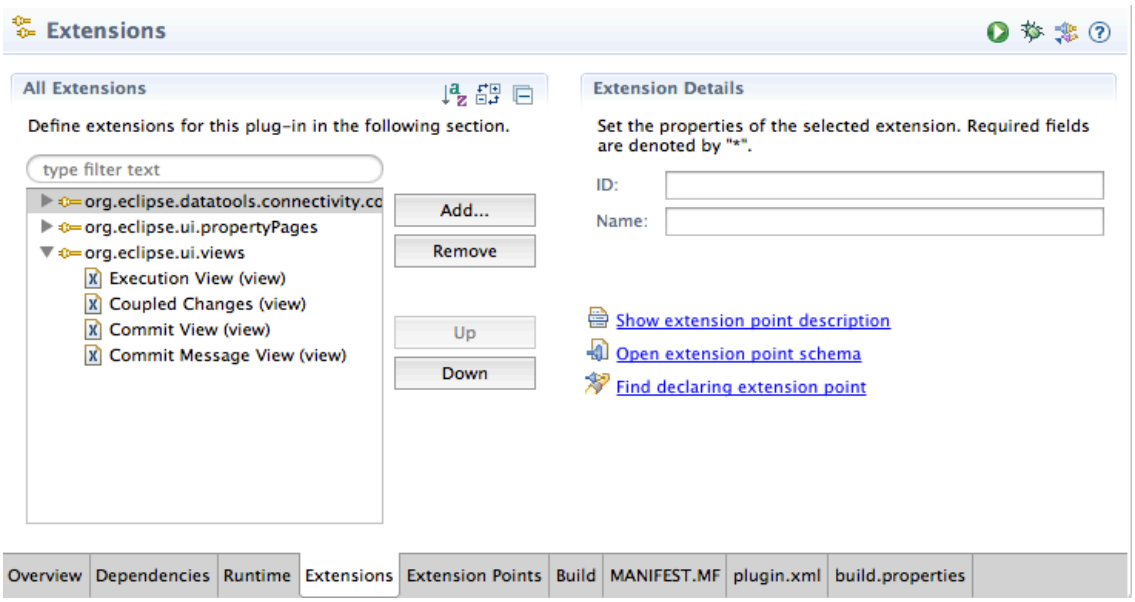

Abbildung 33: Hinzufügen der Views in das SRM Plug-In

Weiterhin werden die Jar Files: FPGA.jar und der MYSQL Treiber: mysql-connector-java-5.1.34-bin.jar in den Classpath des Runtime Tabs hinzugefügt. Abbildung 34 zeigt wie diese Jar Files in das SRM Plug-In integriert wurden.

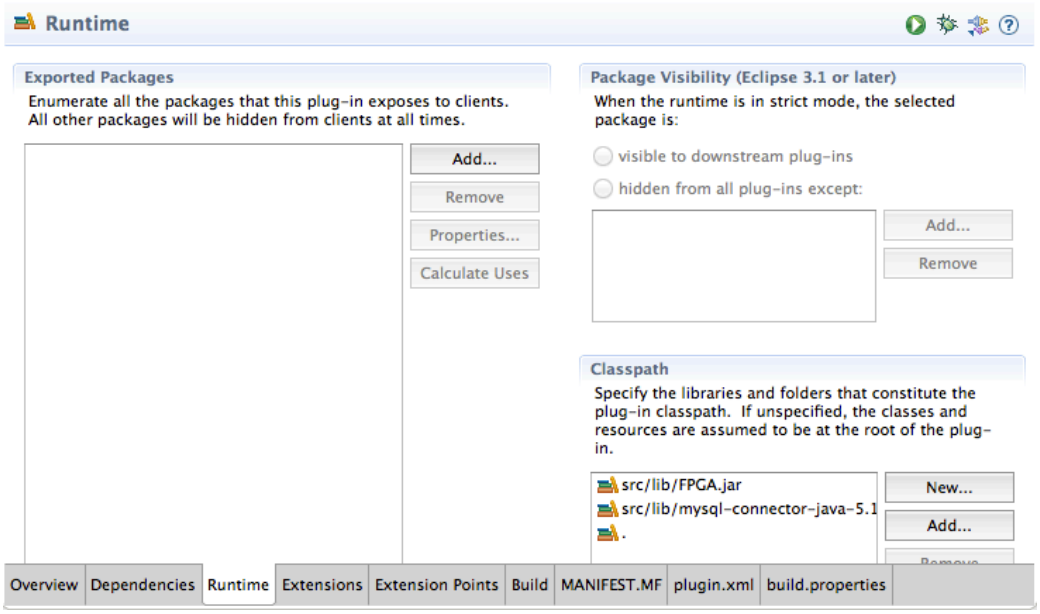

Abbildung 34: Integration von MYSQL Treiber und FPGA.jar File in das SRM Plug-In

Zum Schluss werden die Abhängigkeiten (Dependencies) in den Dependencies Tab eingefügt. Abbildung 35 zeigt alle Abhängigkeiten, die für die Entwicklung des SRM Plug-Ins erforderlich sind.

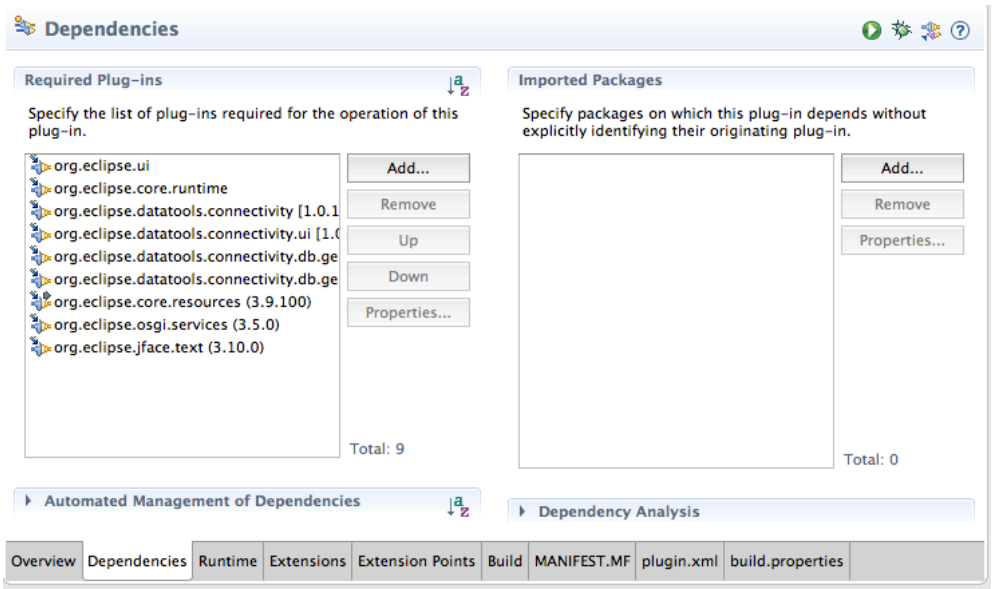

Abbildung 35: Hinzufügen der Abhängigkeiten (Dependencies) in das SRM-Plug-In

Nachdem das Plug-In erstellt und die entsprechenden Views und die Abhängigkeiten (Dependencies) hinzugefügt wurden, werden alle Informationen über das Plug-In in dem File plugin.xml angezeigt. Listing 1 zeigt den Quellcode von plugin.xml des SRM Plug-Ins.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>

  <extension
      point="org.eclipse.datatools.connectivity.connectionProfile">
    <newWizard
        name="%wizard.name"
        icon="icons/new_db_element.gif"
        profile="my.connection.profile"
        class="org.mycompany.myconnectionprofile.MyConnectionProfileWizard"
        id="my.connection.profile.newWizard">
    </newWizard>
    <connectionProfile
        pingFactory="org.eclipse.datatools.connectivity.db.generic.JDBCConnectionFactory"
        name="%connection.profile.name"
        icon="icons/jdbc_16.gif"
        category="org.eclipse.datatools.connectivity.db.category"
        id="my.connection.profile">
    </connectionProfile>
  </extension>
  <extension
      point="org.eclipse.ui.propertyPages">
    <page
        name="%connection.profile.proppage.name"
        class="org.mycompany.myconnectionprofile.MyConnectionProfilePropertyPage"
        id="my.connection.profile.propertyPage">
      <filter
          name="org.eclipse.datatools.profile.property.id"
          value="my.connection.profile">
      </filter>
      <enabledWhen>
        <instanceof
            value="org.eclipse.datatools.connectivity.IConnectionProfile">
        </instanceof>
      </enabledWhen>
    </page>
  </extension>
  <extension
      point="org.eclipse.ui.views">
    <view
        category="de.uni_stuttgart.SRMPlugIn.category"
        class="de.uni_stuttgart.srmplugin.views.ExecutionView"
        id="de.uni_stuttgart.SRMPlugIn.executionview"
        name="Execution View"
        restorable="true">
    </view>
    <view
        category="de.uni_stuttgart.SRMPlugIn.category"
        class="de.uni_stuttgart.srmplugin.views.CoupledChanges"
        id="de.uni_stuttgart.SRMPlugIn.coupledchanges"
```

```
        name="Coupled Changes"
        restorable="true">
    </view>
    <view
        category="de.uni_stuttgart.SRMPlugIn.category"
        class="de.uni_stuttgart.srmplugin.views.CommitView"
        id="de.uni_stuttgart.SRMPlugIn.commitview"
        name="Commit View"
        restorable="true">
    </view>
    <view
        category="de.uni_stuttgart.SRMPlugIn.category"
        class="de.uni_stuttgart.srmplugin.views.CommitMessageView"
        id="de.uni_stuttgart.SRMPlugIn.commitmessageview"
        name="Commit Message View"
        restorable="true">
    </view>
    <category
        id="de.uni_stuttgart.SRMPlugIn.category"
        name="SRM Plugin">
    </category>
  </extension>

</plugin>
```

Listing 1: Quellcode plugin.xml vom SRM-Plug-In

6.3 Auflistung der einzelnen Klassen

Hier werden alle Klassen des SRM Plug-Ins und deren Funktionen in der Tabelle 7 aufgelistet und beschrieben.

Tabelle 7: Überblick über die Klassen des SRM Plug-Ins und deren Funktionen

Klassen	Funktionen
Activator.java	Diese Klasse ist für das Starten und Beenden des SRM Plug-Ins zuständig.
ExecutionView.java (View)	Diese View ist für das Starten der Frequent-Itemset-Analyse verantwortlich. Der Benutzer gibt einen Minimumsupportwert ein und startet dann die Frequent-Itemset-Analyse.
Maincontrol.java	Die Datenübertragung bzw. Kommunikation zwischen den einzelnen Klassen wird durch diese Klasse verwirklicht.

DBConnection.java	Herstellung der Verbindung mit der Datenbank, Lesen der Daten aus der Datenbanktabelle und Speichern der Ergebnisse in eine andere Datenbanktabelle erfolgt in dieser Klasse.
FPGA.jar	Bei diesem Jar File handelt es sich um den *FPGrowth_Itemsets_with_Strings* Algorithmus, welcher die Frequent-Itemset-Analyse durchführt.
Search.java	Diese Klasse ist für das Suchen und Selektieren der gekoppelten Dateiänderungen und den CommitIDs verantwortlich.
Process.java	Hier werden die gekoppelten Dateiänderungen und die CommitIDs voneinander getrennt, um sie an die unterschiedlichen Views zuzuschicken.
CustomComparator.java	Diese Klasse sortiert die gekoppelten Dateiänderungen entsprechend ihren Supportwerten in absteigender Reihenfolge, so dass dann später in dem Coupled Changes die gekoppelten Dateiänderungen mit den größeren Supportwerten an den ersten Stellen stehen und somit eine höhere Priorität besitzen.
ShowPlugin.java	Diese Klasse ist für die Übermittlung und Rückgabe der Informationen zwischen den Views verantwortlich.
CoupledChanges.java (View)	In diesem View werden die gekoppelten Dateiänderungen angezeigt.
CommitView.java (View)	Diese View listet die CommitIDs auf, welche die gekoppelten Dateiänderungen beinhalten.
CommitMessageView.java (View)	In diesem View werden die Einträge zu dem selektierten CommitID aus dem Commit View angezeigt.

6.4 Implementierung der einzelnen Komponenten

In diesem Abschnitt wird die Implementierung der einzelnen Komponenten des SRM Plug-Ins vorgestellt.

6.4.1 ExecutionView.java

Diese Klasse stellt die Schnittstelle zwischen dem Benutzer und dem SRM Plug-In dar. Mittels einer Benutzeroberfläche kann der Benutzer einen Minimumsupportwert eingeben und durch das Betätigen des „Start Execution“ Buttons die Frequent-Itemset-Analyse starten.

```
1  public class ExecutionView extends ViewPart {
2          private Text minimumsupport;
3          public ExecutionView() {}
4          public void createPartControl(Composite parent) {
5                  parent.setLayout(null);
6                  Label lblMinsupport = new Label(parent, SWT.NONE);
7                  lblMinsupport.setBounds(35, 136, 118, 15);
8                  lblMinsupport.setText("Choose minsupp:");
9                  minimumsupport = new Text(parent, SWT.BORDER);
10                 minimumsupport.setBounds(207, 136, 76, 21);
11                 Button btnRun = new Button(parent, SWT.NONE);
12                 btnRun.addSelectionListener(new SelectionAdapter() {
13                         public void widgetSelected(SelectionEvent e) {
14         Maincontrol.minsupp=Double.parseDouble(minimumsupport.getText());
15                                 Maincontrol.invokeDB();}
16                 });
17                 btnRun.setBounds(123, 202, 158, 25);
18                 btnRun.setText("Start Execution");
19         }
20         public void setFocus() {
21         }}
```

Listing 2: Quellcode ExecutionView.java

Gibt der Benutzer einen Wert in das „Choose minsupp“ Textfeld ein und drückt auf den „Start Execution“ Button, wird dieser Wert (das *minimumsupport* Attribut aus der Zeile 2 in Listing 2) an das *minsupp* Attribut der Klasse Maincontrol.java übergeben (Zeile 14 in Listing 2) und im nachhinein dann die *invokeDB() Methode* der Maincontrol.java Klasse aufgerufen (Zeile 15 in Listing 2).

6.4.2 Maincontrol.java

Wie in der Tabelle 7 aufgelistet, ermöglicht diese Klasse den Datenfluss zwischen den Klassen des SRM Plug-Ins.

```
1  public class Maincontrol {
2  public static double minsupp;
3  public static String outputtablename = "Outputtable";
4  public static String SelectedFile;
5  public static List<List<String>> SelectedFileFieldsOfTheOutputtable;
6  public static List<List<String>> coupledchangesfiles;
7  public static List<String> CommitIDs;
8  public static List<List<String>> commitMessageValue = new ArrayList<>();
9          public static void main(String[] args) {
10 }
11                 public Maincontrol(){
12                 }
13         public static void invokeDB()
14         {
15                 DBConnection.ReadInputTable();
16                 SendMinsuppToTheAlgortithm();
17         }
18                 public static void SendMinsuppToTheAlgortithm() {
19                 double MinimumSupport= minsupp;
20                 FPGrowthAlgorithmus.runAlgorithm(MinimumSupport);
21                 DBConnection.CreateOutputTable();
22                 DBConnection.WriteIntoOutputTable();
23         }
24         public static void invokeThePreparationOfTheCoupledChanges()
25         {
26 Search.searchForTheSelectedFile(SelectedFile, FPGrowthAlgorithmus.output);
27 Collections.sort(SelectedFileFieldsOfTheOutputtable, new CustomComparator());
28 Process.processingTheSelectedFileFields(SelectedFileFieldsOfTheOutputtable);
29 Process.removeRedundanciesOfTheCommidIDs(Maincontrol.CommitIDs);
30                 }}
```

Listing 3: Quellcode Maincontrol.java

Das Listing 3 repräsentiert den Quellcode der Maincontrol.java Klasse. Im Folgenden werden die Attribute und Methoden detaillierter erläutert:

Attribute der Maincontrol.java Klasse:

1. ***minsupp* (Zeile 2 in Listing 3):**
   Dieses Attribut beinhaltet den durch den Benutzer eingegebenen Minimumsupportwert. Dieser Wert wird von der ExecutionView.java Klasse übergeben (Zeile 14 in Listing 2).

2. ***outputtablename* (Zeile 3 in Listing 3):**
   Der Name der Outputtabelle, welche die Ergebnisse der Frequent-Itemset-Analyse beinhalten soll, wird in diesem Attribut festgelegt. Hier ist der Name als „Outputtable“ definiert.

3. ***SelectedFile*** **(Zeile 4 in Listing 3):**
   Der Pfad des durch den Benutzer selektierten Files befindet sich in diesem Attribut. Dieser Wert wird diesem Attribut von der CoupledChanges.java Klasse übergeben (Zeile 1 in Listing 13).

4. ***SelectedFileFieldsOfTheOutputtable*** **(Zeile 5 in Listing 3):**
   Dieses Attribut beinhaltet jene Zeilen der Outputtabelle, welche das Attribut *SelectedFile* enthalten. Das bedeutet, aus der Outputtabelle werden die Zeilen, die das selektierte File beinhalten ausgewählt und diesem Attribut übergeben. Dieser Wert wird von der Klasse Search.java hinzugefügt (Zeile 15 in Listing 27).

5. ***coupledchangesfiles*** **(Zeile 6 in Listing 3):**
   Bei diesem Attribut handelt es sich um eine Teilmenge des Attributs *SelectedFileFieldsOfTheOutputtable* (Zeile 5 in Listing 3). Hier befinden sich nur diejenigen Bereiche des Attributes *SelectedFileFieldsOfTheOutputtable*, welche die gekoppelten Dateiänderungen beinhalten. Dieser Wert wird diesem Attribut von der Klasse Process.java hinzugefügt (Zeile 13 in Listing 28).

6. ***CommitIDs*** **(Zeile 7 in Listing 3):**
   Dieses Attribut ist ebenfalls eine Teilmenge des Attributs *SelectedFileFieldsOfTheOutputtable* (Zeile 5 in Listing 3). Hier befinden sich nur diejenigen Bereiche des Attributes *SelectedFileFieldsOfTheOutputtable*, welche die CommitIDs beinhalten. Der Wert wird diesem Attribut von der Klasse Process.java hinzugefügt (Zeile 12 in Listing 29).

7. ***commitMessageValue*** **(Zeile 8 in Listing 3):**
   Dieses Attribut beinhaltet die Einträge zu einem selektierten CommitID. Das bedeutet, selektiert der Benutzer die CommitID: 125 (siehe Abbildung 31), dann beinhaltet dieses Attribut den Eintrag „*während die anderen Klassen*"(siehe Abbildung 32). Dieses Attribut empfängt diesen Wert von der DBConnection.java Klasse (Zeile 25 in Listing 6).

Methoden der Maincontrol.java Klasse:

Diese Klasse beinhaltet insgesamt drei Methoden. Die ersten zwei Methoden sind für die Durchführung der Frequent-Itemset-Analyse und für das Lesen und Schreiben in Datenbanktabellen verantwortlich, während die dritte Methode für die Ermittlung der gekoppelten Dateiänderungen bestimmt ist.

1. ***invokeDB()* (Zeile 13 in Listing 3):**

   - *DBConnection.ReadInputTable()* (Zeile 15 in Listing 3*):*
     Hier wird die Methode *ReadInputTable()* der DBConnection.java Klasse aufgerufen, um die Daten aus der Inputtabelle zu lesen und sie für die Weiterverarbeitung in einem Array zu speichern.

   - *SendMinsuppToTheAlgortithm()* (Zeile 16 in Listing 3):
     Die Methode *SendMinsuppToTheAlgortithm()* aus der Zeile 18 des Listing 3 wird hier aufgerufen.

2. ***SendMinsuppToTheAlgortithm()* (Zeile 18 in Listing 3):**

   - *Double MinimumSupport=minsupp* (Zeile 19 in Listing 3):
     Der Wert von dem Attribut *minsupp* wird dem *MinimumSupport* Attribut zugewiesen.

   - *FPGrowthAlgorithmus.runAlgorithm(MinimumSupport)* (Zeile 20 in Listing 3):
     Hier wird die *runAlgorithm()* Methode des FPGrowthAlgorithmus mit dem *MinimumSupport* als Inputparameter aufgerufen.

   - *DBConnection.CreateOutputTable()* (Zeile 21 in Listing 3):
     Die Methode *CreateOutputtable()* der DBConnection,java Klasse wird aufgerufen, um die Outputtabelle zu erzeugen.

   - *DBConnection.WriteIntoOutputTable()* (Zeile 22 in Listing 3):
     Diese Methode wird aufgerufen, um die Ergebnisse, welche nach der Frequent-Itemset-Analyse entstehen, in die Outputtabelle zu schreiben.

3. ***invokeThePreparationOfTheCoupledChanges()* (Zeile 24 in Listing 3):**

   - *Search.searchForTheSelectedFile(SelectedFile,FPGrowthAlgorithmus.*
     output) (Zeile 26 in Listing 3):
     Der Aufruf dieser Methode überprüft, ob das durch den Benutzer selektierte File in der Outputtabelle vorhanden ist. Wenn ja, dann werden die entsprechenden Bereiche, welche das selektierte File beinhalten aus der Outputtabelle für die Weiterverarbeitung ausgewählt.

- *Collections.sort(SelectedFileFieldsOfTheOutputtable,**new**CustomComparator())* (Zeile 27 in Listing 3):
  Durch den Aufruf dieser Methode werden die aus der Outputtabelle selektierten Bereiche nach Ihren Supportwerten in absteigender Reihenfolge sortiert.

- *Process.processingTheSelectedFileFields(SelectedFileFieldsOfTheOutputtable)* (Zeile 28 in Listing 3):
  Hier werden die aus der Outputtabelle selektierten Bereiche bearbeitet und in zwei Bereiche unterteilt. Ein Bereich beinhaltet nur die gekoppelten Dateiänderungen, während der andere Bereich die entsprechenden CommitIDs zum Inhalt hat.

- *Process.removeRedundanciesOfTheCommidIDs(Maincontrol.CommitIDs) (Zeile* 29 in Listing 3):
  Hier werden die Redundanzen in dem Attribut *CommitIDs* entfernt.

6.4.3 DBConnection.java

Diese Klasse ist für alle Datenbankangelegenheiten verantwortlich. Im Folgenden werden diese Aufgaben mit entsprechenden Quellcodeabschnitten dargelegt:

1. Verbindung mit der Datenbank herstellen:

Wird der Konstruktor der DBConnection.java Klasse aufgerufen, wird eine Verbindung mit der Datenbank hergestellt. Dazu wird in erster Linie der MYSQL Treiber *(mysql-connector-java-5.1.35-bin.jar)* in das SRM Plug-In hinzufügt. Danach wird dann die Verbindung mit der Datenbank definiert und ausgeführt. Wie es auch in der Zeile 1 in Listing 4 auch zu sehen ist, wird innerhalb der DBConnection.java Klasse die Bibliothek *java.sql.** importiert, um alle notwendigen Werkzeuge zu benutzen.

Weiterhin wurde im Rahmen dieser Diplomarbeit auch der FPGrowthAlgorithmus des FPGA.jar Files in diese Klasse importiert (Zeile 2 in Listing 4), um Daten an den FPGrowthAlgorithmus als Input zu senden und dann auch die Ergebnisse des FPGrowthAlgorithmus zu empfangen. Zwischen den Zeilen 4–10 in Listing 4 werden die für den Verbindungsaufbau benötigten Variablen deklariert und initialisiert.

Nachdem all diese Punkte erledigt sind, werden die Datenbankabfragen in den try/catch-Block (Zeilen 13–29 in Listing 4) eigebettet. In der Zeile 15 in Listing 4 wird als Erstes der MYSQL Treiber geladen. Im Anschluss daran wird in den Zeilen zwischen 16-20 in Listing 4 die Verbindung mit der Datenbank hergestellt. Falls in dem try-Block irgendwelche Fehler

auftreten, dann werden diese durch die entsprechenden catch-Anweisungen abgefangen und dem Benutzer mitgeteilt, um was für einen Fehler es sich dabei handelt. Tritt beispielweise ein Problem beim Laden von dem MYSQL Treiber auf, oder ist der MYSQL Treiber überhaupt nicht vorhanden, dann wird dieses Problem durch die catch-Anweisung in der Zeile 21 in Listing 4 abgefangen und dem Benutzer mitgeteilt, dass der Treiber nicht gefunden werden konnte. Entsprechend zu dem Beispiel von vorhin, kann auch ein Problem mit der Datenbankverbindung auftreten. In so einem Fall wird dann die catch-Anweisung in der Zeile 25 in Listing 4 ausgeführt und dem Benutzer die Nachricht übermittelt, dass die Verbindung mit der Datenbank nicht möglich ist.

```
1  import java.sql.*;
2  import FPGA.FPGrowthAlgorithmus;
3  public class DBConnection {
4          private static Connection conn = null;
5          private static String dbHost = "127.0.0.1";
6          private static String dbPort = "8889";
7          private static String database = "Datenbank_local";
8          private static String dbUser = "root";
9          private static String dbPassword = "root";
10         private static Statement statement;
11                 private DBConnection()
12                 {
13                  try
14                          {
15                          Class.forName("com.mysql.jdbc.Driver");
16                   conn = DriverManager.getConnection("jdbc:mysql://" + dbHost      +
17 ":"
18                      + dbPort + "/" + database + "?" + "user=" + dbUser + "&"
19                      + "password=" + dbPassword);
20                          }
21                  catch (ClassNotFoundException e)
22                          {
23                   System.out.println("Treiber nicht gefunden");
24                          }
25                  catch (SQLException e)
26                          {
27                   System.out.println("Connect nicht moeglich");
28                          }
29                 }
30                 private static Connection getInstance()
31                 {
32                  if(conn == null)
33                   new DBConnection();
34                                          return conn;
35                  }
```

Listing 4: Herstellung der Verbindung mit der Datenbank in DBConnection.java

2. Lesen der Daten aus der Inputtabelle:

Nach dem in dem vorherigen Schritt die Datenbankverbindung hergestellt wurde, findet das Lesen der Daten aus der Inputtabelle statt, welche die für die Frequent-Itemset-Analyse benötigten Informationen beinhaltet. Das Listing 5 zeigt den Quellcodeabschnitt der DBConnection.java Klasse, die die Daten aus der Inputtabelle liest und diese an den FPGrowthAlgorithmus übergibt.

```
1  public static void ReadInputTable() {
2  List<List<String>> res = new ArrayList<>();
3                      conn = getInstance();
4                      if(conn != null){
5                       Statement query = null;
6                       String sql;
7                       ResultSet result = null;
8                       try {
9                                   query = conn.createStatement();
10                                  sql = "SELECT * FROM Inputtable";
11                                  result = query.executeQuery(sql);
12                          } catch (SQLException e1) {
13                                  e1.printStackTrace();}
14                      try {
15                       ResultSetMetaData data = result.getMetaData();
16                       int numcols = data.getColumnCount();
17                       while(result.next()){
18                              List<String> row = new ArrayList<>(numcols);
19                              int i = 1;
20                              while (i <= numcols) {
21                                  if(!result.getString(i).equals("")){
22                                          row.add(result.getString(i));
23                                  }i++; }
24                              res.add(row);}
25                      }
26                      catch (SQLException e)
27                      {
28                       e.printStackTrace();
29                      }
30                      FPGrowthAlgorithmus.input=res;}}
```

Listing 5: Leseoperation auf die Inputtabelle in DBConnection.java

Der Ablauf der Leseoperation wird folgendermaßen durchgeführt:

1. Als Erstes wird eine ArrayList namens *res* erzeugt (Zeile 2 in Listing 5).

2. Danach wird in den Zeilen 9-11 in Listing 5 die SQL-Abfrage definiert und ausgeführt. Der ganze Inhalt der Inputtabelle befindet sich dann in der *result* Variable.

3. Anschließend findet dann die Ausführung des zweiten try Blockes statt (Zeilen 14-25 in Listing 5). Bei dieser Ausführung wird die *result* Variable Zeile für Zeile gelesen, und die Daten in die ArrayList *res* hinzugefügt.

4. Nachdem all die vorherigen Schritte durchlaufen sind, wird dann das Attribut *res* dem *input* Attribut des FPGrowthAlgorithmus übergeben (Zeile 30 in Listing 5).

3. Lesen der Daten aus der Commit Message Tabelle:

Diese Methode der DBConnection.java Klasse ist dafür verantwortlich, die Einträge zu dem durch den Benutzer selektierten CommitID aus der Commit Message Tabelle auszuwählen. Selektiert der Benutzer einen CommitID in dem Commit View, wird diese Methode aufgerufen. Bei diesem Aufruf wird dieser Methode dann das selektierte CommitID als Inputparameter übergeben (Zeile 3 in Listing 23). Nachdem diese Methode den CommitID bekommen hat, führt sie eine Abfrage bei der Commit Message Tabelle aus (Zeilen 8-11 in Listing 6). Bei dieser Abfrage werden aus der Commit Message Tabelle die zutreffenden Einträge des selektierten CommitIDs ausgewählt. Zuletzt werden diese Einträge dem Attribut *commitMessaegValue* der Maincontrol.java Klasse übergeben (Zeile 25 in Listing 6).

```
1  public static void ReadCommitMessageTable(String searchedCommitID)
2                     {
3                             List<List<String>> resCommidId = new ArrayList<>();
4                      conn = getInstance();
5                      if(conn != null){
6                        Statement query;
7                        try {
8                       query = conn.createStatement();
9                       String sql = "SELECT CommitMessage FROM CommitMessageTable "
10                                  + "WHERE CommitId='"+searchedCommitID+"'";
11                        ResultSet result = query.executeQuery(sql);
12                        ResultSetMetaData data = result.getMetaData();
13                        int numcols = data.getColumnCount();
14                        while(result.next()){
15                              List<String> row = new ArrayList<>(numcols);
16                              int i = 1;
17                              while (i <= numcols) {
18                               row.add(result.getString(i++));}
19                              resCommidId.add(row); }
20                       }
21                       catch (SQLException e)
22                       {
23                        e.printStackTrace();
24                       }
25                       Maincontrol.commitMessageValue=resCommidId;}}
```

Listing 6: Leseoperation auf die Commit Message Tabelle in DBConnection.java

4. Erzeugen der Outputtabelle:

Nachdem die Frequent-Itemset-Analyse durch den FPGrowthAlgorithmus ausgeführt wurde, wird zuerst die Outputtabelle erzeugt. Die Zeilen 5-18 in Listing 7 zeigen, wie die Outputtabelle erzeugt wird.

Bevor die Outputtabelle erzeugt wird, wird als Erster überprüft, ob die Outputtabelle mit dem Namen „Outputtable“ in der Datenbank vorhanden ist. Ist es der Fall, dann wird diese Outputtabelle aus der Datenbank entfernt (Zeilen 6-9 in Listing 7). Durch diesen Vorgang wird gewährleistet, dass in der Outputtabelle immer die aktuellen Ergebnisse der Analyse sich befinden.

Da die ersten zwei Spalten der Outputtabelle immer gleich bleiben, werden sie gleich als erstes erzeugt (Zeilen 10-13 in Listing 7). Die restlichen Spalten der Outputtabelle sind immer abhängig von dem längsten Frequent Itemset, welcher von der Ausgabe des FPGrowthAlgorithmuses bestimmt wird. Die Zeilen 14-15 in Listing 7 zeigen eine for Schleife, die solange durchlaufen wird, bis das Ende des längsten Frequent Itemsets erreicht ist. Bei jedem Durchlauf der for Schleife wird dann der Outputtabelle eine Spalte hinzugefügt. Ist beispielsweise die Länge des längsten Frequent Itemsets aus dem FPGrowthAlgorithmus gleich 5, dann werden bei der Outputtabellenerzeugung zu den ersten zwei Spalten, noch weitere fünf Spalten hinzugefügt. Somit hätte dann die Outputtabelle insgesamt sieben Spalten.

```
1  public static void CreateOutputTable() {
2                       conn = getInstance();
3                       if(conn != null)
4                       {
5                               try {
6                                       String myTableName;
7  myTableName = "DROP TABLE IF EXISTS  " + Maincontrol.outputtablename;
8                                       statement = conn.createStatement();
9                                       statement.executeUpdate(myTableName);
10 myTableName ="CREATE TABLE  " + Maincontrol.outputtablename +
11                      "(id INTEGER NOT NULL AUTO_INCREMENT, ";
12                    myTableName+="CommidId VARCHAR(250),";
13                   myTableName+="Support VARCHAR(250),";
14           for(int i=1;i<=FPGrowthAlgorithmus.maxSpalteAnzahl;i++)
15                    myTableName+=  " Item" +i+" VARCHAR(250), ";
16                    myTableName+= " PRIMARY KEY ( id))";
17                              statement = conn.createStatement();
18                              statement.executeUpdate(myTableName); }
19                            catch (SQLException e ) {
20                              e.printStackTrace();} }}
```

Listing 7: Erzeugen der Outputtabelle in DBConnection.java

5. Speichern der Ergebnisse in die Outputtabelle:

Sind die ganzen vorherigen Schritte erfolgreich durchgeführt, erfolgt zum Schluss das Speichern bzw. Schreiben der Ergebnisse des FPGrowthAlgorithmuses in die erzeugte Outputtabelle. Der FPGrowthAlgorithmus schreibt jedes Frequent Itemset in einen Array. Ist der Algorithmus mit der Frequent-Itemset-Analyse fertig, befinden sich alle Frequent Itemets in einem Array in dem FPGrowthAlgorithmus. An dieser Stelle der DBConnection.java Klasse, wird der Array aus dem FPGrowthAlgorithmus herangezogen, um dessen Inhalt in die Outputtabelle hinzuzufügen.

6.4.4 FPGA.jar

Bei diesem FPGA.jar File handelt es sich um den FPGrowthAlgorithmus, welcher die Frequent-Itemset-Analyse durchführt. Allgemein formuliert setzt sich dieser Jar File aus den folgenden drei Klassen: *FPGrowthAlgorithmus.java*, *FPTree_Strings.java* und *FPNode_Strings.java*. Diese drei Klasse wurden ursprünglich aus dem SPMF Data Mining Framework genommen, entsprechend den Anforderungen dieser Diplomarbeit verändert und anschließend in das SRM Plug-In als externer Jar File importiert. Die Veränderungen bezogen sich hauptsächlich nur auf zwei Bereiche der FPGrowthAlgorithmus.java Klasse. Diese zwei Bereiche waren jene Bereiche des Algorithmus, welche für das Lesen der Inputdaten und für das Schreiben der Ergebnisse zuständig waren. Der ursprüngliche Algorithmus liest die Daten von einem Text File, führt eine Frequent-Itemset-Analyse durch und schreibt dann die Ergebnisse in einen anderen Text File. Dies wurde im Rahmen dieser Diplomarbeit so verändert, dass der Algorithmus Daten aus einer Datenbanktabelle lesen, diese Daten analysieren und die Ergebnisse in eine andere Datenbanktabelle schreiben kann.

Das in Listing 5 dargestellter Teilquellcode der DBConnection.java Klasse liest die Daten aus der Datenbanktabelle schreibt sie in einen Array und übergibt diesen Array an das *input* Attribut der FPGrowthAlgorithmus.java Klasse (Zeile 30 in Listing 5). Nach diesem Vorgang stehen dann die Daten der Datenbanktabelle als ein Array in dem Attribut *input* der FPGrowthAlgorithmus.java Klasse (Zeile 2 in Listing 8) und können dann für die Analyse benutzt werden.

Der FPGrowthAlgorithmus.java benötigt für die Durchführung der Frequent-Itemset-Analyse aber auch noch den *MinimumSupport*, welcher von dem Benutzer in dem GUI der ExecutionView.java manuell eingegeben und dem *minsupp* Attribut der Maincontrol.java Klasse übergeben wurde (Zeile 14 in Listing 2). Von dieser Maincontrol.java Klasse aus wird dann die *runAlgorithm()* Methode des FPGrowthAlgorithmus.java mit dem *MinimumSupport* als Inputparameter aufgerufen (Zeile 20 in Listing 3). Dadurch startet der FPGrowthAlgorithmus.java mit dem *MinimumSupport* Parameter (Zeile 9 in Listing 8).

Auf diese Art und Weise wird das Lesen der Daten aus der Datenbanktabelle realisiert und für die Durchführung der Frequent-Itemset-Analyse an die weiteren Methoden und Klassen des FPGA.jar Files übergeben.

```
1  public class FPGrowthAlgorithmus {
2          public static List<List<String>> input;
3          public static List<List<String>> output=new ArrayList<>();
4          public static int maxSpalteAnzahl = 0;
5          public static int transactionCount = 0;
6          public static int itemsetCount;
7          public static int relativeMinsupp;
8          public FPGrowthAlgorithmus(){}
9          public static void runAlgorithm (double MinimumSupport)
10                 {
11                 output=new ArrayList<>();
12                 transactionCount = 0;
13                 maxSpalteAnzahl = 0;
14                         startTimestamp = System.currentTimeMillis();
15                         itemsetCount =0;
16 final Map<String, Integer> mapSupport = new HashMap<String, Integer>();
17 scanDatabaseToDetermineFrequencyOfSingleItems(input,mapSupport);
```

Listing 8: Leseoperation des FPGrowthAlgorithmus.java

Nach der Durchführung der Frequent-Itemset-Analyse entstehen Ergebnisse, die in eine andere Datenbanktabelle geschrieben werden müssen. Des Weiteren sollen in die Outputtabelle mehr Informationen gespeichert werden, als die ursprüngliche Version des FPGrowthAlgorithmus vorsieht.

Sobald ein Frequent Itemset gefunden wird, wird das Frequent Itemset und sein Supportwert der *writeItemsetToOutput()* Methode der FPGrowthAlgorithmus.java Klasse übergeben (Zeile 1 in Listing 9).

Dieses Frequent Itemset wird direkt in einen *buffer* gespeichert und dann mit dem *input* Array der FPGrowthAlgorithmus.java verglichen. Bei diesem Vergleich geht es darum zu überprüfen, in welchen Transaktionen das entsprechende Frequent Itemset sich befindet, um dann diese ermittelten Transaktionen zusammen mit dem Frequent Itemset in das Outputarray zu übertragen. Die Zeilen 3-35 in Listing 9 zeigen den Quellcodeabschnitt, welche das Vergleichen und Speichern der Daten in das Outputarray beschreiben.

Nachdem alle Frequent Itemsets sich in dem Outputarray befinden, werden diese Daten dann in die Outputtabelle geschrieben. Darüberhinaus wird auch noch die Länge des längsten Frequent Itemsets errechnet, um mit Hilfe von diesem Wert die Spaltenanzahl der Outputtabelle zu bestimmen (Zeilen 36-38 in Listing 9).

```
1  private static void writeItemsetToOutput(String [] itemset, int support)
2          {
3                  if(itemset.length>1){
4                          itemsetCount++;
5                          StringBuffer buffer = new StringBuffer();
6                          for(int i=0; i< itemset.length; i++){
7                                  buffer.append(itemset[i]);
8                                  if(i != itemset.length-1){
9                                          buffer.append(' ');}}
10                         buffer.append(' ');
11                         buffer.append(support);
12                         String []  strTemp =buffer.toString().split(" ");
13                         int i,k,j,m,n=0;
14                         StringBuffer bufferWrieter = new StringBuffer();
15                         List<String> wrtStrTemp =new ArrayList<>();
16                         for(i=0;i<input.size();i++)
17                 {                       m=0;    j=0;
18                         for(k=0;k<strTemp.length-1;k++){
19                                 for(j=0;j<input.get(i).size();j++){
20         if(input.get(i).get(j).contains(strTemp[k]))
21                                         {  break;}}
22                         if(j!=input.get(i).size())
23                         {          m++;}}
24                                 if(j==input.get(i).size()&& k==strTemp.length-1)
25                                 { }else if(j!=input.get(i).size()&&    m==strTemp.length-1)
26                         { bufferWrieter.append(input.get(i).get(0));
27                                 n++;
28                                 if(n<support)
29                                 {bufferWrieter.append(':');}
30                         } else if(j!=input.get(i).size()&& m!=strTemp.length-1){}}
31                         wrtStrTemp.add(bufferWrieter.toString());
32                          wrtStrTemp.add(strTemp[strTemp.length-1]);
33                          for(int l=0;l<strTemp.length-1;l++)  {
34                                  wrtStrTemp.add(strTemp[l]); }
35                         output.add(wrtStrTemp);
36                         if (itemset.length>= maxSpalteAnzahl)
37                         {maxSpalteAnzahl =itemset.length;}}
38                     else{}}
```

Listing 9: Schreibeoperation des FPGrowthAlgorithmus.java

6.4.5 CoupledChanges.java

Diese Klasse hat die Aufgabe die gekoppelten Dateiänderungen des in dem Project Explorer selektierten Files anzuzeigen. Gleichzeitig ist aber auch die CommitView.java Klasse von dieser Klasse abhängig. Diese Klasse ist in diesem Zusammenhang der Sender und die CommitView.java Klasse der Empfänger.

Der Gesamtablauf dieser Klasse wird im Folgenden Schritt für Schritt erläutert:

1. Reaktion auf Selektionsänderungen in dem Project Explorer:

Um die gekoppelten Dateiänderungen eines in dem Project Explorer selektierten Files auch anzuzeigen, ist die sofortige Reaktion auf die Selektionsänderungen in dem Project Explorer von entscheidender Bedeutung. Das bedeutet, klickt der Benutzer in dem Project Explorer auf einen File, so muss diese Klasse auf diese Ereignisse reagieren können, um dann entsprechend die richtigen Ergebnisse bezogen auf das selektierte File auch anzeigen zu können.

Der beste Weg, um auf die Selektionsänderungen in dem Project Explorer zu reagieren, ist die Anwendung von Selection Service. Selection Service ermöglichen es den einzelnen Sichten (Views) des Eclipse, auf Selektionsänderungen in dem Workbench-Fenster zu reagieren, ohne dabei direkt mit dem Workbench-Fenster kommunizieren zu müssen. Die Kommunikation erfolgt über Selection Service.

Betrachtet man zum Beispiel in Eclipse den Package Explorer View und die Properties View. Die Abbildung 36 repräsentiert wie die beiden Views miteinander kommunizieren. Klickt der Benutzer auf einen File in dem Package Explorer View, so werden dann die Eigenschaften dieses selektierten Files in dem Properties View angezeigt. Klickt der Benutzer danach auf einen anderen File, also findet eine Selektionsänderung in dem Package Explorer statt, so muss die Properties View sofort auf diese Selektionsänderung reagieren und seinen Inhalt aktualisieren.

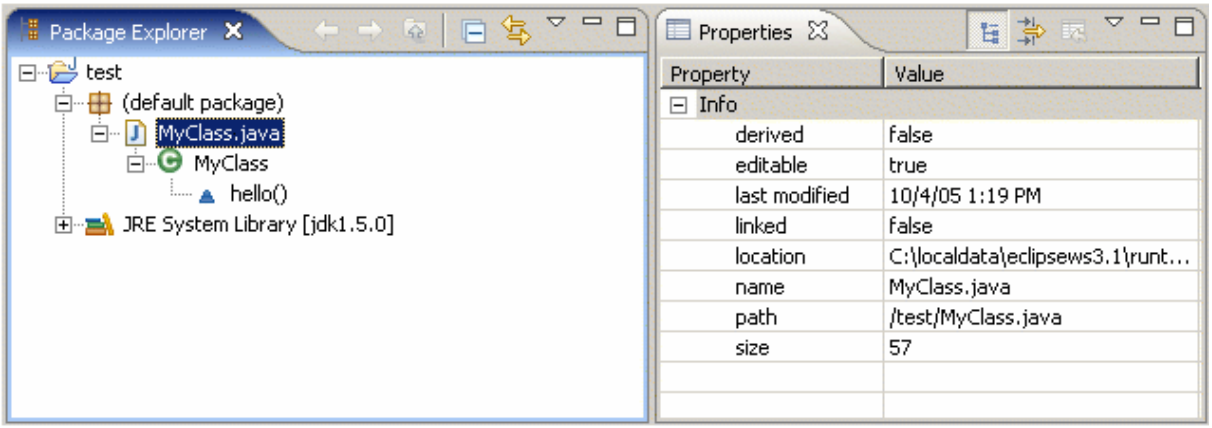

Abbildung 36: Relation zwischen Package Explorer und Properties in Eclipse [36]

Die Abbildung 37 veranschaulicht die Gesamtübersicht über die Anwendung von Selection Service. Der Package Explorer, wo der Benutzer bestimmte Files selektiert, agiert als Selection Provider. D.h., immer wenn der Benutzer einen File in dem Package Explorer

selektiert, wird die aktuelle Selektion dem Selection Service mitgeteilt. Dieses Selection Service beinhaltet diese aktuelle Selektion solange, bis der Benutzer einen anderen File selektiert. Eine andere View auf der rechten Seite des Selection Service, beispielweise die Properties View agiert als Selection Listener. Das bedeutet, er liest die aktuellen Selektionen des Package Explorers aus dem Selection Service.

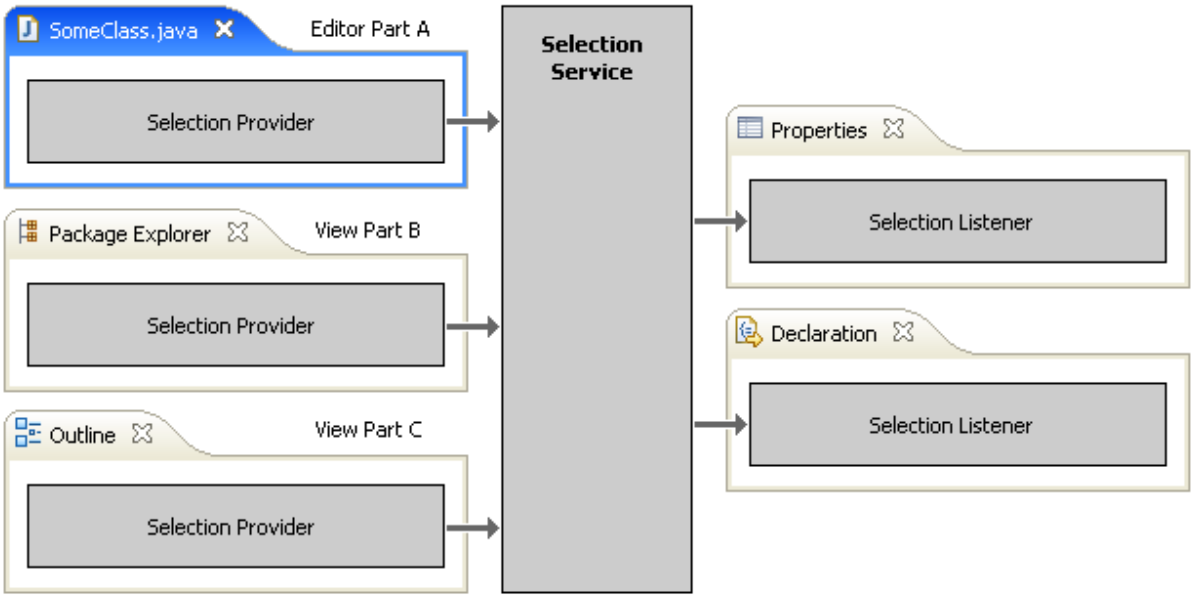

Abbildung 37: Selection Service [36]

Basierend auf diesem Selection Service Prinzip findet dann im SRM Plug-In die Reaktion des Coupled Changes auf die Selektionsänderungen im Project Explorer statt. So wie man es aus der Abbildung 38 entnehmen kann erfolgt die Relation zwischen dem Project Explorer und Coupled Changes via Selection Service. Dabei agiert der Project Explorer als Selection Provider und Coupled Changes als Selection Listener. Dadurch wird eine direkte Relation zwischen dem Project Explorer und dem Coupled Changes vermieden.

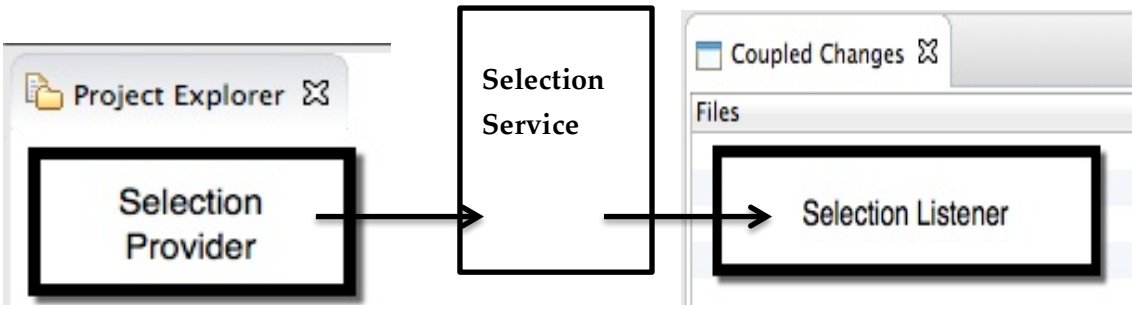

Abbildung 38: Relation zwischen Project Explorer und Coupled Changes

Damit der Coupled Changes auch als Selection Listener agieren kann, ist zuerst eine Registrierung beim Selection Service notwendig. Das Listing 10 zeigt den Quellcodeabschnitt der CoupledChanges.java Klasse, in dem der Selection Listener des Coupled Changes beim Selection Service registriert und im Falle einer Selektion dann die Methode in der Zeile 4 in Listing 10 mit seinen entsprechenden Parametern aufgerufen wird.

```
1  private ISelectionListener listener = new ISelectionListener() {
2  public void selectionChanged(IWorkbenchPart sourcepart, ISelection selection) {
3                  if (sourcepart != CoupledChanges.this) {
4  determineThePathOfTheSelectedFileAndSendIt(sourcepart, selection);}}};
```

Listing 10: Registrierung des Selection Listeners in CoupledChanges.java

2. Pfadbestimmung und Transformation der selektierten Files:

Nachdem durch den vorherigen Schritt gewährleistet wurde, dass die Coupled Changes über alle aktuellen Selektionsänderungen im Project Explorer View informiert wird, erfolgt in diesem Schritt die Bearbeitung des selektierten Files, um dessen Pfad zu bestimmen. Grund dafür ist die Datenstruktur der Files in der Outputtabelle, in der die Files als Pfade repräsentiert sind (siehe Tabelle 5). Um jetzt die in dem Project Explorer durch den Benutzer selektierten Files mit dieser Outputtabelle zu vergleichen und die Ergebnisse dieses Vergleiches anzuzeigen, bedarf es der Ermittlung des Pfades des selektierten Files. D.h., dass der Pfad des in dem Project Explorer selektierten Files mit der Inhaltsstruktur der Outputtabelle übereinstimmen muss, um überhaupt den Vergleich durchführen zu können.

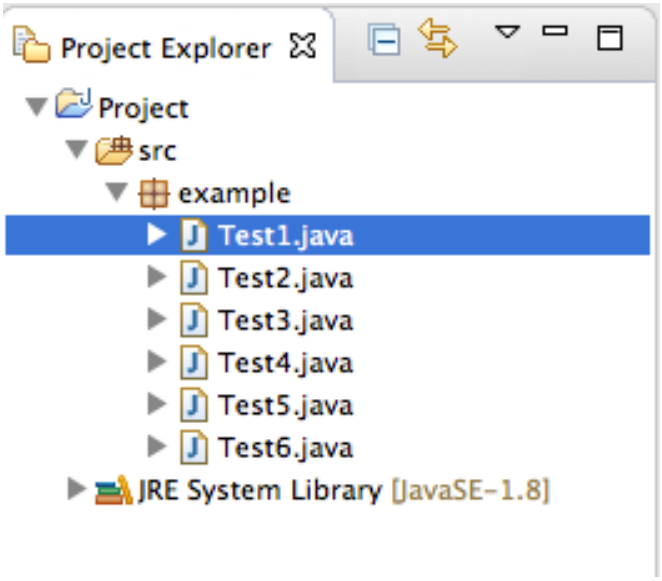

Abbildung 39: Selektion eines Files in dem Project Explorer

Selektiert beispielweise der Benutzer wie es in der Abbildung 39 dargestellt ist, das File *Test1.java*, so muss in der CoupledChanges.java Klasse der Pfad dieses Files ermittelt werden, um den Vergleich mit der Outputtabelle aus der Tabelle 5 durchführen zu können. Denn in der Tabelle 5 liegt der Pfad dieses Files in folgender Form vor: *Project/src/example/Test1.java*.

Für die Ermittlung des Pfades von einem File ist die Anwendung der Interfaces *IStructuredSelection* und *ITreeSelection* erforderlich. Generell existieren zwei grundlegend verschiedene Selektionsarten:

1. Eine Liste von Objekten.
2. Ein Textstück.

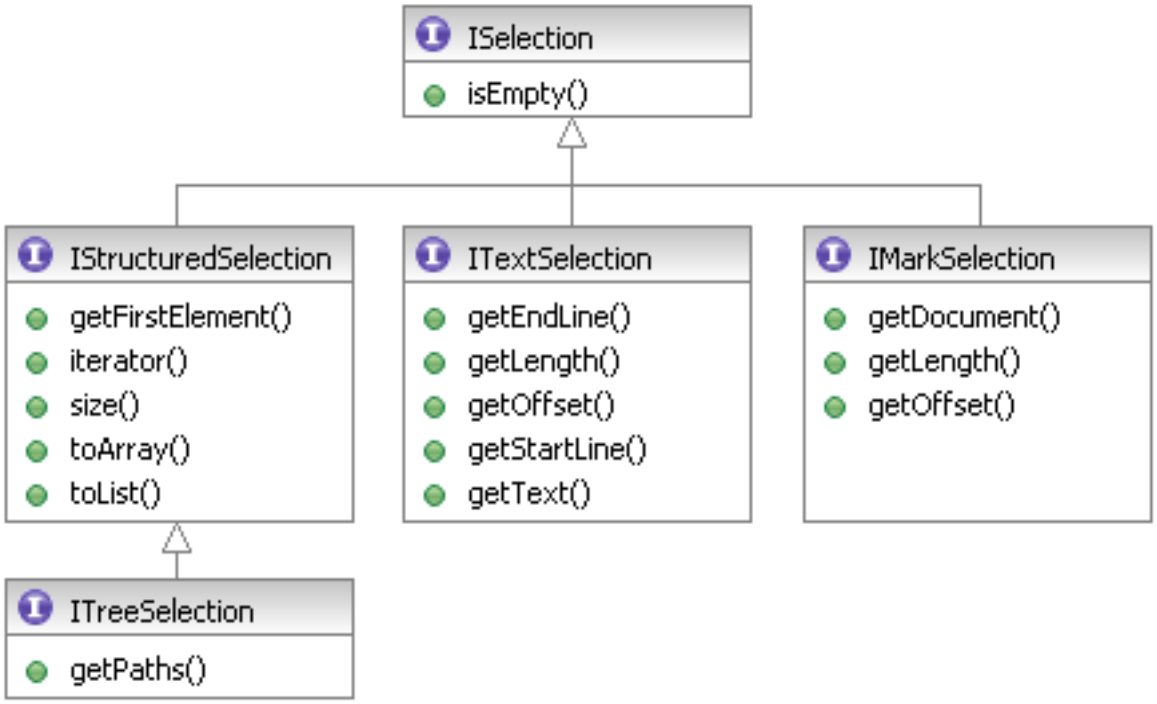

Abbildung 40: Überblick über alle Selektionsarten im Project Explorer [36]

So wie man es aus der Abbildung 40 auch entnehmen kann, existieren drei verschiedene Selektionsinterfaces:

1. ***ITextSelection***:
   Dieses Interface ist für die Repräsentation von textuellen Selektionen zuständig [37].

2. ***IMarkSelection***:
   Handelt es sich bei der Selektion um eine Markierung eines Textstückes, so kommt die I*MarkSelection* zum Einsatz [38].

3. ***IStructuredSelection***:
   Dieses Interface bezieht sich auf eine Liste von Objekten [39].

4. ***ITreeSelection***:
   Bei diesem Interface handelt es sich um eine spezielle Form der *IStructuredSelection*. Dieses Interface zeichnet sich dadurch aus, dass es jedes Objekt in der Liste als einen Pfad beschreibt. Unter einem Pfad versteht man dabei eine Liste von Objekten, welche angefangen von der Wurzel bis hin zum selektierten File alle Files in einem Baum repräsentieren [36].

Für die Ermittlung der Pfade der selektierten Files werden die letzten zwei Interfaces eingesetzt. Die Ermittlung des Pfades erfolgt in zwei Schritten.

1. **Bestimmung des Pfades des selektierten Files:**
   Zu Beginn wird mit Hilfe der Interfaces *IStructuredSelection* und *ITreeSelection* der Pfad des selektierten Files ermittelt. Listing 11 zeigt den Teilquellcodeabschnitt der CoupledChanges.java Klasse, der für die Ermittlung des Pfades des selektierten Files verantwortlich ist.

   Klickt der Benutzer auf einen File im Project Explorer, wird als Erstes in der Zeile 3 in Listing 11 überprüft, ob die Selektion eine Instanz von *IStructuredSelection* ist. Wenn ja, dann wird die aktuellste Selektion in dem Project Explorer gelesen und in die Variable *selection1* übertragen.

   Im Anschluss daran wird im nächsten Schritt überprüft, ob *selection1* eine Instanz von *ITreeSelection* ist (Zeile 7 in Listing 11). Wenn es der Fall ist, dann werden in den darauffolgenden Schritten, also in den Zeilen 10-23 in Listing 11 der Pfad des aktuell selektierten Files bestimmt und der Variablen *fileName* übergeben.

   Somit wird bei jeder Fileselektion der entsprechende Pfad des selektierten Files bestimmt und auf die gleiche Art und Weise der Variablen *fileName* übergeben.

```
1  public void determineThePathOfTheSelectedFileAndSendIt(IWorkbenchPart
2  sourcepart, ISelection selection) {
3          if (selection instanceof IStructuredSelection) {
4  window = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
5                              activePage = window.getActivePage();
6                              ISelection selection1 = activePage.getSelection();
7                               if(selection1 instanceof ITreeSelection)
8                               {
9              TreeSelection treeSelection = (TreeSelection) selection1;
10                            TreePath[] treePaths = treeSelection.getPaths();
11                            TreePath treePath = treePaths[0];
12                             Object firstSegmentObj = treePath.getFirstSegment();
13                           theProject = (IProject) ((IAdaptable)
14 firstSegmentObj).getAdapter(IProject.class);
15                             Object lastSegmentObj = treePath.getLastSegment();
16                             theResource = (IResource) ((IAdaptable)
17 lastSegmentObj).getAdapter(IResource.class);
18                             theFile = (IFile) ((IAdaptable)
19 lastSegmentObj).getAdapter(IFile.class);
20                             workspaceName =
21 theResource.getWorkspace().getRoot().getLocation().toOSString();
22                             projectName =theProject.getName();
23                             fileName = theResource.getFullPath().toOSString();
```

Listing 11: Pfadermittlung des selektierten Files in CoupledChanges.java

2. **Transformation des Pfades des selektierten Files:**
   In diesem Schritt wird der im vorherigen Schritt ermittelte Pfad transformiert, so dass es mit der Datenstruktur der Outputtabelle (siehe Tabelle 5) übereinstimmt.

   An dieser Stelle wird hier noch einmal auf das Beispiel vom vorherigen Schritt eingegangen (Abbildung 39). Der Pfad des durch den Benutzer in dem Project Explorer selektierten Files *Test1.java* befindet sich momentan in der Variablen *fileName* in folgender Form: */Project/src/example/Test1.java*.

So wie man es aus der Outputtabelle (siehe Tabelle 5) auch entnehmen kann, liegt der Pfad des Files *Test1.java* als *Project/src/example/Test1.java* vor. Das bedeutet, der Pfad aus der Variable *fileName* muss umgewandelt und der Datenstruktur aus der Outputtabelle angepasst werden, um den Vergleich durchzuführen.

Das Listing 12 repräsentiert jenen Bereich des Quellcodes der CoupledChanges.java Klasse, welcher für die Pfadtransformation der selektierten Files verantwortlich ist. Es verdeutlich wie der Pfad der Variablen *fileName* transformiert und anschließend in die Variable *buffTemp* gespeichert wird.

Nach der Durchführung der Pfadtransformation befindet sich dann der aktuell transformierte Pfad des selektierten Files *Test1.java* in der Variable *buffTemp* in folgender Form: *Project/src/example/Test1.java*.

```
1   StringBuffer buffTemp = new StringBuffer();
2                           char[] CUT_POINTS = {'\ \ '};
3                           char [] probe=fileName.toCharArray();
4                           for(int i=1;i<probe.length;i++)
5                           {
6                               if(probe[i]==CUT_POINTS[0])
7                               {
8                                       buffTemp.append("/");
9
10                              }else
11                              {
12                                      buffTemp.append(probe[i]);
13                          } }
```

Listing 12: Pfadtransformation in CoupledChanges.java

Nach diesem letzten Schritt wird der Pfad des aktuell selektierten Files an die Variable *SelectedFile* der Maincontrol.java Klasse übergeben und die Methode, welche den Vergleich und alle weiteren Bearbeitungen durchführen soll aufgerufen. Das Listing 13 zeigt die Übergabe des Pfades und den Methodenaufruf.

```
1   Maincontrol.SelectedFile=buffTemp.toString();
2   Maincontrol.invokeThePreparationOfTheCoupledChanges();
```

Listing 13: Filepfadübergabe und Methodenaufruf in CoupledChanges.java

3. Initialisierung der Coupled Changes und Commit View:

Die vorherigen zwei Schritte waren für die Reaktion auf die Selektionsänderungen aus dem Project Explorer und für die Ermittlung und Übergabe der Filepfade verantwortlich.

Dieser Schritt ist für das Anzeigen der eigenen Ergebnisse in der eigenen Klasse (d.h., CoupledChanges.java Klasse) und die Weiterleitung der anderen Daten an die CommitView.java Klasse bestimmt. Er ist für Initialisierung der eigenen Klasse und der CommitView.java Klasse verantwortlich und wird bei jedem Ausführen der CoupledChanges.java Klasse durchlaufen. Das Listing 14 zeigt den Quellcodeabschnitt der CoupledChanges.java Klasse, welcher die Initialisierung ermöglicht.

Selektiert beispielweise der Benutzer einen File in dem Project Explorer, so wird nach den ganzen Bearbeitungen, dieser Teil des Quellcodes aufgerufen und die entsprechenden Klassen initialisiert. Entscheidet sich der Benutzer danach für einen anderen File als dem

vorherigen, so wird wieder dieser Bereich ausgeführt und die entsprechenden Klassen wieder initialisiert. Dadurch wird sichergestellt, dass die entsprechenden Klassen, die die Ergebnisse anzeigen sollen (in diesem Fall die CoupledChanges.java und CommitView.java Klassen), immer die richtigen Ergebnisse des entsprechenden aktuell selektierten Files beinhalten.

D.h., die alten Werte werden immer durch die neuen Werte ersetzt. Die Zeile 1 in Listing 14 definiert eine Variable namens *viewvalue* und initialisiert es mit dem Wert 1. Dadurch wird die sichere Ausführung der nachfolgenden if-Anweisung gewährleistet (Zeile 2 in Listing 14). In der if-Anweisung wird dann die Initialisierung der Coupled Changes durch die Ausführung des Quellcodes in den Zeilen 4-6 in Listing 14 und die Initialisierung der Commit View durch die Ausführung des Quellcodes in den Zeilen 8-10 in Listing 14 verwirklicht.

```
1   viewvalue=1;
2   if(viewvalue!=0)
3    {
4   properties.put("file", showPlgn.Control());
5   Event event = new Event("viewcommunicationfile/syncEvent", properties);
6   eventAdmin.sendEvent(event);
7
8   properties.put("commidid", showPlgn.Control());
9   Event event1 = new Event("viewcommunicationcommidid/syncEvent", properties);
10  eventAdmin.sendEvent(event1);
11                                          }
```

Listing 14: Initialisierung Coupled Changes und Commit View in CoupledChanges.java

4.Senden und Empfangen von Daten:

Nach der Durchführung des Vergleiches von dem selektierten File mit der Outputtabelle werden mehrere Informationen erzeugt, die in unterschiedlichen Views angezeigt werden müssen. Um diese Informationen in ihren Views anzuzeigen, müssen diese erst diese Informationen von anderen Views empfangen. Das bedeutet, die Informationen werden von einem View gesendet und von einem anderen View oder aber auch von dem gleichen View wieder empfangen und angezeigt. Die CoupledChanges.java Klasse ist für das Senden von Informationen über die gekoppelten Dateiänderungen und die CommitIDs, welche die gekoppelten Dateiänderungen beinhalten, verantwortlich.

Senden und Empfangen von gekoppelten Dateiänderungen oder Fehlermeldung:

Im Falle von Informationen über gekoppelte Dateiänderungen agiert die CoupledChanges.java Klasse sowohl als Sender als auch Empfänger. Die gekoppelten Dateiänderungen, die durch ihn gesendet werden, werden später in einer anderen Methode der gleichen Klasse wieder empfangen und dann angezeigt. Dabei werden zwei Arten von Informationen gesendet und später wieder empfangen.

Die eine Information tritt immer dann auf, wenn das selektierte File nicht in der Outputtabelle vorhanden ist. Klickt der Benutzer auf einen File in dem Project Explorer, welcher nicht in der Outputtabelle vorhanden ist, so wird eine Fehlermeldung gesendet, die die folgende Nachricht beinhaltet: „The selected File does not exist in the Ouputtable“ („das selektierte File existiert nicht in der Outputtabelle“). Das Listing 15 zeigt jenen Bereich der CoupledChanges.java Klasse, der für das Senden der Fehlermeldung zuständig ist.

```
1  if(Maincontrol.coupledchangesfiles.size()==0)
2                                          {
3  viewvalue=0;
4  properties.put("file", showPlgn.showFehler("fehler"));
5  Event event = new Event("viewcommunicationfile/syncEvent", properties);
6  eventAdmin.sendEvent(event);
7                                          }
```

Listing 15: Senden von Fehlerinformationen in CoupledChanges.java

Bei der Durchführung des Quellcodes in Listing 15 wird als Erstes überprüft, ob das Attribut *coupledchangesfiles* der Klasse Maincontrol.java leer ist (Zeile 1 in Listing 15). Das Array *coupledchangesfiles* beinhaltet alle gekoppelten Dateiänderungen des selektierten Files. Ist dieses Array leer, dann folgt daraus, dass das in dem Project Explorer selektierte File nicht in der Outputtabelle vorhanden ist. In so einem Fall wird dann der *properties* Variablen der CoupledChanges.java Klasse die Fehlermeldung aus der ShowPlugin.java Klasse hinzugefügt.

Das Listing 16 zeigt jenen Teil der ShowPlugin.java Klasse, welcher die Fehlermeldung erzeugt und zurückgibt. Beim Aufruf der Methode *showFehler()* aus der ShowPlugin.java Klasse wird die Fehlermeldung „The selected File does not exist in the Outputtable“ erzeugt und der Variablen *f* übergeben (Zeile 3 in Listing 16). Anschließend wird die Variable *f* zurückgegeben (Zeile 4 in Listing 16).

```
1  public String showFehler(String f)
2          {
3  f="The selected File does not exist in the Outputtable";
4  return f;
5          }
```

Listing 16: Erzeugung und Rückgabe der Fehlermeldung in ShowPlugin.java

Wieder zurück in Listing 15 wird dann diese Fehlermeldung, die sich in der Variable *properties* befindet in den darauffolgenden Zeilen gesendet (Zeilen zwischen 5-6 in Listing 15).

Ist das Attribut *coupledchangesfiles* der Maincontrol.java Klasse aber nicht leer, also ist das selektierte File in der Outputtabelle vorhanden, so wird der Inhalt dieses Attributes gesendet. Das Listing 17 repräsentiert den Quellcodeabschnitt der CoupledChanges.java Klasse, der die gekoppelten Dateiänderungen sendet.

```
1  for(int s=0;s<Maincontrol.coupledchangesfiles.size();s++)
2  {
3  viewvalue=0;
4  properties.put("file", showPlgn.showFile("file",s));
5  Event event = new Event("viewcommunicationfile/syncEvent", properties);
6  eventAdmin.sendEvent(event);
7                                  }
```

Listing 17: Senden von gekoppelten Dateiänderungen in CoupledChanges.java

In der Zeile 1 in Listing 17 ist eine for Schleife definiert. Diese for Schleife geht alle gekoppelten Dateiänderungen einzeln durch. Danach wird der *properties* Variablen die gekoppelten Dateiänderungen von der ShowPlugin.java Klasse übergeben (Zeile 4 in Listing 17). Die Methode *showFile()* der ShowPlugin.java Klasse weist die gekoppelten Dateiänderungen einer Variable zu und gibt diese Variable zurück. Das Listing 18 zeigt den Teilquellcode der ShowPlugin.java Klasse, welcher die gekoppelten Dateiänderungen durchläuft und sie einer Variablen zuordnet.

```
1  public String showFile(String a,int i)
2          {
3  a= Maincontrol.coupledchangesfiles.get(i).toString();
4  return a;
5          }
```

Listing 18: Übermittlung und Rückgabe der gekoppelten Dateiänderungen in ShowPlugin.java

Nach diesem Durchlauf werden diese gekoppelten Dateiänderungen, die sich in der Variablen *properties* befinden, gesendet (Zeilen 5-6 in Listing 17). Somit wurden durch die beschriebenen Schritte die Informationen über die gekoppelten Dateiänderungen übermittelt.

Je nachdem welche Information gesendet ist, wird dann auch die entsprechende Information in diesem View angezeigt. Handelt sich bei der gesendeten Information um die gekoppelten Dateiänderungen, werden dementsprechend diese Informationen auch angezeigt. Wird im Gegensatz dazu eine Fehlermeldung erzeugt und gesendet, wird diese Fehlermeldung in dem View repräsentiert. Das Listing 19 zeigt wie die Informationen empfangen und angezeigt werden.

```
1  public void handleEvent(final Event event)
2  {
3  if(CoupledChanges.viewvalue!=0)
4  {viewer.getTable().removeAll();}
5  else{
6  if( parent.getDisplay().getThread() == Thread.currentThread() ) {
7                          viewer.add(event.getProperty("file"));
8                            } else {
9                              parent.getDisplay().syncExec(new Runnable() {
10                               public void run() {
11                                 viewer.add(event.getProperty("file"));
12                               } }); }}}
```

Listing 19: Empfangen und Anzeigen von Informationen in CoupledChanges.java

Senden von CommitIDs:

Eine weitere Aufgabe der CoupledChanges.java Klasse ist zusätzlich noch das Senden von den CommitIDs. Das Listing 20 zeigt den Quellcodeabschnitt der CoupledChanges.java Klasse, der für das Senden von CommitIDs verantwortlich ist.

```
1  for(int s=0;s<Maincontrol.CommitIDs.size();s++)
2  {
3  viewvalue=0;
4  properties.put("commidid", showPlgn.showCommidID("commidid",s));
5  Event event = new Event("viewcommunicationcommidid/syncEvent", properties
6  eventAdmin.sendEvent(event);}
```

Listing 20: Senden von CommitIDs in CoupledChanges.java

Beim Senden von den CommitIDs wird ähnlich vorgegangen wie beim Senden von gekoppelten Dateiänderungen oder Fehlermeldung. Die in der Zeile 1 in Listing 20 definierte for-Schleife durchläuft alle CommitIDs in der Variable *CommitIDs*. Entsprechend werden

dann die CommitIDs aus der *showCommitID()* Methode der ShowPlugin.java Klasse der *properties* Variable übergeben (Zeile 4 in Listing 20). Das Listing 21 zeigt den Teilquellcode der ShowPlugin.java Klasse, welcher die CommitIDs durchläuft und diese einer Variablen zuordnet.

```
1  public String showCommidID(String a,int i)
2          {
3                  a= Maincontrol.CommitIDs.get(i).toString();
4                  return a;
5          }
```

Listing 21: Übermittlung und Rückgabe der CommitIDs in ShowPlugin.java

Ist dieser Durchlauf abgeschlossen, dann wird die Variable *properties*, welche die CommitIDs beinhaltet, gesendet (Zeilen 5-6 in Listing 20).

6.4.6 CommitView.java

Die CommitView.java Klasse agiert sowohl als eine Empfängerklasse als auch eine Senderklasse, wobei die Reihenfolge zuerst Empfangen und dann Senden ist. Die erste Aufgabe von dieser Klasse ist das Empfangen und Anzeigen der CommitIDs, die von der CoupledChanges.java gesendet werden. Die CommiIDs werden in einem Combobox angezeigt (Abbildung 31). Das Listing 22 zeigt wie die CommitIDs empfangen und angezeigt werden.

```
1  public void handleEvent(final Event event) {
2                                  if(CoupledChanges.viewvalue!=0)
3                                  {comboViewer.getCombo().removeAll();}
4                                  else{
5                  if( parent.getDisplay().getThread() == Thread.currentThread() ) {
6                    comboViewer.add(event.getProperty("commidid"));
7                      } else {
8                        parent.getDisplay().syncExec(new Runnable() {
9                          public void run() {
10                           comboViewer.add(event.getProperty("commidid")); }});
11                     }
12                                 }
```

Listing 22: Empfangen und Anzeigen von CommitIDs in CommitView.java

Nachdem die CommitIDs empfangen und in dem Combobox aufgelistet sind, stehen sie dem Benutzer als Auswahl zur Verfügung. Somit hat der Benutzer die Möglichkeit einen von diesen CommitIDs auszuwählen, um sich dann die entsprechenden Einträge zu dem

selektierten CommitID aus der Commit Message Tabelle in dem Commit Message View anzeigen zu lassen.

Wählt beispielweise der Benutzer die CommitID: 125 in dem Commit View (siehe Abbildung 31), wird dieses selektierte CommitID mit den Einträgen in der Commit Message Tabelle (siehe Tabelle 6) verglichen und anschließend der entsprechende Eintrag: „*während die anderen Klassen*“ in dem Commit Message View dem Benutzer vorgestellt.

Dieser Vorgang angefangen von der Selektion der CommitIDs in dem Combobox, bis hin zum Senden der Einträge werden in den folgenden zwei Schritten anhand von Quellcodes näher in Betracht genommen.

1. **Selektion von CommitIDs und Vergleich mit der Commit Message Tabelle:**
   Aus dem Listing 23 sind die Selektion und der Vergleich der CommitIDs mit der Commit Message Tabelle zu entnehmen.

```
1  combo.addSelectionListener(new SelectionListener() {
2          public void widgetSelected(SelectionEvent e) {
3          DBConnection.ReadCommitMessageTable(combo.getText());
```

Listing 23: Selektion von CommitIDs und Vergleich mit der Commit Message Tabelle in CommitView.java

Nachdem in dem Quellcode von Listing 22 dem Combobox die CommitIDs hinzugefügt wurden, wird in diesem Quellcodeabschnitt der CommitView.java Klasse dem Combobox ein SelelectionListener hinzugefügt, um auf die Selektion der CommitIDs durch die Benutzer zu reagieren (Zeile 1 in Listing 23).

Entscheidet der Benutzer sich für ein CommitID und selektiert dieses, dann erfolgt der Methodenaufruf in der Zeile 3 in Listing 23. In dieser Methode wird die *ReadCommitMessageTable()* Methode der DBConnection.java Klasse mit dem selektierten CommitID als Inputparameter aufgerufen. Dort erfolgt dann der Vergleich des selektierten CommitIDs mit der Commit Message Tabelle.

2. **Senden von den Einträgen:**
   Nachdem der Vergleich mit der Commit Message Tabelle stattgefunden hat, erfolgt das Senden dieser Einträge an die Commit Message View. Das Listing 24 zeigt, wie die entsprechenden Einträge des selektierten CommitIDs gesendet werden.

```
1  commitViewValue=0;
2  properties.put("commididmesssage",
3  showPlgn.showCommiIdMessage("commididmessage"));
4  Event event = new Event("viewcommunicationcommididmessage/syncEvent",
5  properties);
6  eventAdmin.sendEvent(event);
```

Listing 24: Senden von Einträgen in CommitView.java

Beim Durchlauf von diesem Quellcodeabschnitt der CommitView.java Klasse wird in erster Linie dem *properties* Variablen die Einträge aus der *showCommitIDMessage()* Methode der ShowPlugin.java Klasse hinzugefügt (Zeilen 2-3 in Listing 24). Das Listing 25 zeigt den Quellcodeabschnitt der ShowPlugin.java Klasse, welcher die Einträge einem Variablen zuordnet.

```
1  public String showCommiIdMessage(String a)
2          {
3                  a=Maincontrol.commitMessageValue.toString();
4                  return a;
5          }
```

Listing 25: Übermittlung und Rückgabe von den Einträgen in ShowPlugin.java

Abschließend wird die Variable *properties* gesendet (Zeilen 4-6 in Listing 24).

6.4.7 CommitMessageView.java

Am Ende steht die Commit Message View, die für das Empfangen und Repräsentieren von den Einträgen zuständig ist. Dementsprechend agiert diese Klasse nur als Empfängerklasse. Das Listing 26 verdeutlicht, wie diese Klasse die Einträge empfängt und präsentiert.

```
1  public void handleEvent(final Event event) {
2                          if(CommitView.commitViewValue!=0)
3                          {viewer.getTable().removeAll();}else{
4                     if( parent.getDisplay().getThread() == Thread.currentThread() ) {
5                   viewer.add(event.getProperty("commididmessage"));
6                  } else {
7                   parent.getDisplay().syncExec(new Runnable() {
8                   public void run() {
9                    viewer.add(event.getProperty("commididmessage"));}}); }} }
```

Listing 26: Empfangen und Anzeigen von den Einträgen in CommitMessageView.java

6.4.8 Search.java

So wie in der Tabelle 7 auch beschrieben, ist diese Klasse für das Suchen und Selektieren der gekoppelten Dateiänderungen und den CommitIDs aus der Outputtabelle zuständig. Diese Klasse besteht aus einer Methode, die im Folgenden erläutert wird.

1. ***searchForTheSelectedFile:***

   In dieser Methode wird in der Outputtabelle nach dem selektierten File gesucht. Existiert das selektierte File in der Outputtabelle, so werden dann die entsprechenden Bereiche der Outputtabelle ausgewählt und anschließend der Variablen *SelectedFileFieldsOfTheOutputtable* der Maincontrol.java Klasse übergeben (Zeile 15 in Listing 27).

```
1  public static void searchForTheSelectedFile(String geSuchtstr ,
2  List<List<String>> gesuchtWriteStr) {
3                  int i,j,m;
4                  for(i=0;i<gesuchtWriteStr.size();i++)
5          {
6                          for(j=2;j<gesuchtWriteStr.get(i).size();j++)
7                          {
8                          if(gesuchtWriteStr.get(i).get(j).equals(geSuchtstr))
9                                  {   break;}}
10                         if(j==gesuchtWriteStr.get(i).size()){ }
11                         else{
12                 List<String> wrtgesuchtStrTemp =new ArrayList<>();
13                 for(m=0;m<gesuchtWriteStr.get(i).size();m++)
14                 {wrtgesuchtStrTemp.add(gesuchtWriteStr.get(i).get(m));}
15 Maincontrol.SelectedFileFieldsOfTheOutputtable.add(wrtgesuchtStrTemp);}
16         }}
```

Listing 27: Suche nach dem selektierten File in der Outputtabelle in Search.java

6.4.9 Process.java

In dieser Klasse findet die Trennung der gekoppelten Dateiänderungen von den CommitIDs statt, um sie dann an die entsprechenden Views zuzuschicken. Folglich werden die gekoppelten Dateiänderungen an die Coupled Changes und die CommitIDs an die Commit View gesendet. Die Process.java Klasse besteht aus drei Methoden:

1. ***processingTheSelectedFileFields*:**

   Die Aufgabe dieser Methode ist es die Variable *SelectedFileFieldsOfTheOutputtable* so zu bearbeiten, dass die gekoppelten Dateiänderungen sich in einer Variable und die CommitIDs sich in einer anderen Variable befinden. Das Listing 28 zeigt, wie die Verarbeitung und Übergabe in dem Quellcode der Process.java Klasse stattfindet.

Als Erstes wird die Variable *SelectedFileFieldsOfTheOutputtable* als *plugInWriteStr* Variable der *processingTheSelectedFileFields()* Methode als Inputparameter übergeben (Zeile 1-2 in Listing 28).

Danach wird der Inhalt der *plugInWriteStr* Variable in der for-Schleife einzelnen durchlaufen (Zeile 5 in Listing 28). Bei jedem Durchlauf werden dann die von der *plugInWriteStr* Variable getrennten gekoppelten Dateiänderungen der *coupledchangesfiles* Variable der Maincontrol.java Klasse hinzugefügt (Zeile 13 in Listing 28). Die entkoppelten *CommitIDs* hingegen werden der *splitCommitIDs()* Methode der Process.java Klasse zur Weiterverarbeitung als Parameter übergeben (Zeile 15 in Listung 28).

```
1  public static void processingTheSelectedFileFields(List<List<String>>
2  plugInWriteStr)
3          {
4                  int i,j;
5                  for(i=0;i<plugInWriteStr.size();i++)
6                  {
7          List<String> plugInwrtStrTempFile =new ArrayList<>();
8          List<String> plugInwrtStrTempCommidiD =new ArrayList<>();
9                          for(j=2;j<plugInWriteStr.get(i).size();j++)
10                         {
11         plugInwrtStrTempFile.add(plugInWriteStr.get(i).get(j));
12                         }
13         Maincontrol.coupledchangesfiles.add(plugInwrtStrTempFile);
14         plugInwrtStrTempCommidiD.add(plugInWriteStr.get(i).get(0));
15         splitCommiDs(plugInwrtStrTempCommidiD);}
16         }
```

Listing 28: Entkopplung der gekoppelten Dateiänderungen von den CommitIDs in Process.java

2. ***splitCommitIDs*:**

In der CommitIDs Spalte der Tabelle 5 ist zu erkennen, dass die CommitIDs durch das Zeichen „:“ voneinander getrennt sind. Nach der Durchführung der *processingTheSelectedFileFields()* Methode der Process.java Klasse, befinden sich die CommitIDs genau in derselben Form wie sie in der Tabelle 5 zu sehen sind. Damit aber diese CommitIDs in der gewünschten Form in dem Commit View anzuzeigen (siehe Abbildung 31), bedarf es einer weiteren Verarbeitung von diesen CommitIDs. Aus diesem Grund erfolgt in dieser Methode die Separierung und Transformation der CommitIDs, welche in dem Listing 29 sehr anschaulich dargestellt sind. Die CommitIDs werden dann zum Schluss der Variable *CommitIDs* der Maincontrol.java Klasse hinzugefügt (Zeile 12 in Listing 29).

```
1   public static void splitCommiDs(List<String> CommidiD)
2           {
3                   StringBuffer buffer = new StringBuffer();
4                   for(int i=0; i< CommidiD.size(); i++){
5                           buffer.append(CommidiD.get(i));
6                           if(i != CommidiD.size()-1){
7                                   buffer.append(':');}}
8                   buffer.append(':');
9           String []  plugInstrTempCommidId =buffer.toString().split(":");
10                  for(int j=0;j<plugInstrTempCommidId.length;j++)
11                  {
12                  Maincontrol.CommitIDs.add(plugInstrTempCommidId[j]);
13                  }}
```

Listing 29: Separierung und Transformation der CommitIDs in Process.java

3. ***removeRedundanciesOfTheCommidIDs*:**

Diese Methode wird in der Maincontrol.java Klasse aufgerufen (Zeile 29 in Listing 3). Der Zweck dieser Methode ist es redundante CommitIDs aus der Variable *CommitIDs* zu entfernen. Nachdem durch die Methode *splitCommitIDs* die CommitIDs separiert und transformiert wurden, stehen sie zwar in gewünschter Form, weisen aber Redundanzen auf. Dies führt dazu, dass die identischen CommitIDs mehrmals in dem Commit View auftreten. Um das Problem zu beheben und somit die CommitIDs in exakt der gewünschten Form, wie sie in der Abbildung 31 aufgelistet sind darstellen zu können, ist die Entfernung von redundanten CommitIDs aus dem Variablen *CommitIDs* der Maincontrol.java Klasse zwingend erforderlich. Das Listing 30 veranschaulicht das Entfernen der redundanten CommitIDs.

```
1   public static void removeRedundanciesOfTheCommidIDs(List<String>
2   sortCommidID)
3           {
4                   Collections.sort(sortCommidID);
5           int j,i;
6                   for (i=0;i<sortCommidID.size();i++)
7                   {
8                           for(j=i;j<sortCommidID.size();j++)
9                           {
10  if((sortCommidID.get(i).contains(sortCommidID.get(j))&&i!=j))
11                                  {
12                                          sortCommidID.remove(j);
13                                          j--;} } }
14          }
```

Listing 30: Entfernen von redundanten CommitIDs in Process.java

7 Vergleich von Sequential Pattern Mining und Frequent Itemset Mining

In diesem Kapitel findet der Vergleich zwischen den Algorithmen Sequential Pattern Mining und Frequent Itemset Mining statt. Zuerst erfolgt eine Einleitung in dieses Themengebiet. Im Anschluss daran werden der Aufbau und die Funktionsweise dieser Algorithmen dargestellt und eine tabellarische Gegenüberstellung repräsentiert. Zum Schluss wird dann dieses Kapitel mit dem Fazit des Vergleiches abgeschlossen.

7.1 Einleitung

So wie es in Kapitel 2 auch sehr ausführlich erläutert wurde, ist das Ziel von Data Mining die Wissensextraktion und die Entdeckung von interessanten Mustern und Charakteristiken, welche in den Datenbanken nicht explizit repräsentiert werden.

Für die Durchführung von Data Mining existieren mehrere Techniken. Eine dieser Techniken ist die Frequent Itemset Mining, die im Rahmen dieser Diplomarbeit für die Analyse der Datenbanken angewendet wurde. In dieser Technik geht es hauptsächlich darum, häufige Itemmengen (Frequent Itemsets) aus Datenbanken zu entdecken, wobei es sich bei den Datenbanken um Transaktionsdatenbanken handelt, die aus einer Menge von Transaktionen bestehen. Jede Transaktion enthält wiederum eine Menge an Items.

In vielen Applikationen liegen die Daten auch in sequenzieller Form vor, was dementsprechend den Einsatz der Data Mining Technik Sequential Pattern Mining für die Verarbeitung dieser sequenziellen Daten erforderlich macht. Das Ziel von Sequential Pattern Mining ist die Entdeckung von häufig auftretenden sequenziellen Mustern (Frequent Sequential Patterns) aus den Sequenzdatenbanken, die sich aus einer Menge von Sequenzen zusammensetzen. Jede Sequenz hingegen besteht aus einer Liste von Itemmengen (Itemsets).

7.2 Aufbau und Funktionsweise der beiden Algorithmen

In diesem Abschnitt wird auf den Aufbau und die Funktionsweise der beiden Algorithmen eingegangen. Während der FP-Growth Algorithmus des SPMF Data Mining Frameworks für die Erläuterung des Aufbaus und der Funktionsweise von Frequent Itemset Mining herangezogen wird [32], findet die Erläuterung des Aufbaus und der Funktionsweise von Sequential Pattern Mining unter Einbeziehung des PrefixSpan Algorithmus vom SPMF Data Mining Framework statt [40].

7.2.1 Aufbau und Funktionsweise von Frequent Itemset Mining

Die Abbildung 41 zeigt wie der FP-Growth Algorithmus die Daten aus dem Input.txt File liest und die Ergebnisse in den Output.txt File schreibt. Die Transaktionsdatenbank wird hier dem Algorithmus als Text File zur Verfügung gestellt. Jede Zeile in dem Input.txt File entspricht einer Transaktion. Die Items innerhalb einer Transaktion sind durch Leerzeichen voneinander getrennt. Beispielweise stellt die erste Zeile in dem Input.txt File eine Transaktion dar, die aus den Items {File1}, {File2} und {File6} besteht. Der Algorithmus liest diese Transaktionen Zeile für Zeile und erzeugt die Frequent Itemsets. Der Output.txt File der Abbildung 41 repräsentiert die Frequent Itemsets, die nach der Durchführung des FP-Growth Algorithmus mit einem Minimumsupportwert = 0.5 (3 Transaktionen) in den Output.txt File hinzugefügt wurden. In diesem Output.txt File entspricht jede Zeile einem Frequent Itemset. Zum Beispiel besagt die zweite Zeile des Output.txt Files, dass das Frequent Itemset, welches die Items {File5} und {File3} enthält, einen Support von 3 Transaktionen hat [32].

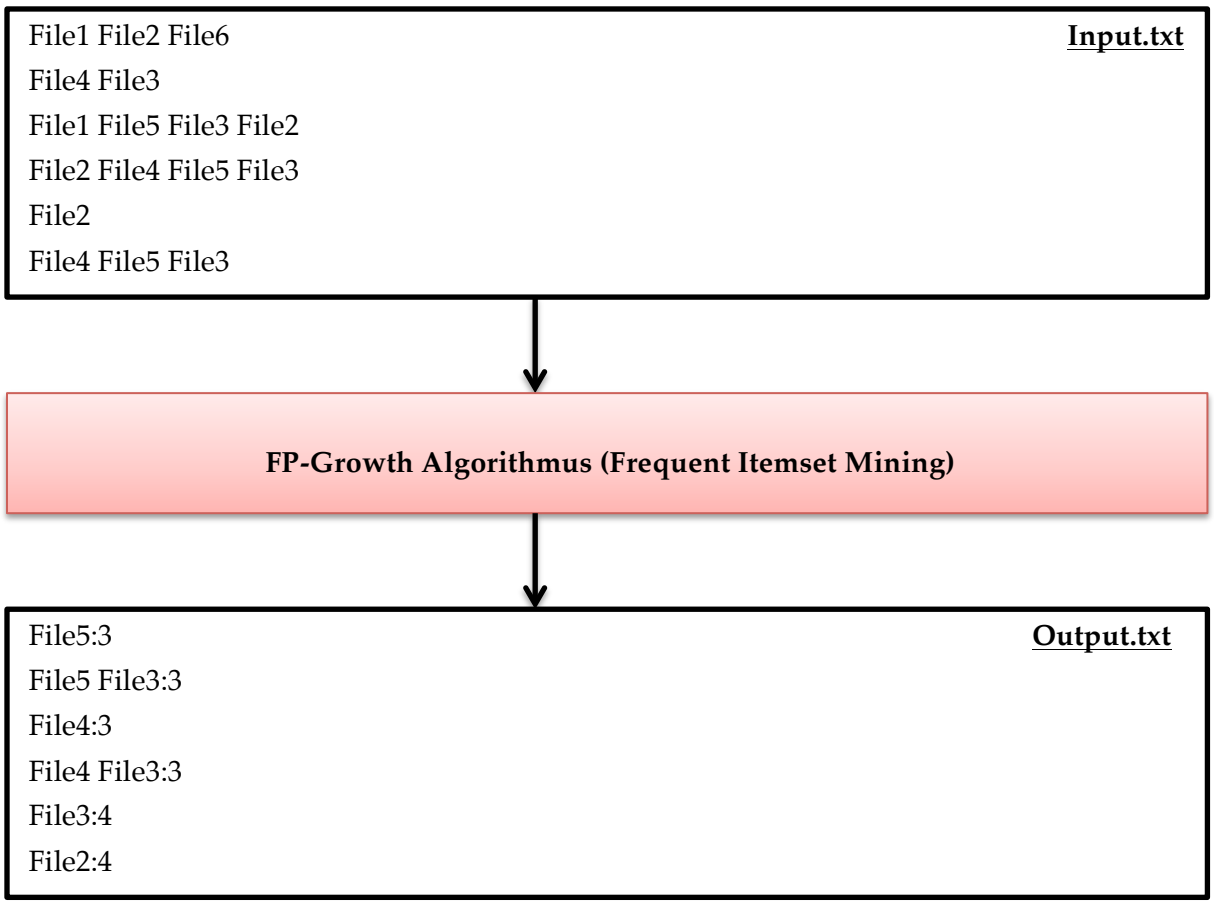

Abbildung 41: Lese- und Schreibeoperation von FP-Growth Algorithmus [32]

7.2.2 Aufbau und Funktionsweise von Sequential Pattern Mining

Hier wird der Aufbau und die Funktionsweise von Sequential Pattern Mining anhand des PrefixSpan Algorithmus detaillierter beschrieben.

Die Aufgabe des PrefixSpan Algorithmus ist es sequenzielle Mustern (Sequential Patterns) in Sequenzdatenbanken zu entdecken. Dieser Algorithmus bekommt als Input eine Sequenzdatenbank und einen Minimumsupportwert in dem Intervall von [0.1, 1.0]. Nach dem der Algorithmus die Daten aus der Sequenzdatenbank gelesen hat, führt er die Analyse von diesen Inputdaten durch und erzeugt dann einige Ergebnisse. Bei diesen Ergebnissen handelt es sich um häufigen sequenziellen Mustern (Frequent Sequential Pattern), deren Suppportwerte größer oder gleich dem Minimumsupportwert sind.

Im Folgenden wird der Aufbau und die Funktionsweise des PrefixSpan Algorithmus des SPMF Data Mining Frameworks repräsentiert und erläutert. Die Sequenzdatenbank, die den Input für den Algorithmus bildet, wird als Text File zur Verfügung gestellt. Das bedeutet der PrefixSpan Algorithmus liest die Daten aus einem Text File, bearbeitet sie mit Hilfe von dem durch den Benutzer eingegebenen Minimumsupportwert und speichert die Ergebnisse dann in einen anderen Text File.

Die Abbildung 42 zeigt den Ablauf der Lese- und Schreibeoperation des PrefixSpan Algorithmus. Dabei sind die Daten in den Abbildungen 41 und 42 die gleichen. Der Unterschied liegt in der Darstellung und der Interpretation dieser Daten. Man kann hier sehen, wie der Algorithmus die Daten aus dem Input.txt File liest und bei einem gegebenen Minimumsupportwert von 0.5 die Ergebnisse in den Output.txt File schreibt.

Betrachtet man den Input.txt File, so kann man sehen, dass jede Zeile einer Sequenz von einer Sequenzdatenbank entspricht. Der Wert „-1“ indiziert das Ende eines Itemsets, während das Ende einer Sequenz durch den Wert „-2“ indiziert wird. Beispielweise repräsentiert die zweite Zeile des Input.txt Files eine Sequenz, die aus dem Itemset {File4}, gefolgt durch das Itemset {File3} besteht.

Ist die Analyse abgeschlossen, dann werden die Ergebnisse in den Output.txt File geschrieben. Jede Zeile des Output.txt Files stellt einen Frequent Sequential Pattern dar. Der Wert „-1“ indiziert auch hier das Ende von einem Itemset. In jeder Zeile werden die Sequential Pattern zuerst angegeben. Danach erscheint das Schlüsselwort „#SUP:“ gefolgt durch einen Integerwert, welcher den Support von dem Sequantial Pattern angibt. Die zweite Zeile in dem Output.txt File repräsentiert den Frequent Sequential Pattern, welcher aus dem Itemset {File5}, gefolgt durch das Itemset {File3} besteht und einen Support von 3 Sequenzen hat [40].

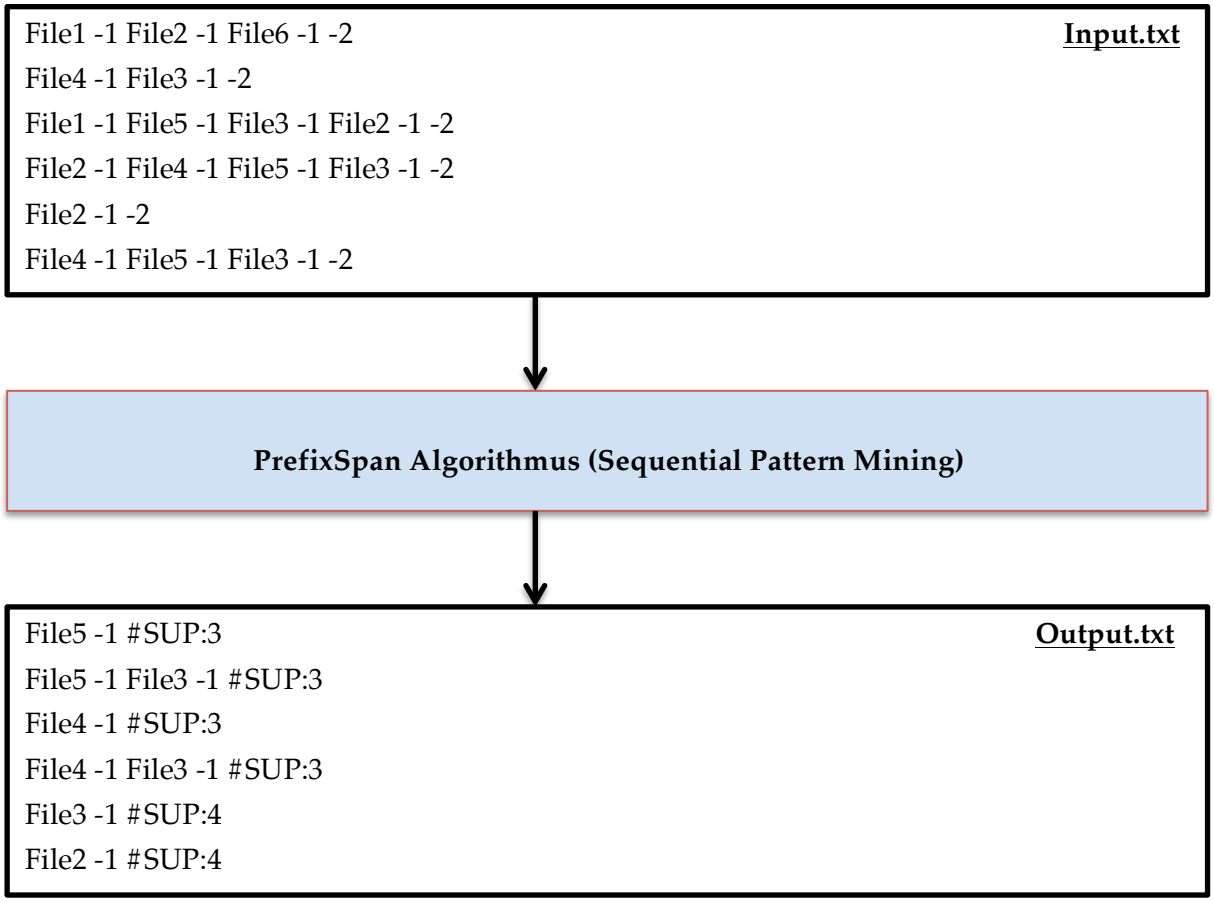

Abbildung 42: Lese- und Schreibeoperation des PrefixSpan Algorithmus [40]

7.2.3 Tabellarische Gegenüberstellung der beiden Algorithmen

Die Erläuterungen haben gezeigt, dass die beiden Algorithmen Unterschiede aber auch Ähnlichkeiten aufgewiesen haben. Die nachfolgende Tabelle 8 stellt zum Schluss die Gemeinsamkeiten und die Unterschiede noch einmal tabellarisch im Überblick dar.

Tabelle 8: Tabellarische Gegenüberstellung von FP-Growth und PrefixSpan Algorithmen

	FP-Growth Algorithmus	**PrefixSpan Algorithmus**
Input	Minimumsupportwert und Transaktionsdatenbank	Minimumsupportwert und Sequenzdatenbank
Output	Häufige Itemmengen	Häufige sequenzielle Muster
Eine Zeile in der Datenbank	Ist eine Transaktion	Ist eine Sequenz

7.3 Konzept zur Integration von Sequential Pattern Mining ins SRM Plug-In

Die Entwicklung des SRM Plug-Ins bildet das Ziel dieser Diplomarbeit. In diesem Plug-In wurde, wie in den vergangenen Kapiteln sehr detailliert erläutert, die Data Mining Technik Frequent Itemset Mining für das Suchen der gekoppelten Dateiänderungen angewendet. Selbstverständlich ist auch die Anwendung von Sequential Pattern Mining für die Suche der gekoppelten Dateiänderungen möglich. Dies ist vor allem bei Sequenzdatenbanken vom Vorteil. Weiterhin würde die Anwendung von Sequential Pattern Mining innerhalb des SRM Plug-Ins den Entwicklern die Möglichkeit geben, sich nicht nur über die gekoppelten Dateiänderungen sondern auch über deren Reihenfolge informieren zu lassen. Aus diesem Grund wird hier auf konzeptioneller Ebene beschrieben, wie die Integration des PrefixSpan Algorithmus von Sequential Pattern Mining in das SRM Plug-In realisiert werden kann.

7.3.1 Lese- und Schreibeoperationen auf Datenbanktabellen

Bevor der PrefixSpan Algorithmus in das SRM Plug-In integriert wird, sind einige Änderungen am Algorithmus erforderlich. Diese Änderungen umfassen das Lesen von den Daten und das Schreiben der Ergebnisse. Aktuell liest dieser Algorithmus die Daten aus einem Input.txt File, bearbeitet diese Daten mit Hilfe des durch den Benutzer eingegebenen Minimumsupportwertes und schreibt anschließend die Ergebnisse in einen Output.txt File.

Die Abbildung 43 zeigt wie das Lesen von einer Inputdatenbanktabelle und das Schreiben der Ergebnisse in eine Outputdatenbanktabelle realisiert werden soll. Die Lese- und Schreibeoperation auf die Datenbanktabellen bestehen insgesamt aus 6 Punkten, die im Folgenden erläutert werden:

1. Der Benutzer übergibt dem PrefixSpan Algorithmus einen Minimumsupportwert.

2. Die Daten aus der Inputdatenbanktabelle werden in einem Array gespeichert.

3. Dieser Array wird dem PrefixSpan Algorithmus als Input übergeben.

4. Die Ergebnisse der Analyse, also die Frequent Sequential Pattern werden in einem anderen Array gespeichert.

5. Anschließend werden dann die Ergebnisse aus dem Array gelesen und in die Outputdatenbanktabelle geschrieben.

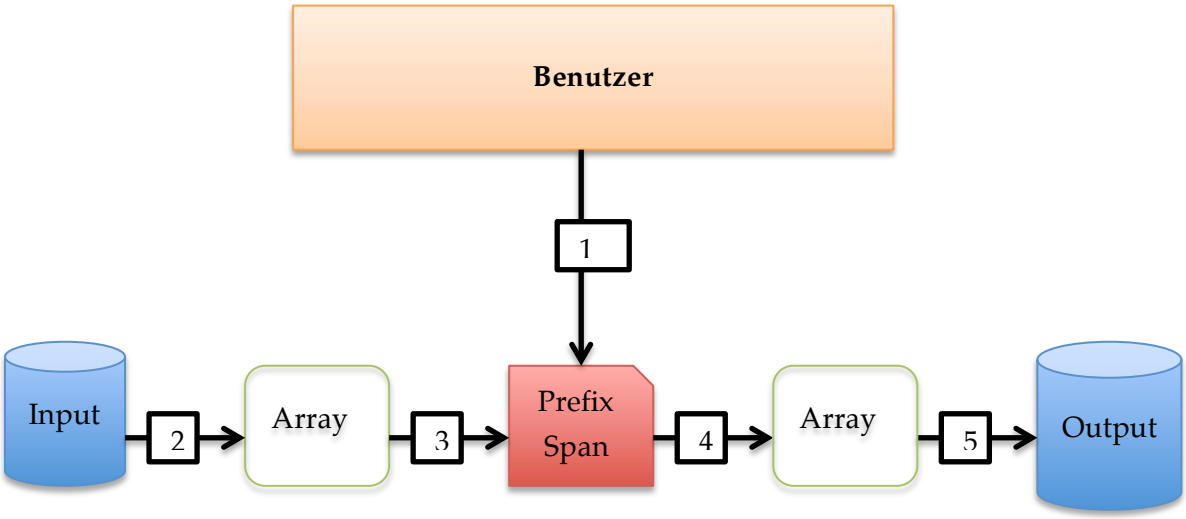

Abbildung 43: Lese- und Schreibeoperationen von PrefixSpan Algorithmus auf Datenbanktabellen

7.3.2 Architektur SRM Plug-Ins unter Anwendung von Sequential Pattern Mining

Die Abbildung 44 zeigt die Architektur des SRM Plug-Ins, die für die Ermittlung von gekoppelten Dateiänderungen den PrefixSpan Algorithmus von Sequential Pattern Mining anwendet. Man kann sehr leicht sehen, dass die Architektur des SRM Plug-Ins aus der Abbildung 44 größtenteils mit der Architektur des SRM Plug-Ins aus der Abbildung 23 übereinstimmt. Der einzige Unterschied zwischen diesen beiden Abbildungen ist der für die Analyse eingesetzte Algorithmus. Die Integration des PrefixSpan Algorithmus kann aufgrund von Lizenzangelegenheiten nur als externer Jar File in das SRM Plug-In erfolgen [41].

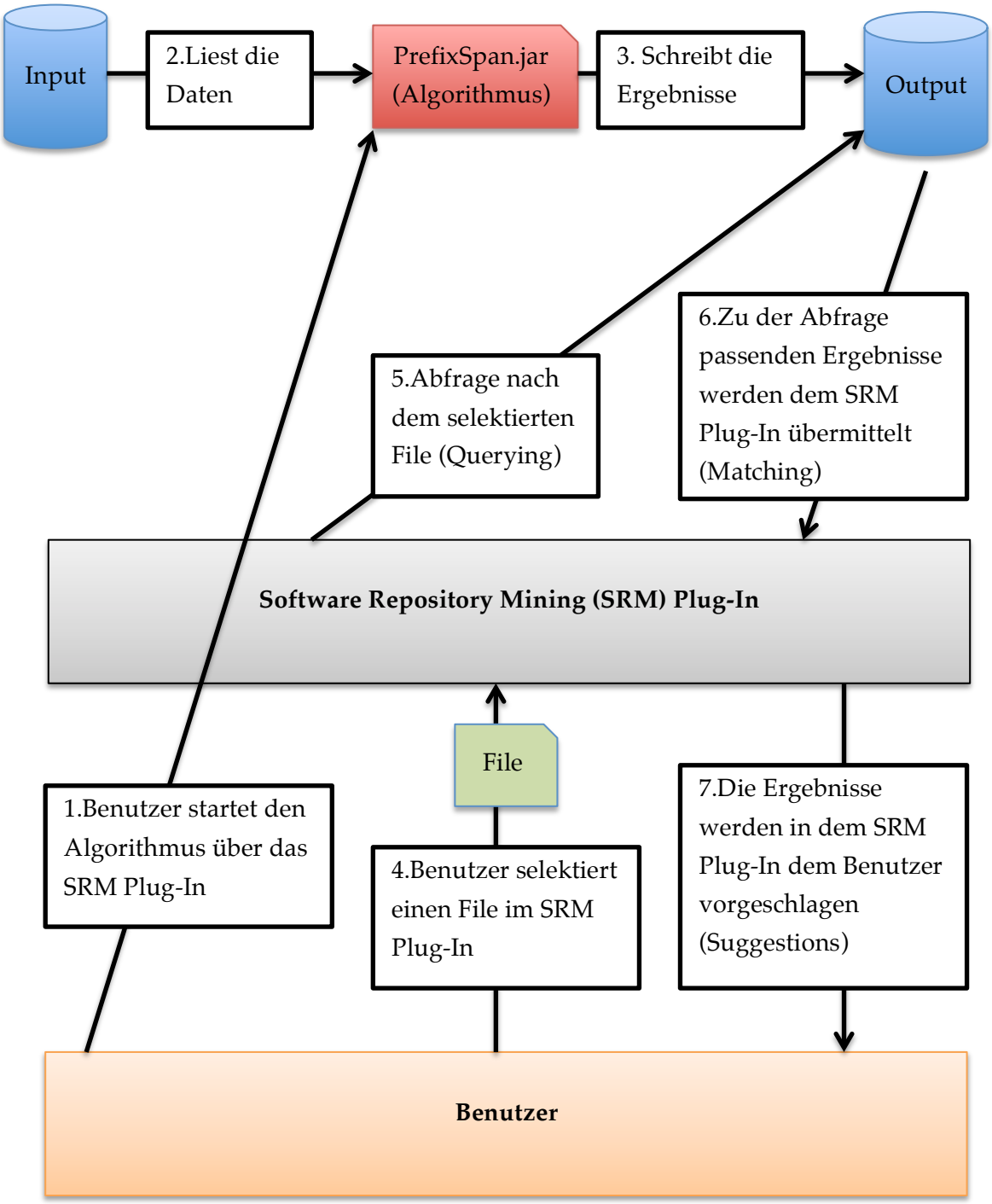

Abbildung 44: Architektur SRM Plug-In mit PrefixSpan Algorithmus

7.4 Fazit des Vergleiches

Die Realisation des SRM Plug-Ins unter der Anwendung von Sequential Pattern Mining zur Extraktion der gekoppelten Dateiänderungen aus Software-Repositories würde den Vorteil mit sich bringen, dass die ermittelten gekoppelten Dateiänderungen gleichzeitig in einer bestimmten sequenziellen Reihenfolge angezeigt werden und somit den Benutzer darüber informieren, in welcher Reihenfolge die gekoppelten Dateiänderungen zu ändern sind.

Des Weiteren kann man aus dem vorherigen Kapitel auch erkennen, dass der Aufwand für die Integration von Sequential Pattern Mining in das SRM Plug-In minimal ist. Der Aufwand besteht lediglich in der Transformation des PrefixSpan Algorithmus entsprechend der Abbildung 43. D.h., der Quellcode des PrefixSpan Algorithmus muss so geändert werden, dass er Daten aus einer Datenbanktabelle lesen und die Ergebnisse in eine andere Datenbanktabelle schreiben kann. Anschließend findet dann die Integration dieses transformierten PrefixSpan Algorithmus in das SRM Plug-In als externer Jar File statt. Alle anderen Bereiche des SRM Plug-Ins bleiben unverändert.

8 Evaluation

In diesem Kapitel wird das entwickelte SRM Plug-In seiner Zielgruppe, also den Entwicklern vorgestellt, um zu überprüfen, ob es den Anforderungen der Entwickler gerecht ist.

8.1 Überblick über den Evaluationsprozess

Der Evaluationsprozess gliedert sich in drei Phasen. Vorbereitungsphase, Testphase, Analyse- und Auswertungsphase. Die Abbildung 45 repräsentiert die einzelnen Phasen.

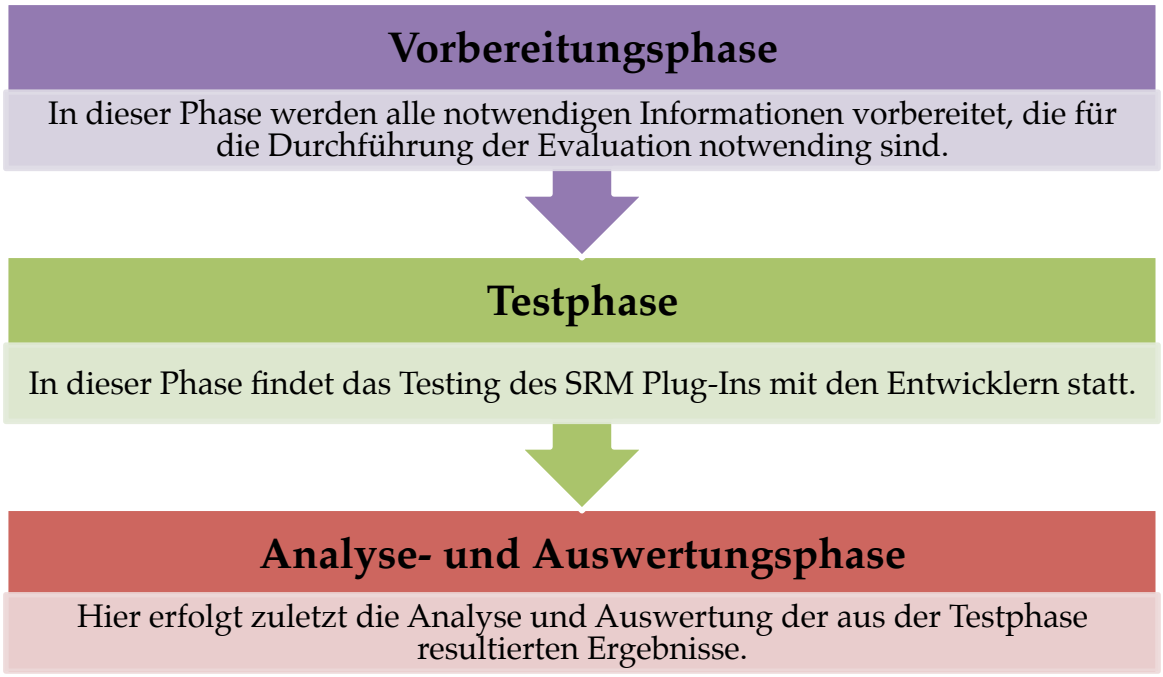

Abbildung 45: Phasen des Evaluationsprozesses

8.2 Vorbereitungsphase

8.2.1 Erstellung von Testdaten

Um die Evaluation in den nächsten Schritten überhaupt durchführen zu können, benötigt man Testdaten. Unter der Erstellung von Testdaten versteht man die Erzeugung von Testdatenbanken, deren Inhalte mit Beispielwerten belegt werden. Es werden zwei Datenbanktabellen erzeugt. Die eine Datenbanktabelle ist die Inputtabelle. Diese Inputtabelle beinhaltet die Files, welche in der Testphase analysiert werden und besteht insgesamt aus 10 Transaktionen, die unterschiedliche Anzahl an Items enthalten. Die zweite Tabelle ist die

Commit Message Tabelle, die ebenfalls mit Beispielwerten erzeugt wird. Diese Commit Message Tabelle beinhaltet die Einträge zu den Transaktionen aus der Inputtabelle.

8.2.2 Festlegung des Testfallszenarios

Sind die Datenbanktabellen erzeugt, wird ein Testfallszenario festgelegt, anhand derer die Entwickler das Tool testen sollen. Die Abbildung 46 zeigt den Inhalt des Testfallszenarios.

Testfallszenario

Bitte führen Sie die nachfolgenden Schritte aus :

1. Öffnen Sie Eclipse.
2. Importieren Sie das ASTPA Projekt in das Project Explorer.
3. Im Eclipse-Menü `Window` auswählen → `Show View`→ `Other`.
4. Danach den Ordner SRM Plug-In öffnen und die Views (Execution View, Coupled Changes, Commit Message View und Commit View) markieren und auf `OK` Klicken.
5. Gehen Sie dann auf die Execution View.
6. Geben Sie in das Textfeld einen Wert im Intervall [0.1, 1.0] ein, wobei 0.1= 10% und 1.0= 100% ist.
7. Klicken Sie auf den „Start Execution“ Button.
8. Klicken Sie auf einen File in dem Project Explorer.
9. Klicken Sie auf die Coupled Changes. Die gekoppelten Dateiänderungen des selektierten Files werden in dem Coupled Changes angezeigt.
10. Gehen Sie auf die Commit View.
11. Öffnen Sie das Combobox. Die Transaktionen, welche die gekoppelten Dateiänderungen beinhalten werden in dem Combobox angezeigt.
12. Selektieren Sie einen von diesen Transaktionen.
13. Öffnen Sie die Commit Message View. Die Einträge zu der selektierten Transaktion erscheinen dann in dem Commit Message View.
14. Schließen Sie Eclipse.

Abbildung 46: Testfallszenario

8.2.3 Erstellung von dem Fragebogen

Am Ende der Vorbereitungsphase findet dann die Erstellung des Fragebogens statt. Durch diesen Fragebogen, bekommt man einen Feedback von den Entwicklern, anhand derer dann die Analyse und die Auswertung der Evaluation vollzogen wird. Die Abbildung 47 zeigt den Aufbau des Fragebogens.

Fragebogen zum SRM Plug-In

Dauer: 10 Minuten

-2 = trifft nicht zu, -1 = trifft eher nicht zu, 0 = weder noch, 1 = trifft eher zu, 2 = trifft zu

Fragen		Ergebnisse				
		-2	-1	0	1	2
1	Ich finde das **SRM Plug-In** sehr hilfreich, um Software Repository Mining durchzuführen	☐	☐	☐	☐	☐
2	Ich konnte in dem **SRM Plug-In** alle für mich relevanten Informationen finden	☐	☐	☐	☐	☐
3	Die Informationen, die in dem **SRM Plug-In** repräsentiert wurden sind sehr nützlich	☐	☐	☐	☐	☐
4	Das **SRM Plug-In** ist eine gute Unterstützung, um Bugfixings- und Modifikationsaufgaben viel schneller und effizienter zu erledigen	☐	☐	☐	☐	☐
5	Die Benutzeroberfläche des **SRM Plug-In** ist sehr übersichtlich	☐	☐	☐	☐	☐
6	Ich konnte durch die Anwendung von **SRM Plug-In** sehr viel Zeit sparen	☐	☐	☐	☐	☐
7	Die Informationen, die in dem **SRM Plug-In** repräsentiert werden sind übersichtlich und leicht verständlich	☐	☐	☐	☐	☐
8	Die Bedienung des **SRM Plug-Ins** ist einfach und komfortabel	☐	☐	☐	☐	☐
9	Die Verfahrensweise in dem **SRM Plug-In** ist gut strukturiert	☐	☐	☐	☐	☐
10	Ich würde das **SRM Plug-In** benutzen und weiterempfehlen	☐	☐	☐	☐	☐

Abbildung 47: Fragebogen zum SRM Plug-In

8.3 Testphase

Die Testphase bildet den Kernpunkt der Evaluation, da hier durch die Feedbacks der Entwickler, das SRM Plug-In analysiert werden kann. An der Testphase nahmen insgesamt 10 Entwickler teil, die folgenden Tätigkeiten nachgehen:

- Ein Doktorand der Universität Stuttgart.
- Zwei Absolventen der Studiengang Informatik.
- Ein Absolvent der Studiengang Wirtschaftsinformatik.
- Ein SimTech Student der Universität Stuttgart.
- Zwei Computerlinguistik-Studenten der Universität Stuttgart.
- Drei Informatikstudenten der Universität Stuttgart.

Dabei hatte jeder dieser Teilnehmer unterschiedliche Kenntnisse beim Umgang mit dem Eclipse Tools. Jedem Teilnehmer wurden das Testfallszenarioblatt und der Fragebogen übergeben. Die Dauer der Testphase pro Teilnehmer betrug ca. 20-30 Minuten.

8.4 Auswertungsphase der Evaluationsergebnisse

Die letzte Phase legt die Auswertung der Evaluationsergebnisse dar. Im Folgenden werden die Ergebnisse der Umfragen präsentiert und kommentiert.

Die Abbildung 48 präsentiert die Ergebnisse, welche von den Umfragen mit den Entwicklern entstanden sind. Obwohl der Fragebogen auch die Punkte „trifft nicht zu“ und „trifft eher nicht zu“ als Auswahl hatte, wurden diese Punkte durch die Entwickler nicht ausgewählt. Daraus kann man folgern, dass das SRM Plug-In von den meisten Entwicklern als seinen Anforderungen gerecht empfunden wurde.

Die Fragen in dem Fragebogen gliedern sich in funktionale und nicht funktionale Eigenschaften des SRM Plug-Ins.

Die Fragen 1, 2, 3, 4, 6 und 10 beziehen sich auf den funktionalen Teil des SRM Plug-Ins. Man kann aus der Abbildung 48 sehr deutlich sehen, dass genau diese Punkte durch die Teilnehmer sehr gut bewertet wurden, was besagt, dass die Entwickler die Funktionalitäten des SRM Plug-Ins sehr nützlich und hilfreich finden. Vor allem die Antworten der Entwickler auf die Frage 10, ob sie das SRM Plug-In benutzen und weiterempfehlen würden, wiederspiegeln die Zufriedenheit der Entwickler mit den Funktionalitäten des SRM Plug-Ins.

Die Fragen 5, 7, 8 und 9 hingegen repräsentieren jene Fragen, die das Design des SRM Plug-Ins anbetreffen. Bei diesen Fragen haben die meisten Teilnehmer wieder sehr viele positive Antworten gegeben, während aber auch einige Teilnehmer keine Stellung zu diesen Fragen genommen haben. Vor allem auf die Frage 5 haben 40% der Probanden (also 4 Personen) keine Stellung genommen. Dies bedeutet, dass bei der Benutzeroberfläche des SRM Plug-Ins noch einige Optimierungsarbeiten durchzuführen sind.

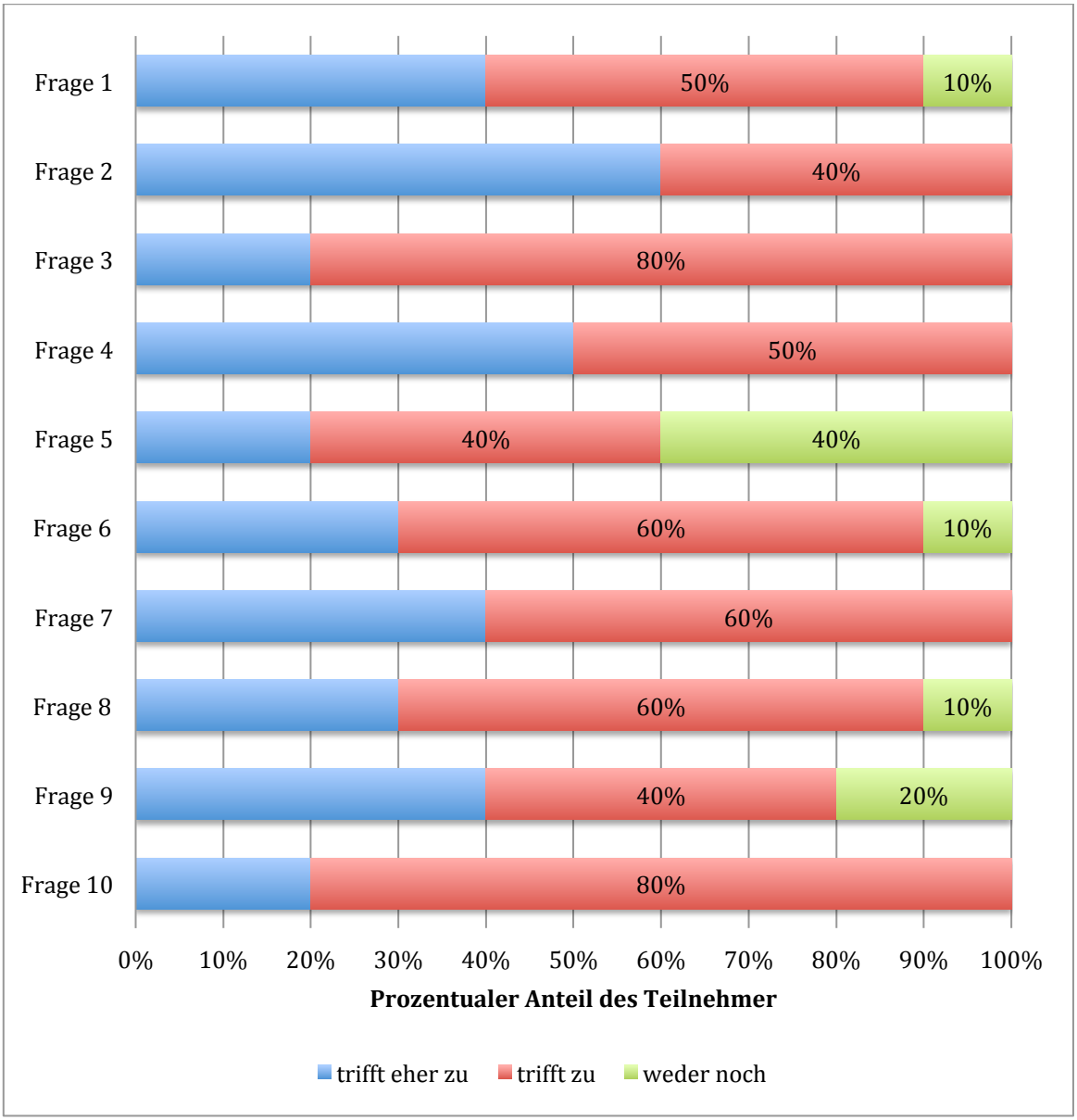

Abbildung 48: Überblick über die Evaluationsergebnisse

9 Zusammenfassung und Ausblick

Entwickler, die sich mit einem Softwaresystem nicht gut auskennen, benötigen oft lange bei der Durchführung ihrer Aufgaben. Ändert beispielweise der Entwickler einen File oder einen Code eines Softwaresystems, so sind meistens auch Änderungen an anderen Files bzw. Codes erforderlich. Das bedeutet, der Entwickler muss sobald er eine Änderung an einem File bzw. Code vorgenommen hat, die mit diesem File zusammen geänderte Files auch ändern. Diese Tatsache führt oft dazu, dass der Entwickler bei großen Softwaresystemen den Überblick verliert und nicht alle Files ändern kann. Aber auch die erfahrenen Entwickler könnten von dem gleichen Problem betroffen sein. Um genau dieser Problematik entgegen zu wirken und die Entwickler in ihren Modifikations- und Wartungsaufgaben zu unterstützen wurde im Rahmen dieser Diplomarbeit ein Eclipse Plug-In namens Software Repository Mining (SRM) Plug-In entwickelt. Dieses Plug-In ermöglicht es den Entwicklern Software Repositories zu analysieren, um sich die Ergebnisse in dem Eclipse IDE anzeigen zu lassen.

Aktuell hat der Entwickler die Möglichkeit folgenden Aufgaben mit Hilfe des SRM Plug-Ins zu erledigen:

1. Der Entwickler kann die Analyse der Software-Repositories über das Execution View des SRM Plug-Ins durchführen.

2. Der Entwickler kann sich alle Informationen über die gekoppelten Dateiänderungen in dem SRM Plug-In anzeigen lassen. Ändert er einen File in einem Projekt, so erscheinen ihm diese Informationen in drei separaten Views.

   Die Coupled Changes listet alle gekoppelten Dateiänderungen auf. Somit kann der Entwickler in diesem View sehen, welche anderen Files er noch ändern muss.

   Die Commit View repräsentiert alle Transaktionen, in denen die gekoppelten Dateiänderungen sich befinden. Durch die Informationen in dem Coupled Changes weiß der Entwickler zwar welche weiteren Files er ändern muss, aber er weiß nicht, wo die zu ändernden Files sich befinden. Diese Informationslücke wird dann durch die Commit View gedeckt. In diesem Commit View wird dann der Entwickler zusätzlich auch noch darüber informiert, wo er diese Änderungen vorzunehmen hat.

   Zuletzt existiert in dem Plug-In noch ein letztes View namens Commit Message View, die dem Entwickler zeigt, wie er die Änderung durchzuführen hat.

Der Ablauf dieser Diplomarbeit untergliederte sich in die Phasen Anforderungsphase, Konzeptphase, Implementierungsphase und Evaluationsphase, wobei die ersten drei Phasen die Implementierung und die letzte Phase die Evaluation des SRM Plug-Ins darstellen.

Zuerst wurden in der Anforderungsphase die Anforderungen an das SRM Plug-Ins definiert. Dabei wurde festgelegt, dass das Plug-In für die Analyse die Data Mining Technik „Frequent Itemset Mining " anwenden soll. Weitere Anforderungen wurden dann wie folgt definiert:

- Der Entwickler soll die Frequent-Itemset-Analyse von dem SRM Plug-In aus starten können.
- Die Ergebnisse der Frequent-Itemset-Analyse sollen dem Entwickler in dem SRM Plug-In vorgestellt werden.

In der darauffolgenden Phase wurde dann ein den Anforderungen gerechtes Konzept für das SRM Plug-Ins entwickelt. Dieses Konzept gibt Aufschluss darüber wie der Aufbau und die Funktionsweise des Datenflusses in dem SRM Plug-Ins auszusehen hat. Zu diesem Zweck wurden die nachfolgenden Punkte in Betracht gezogen und konzeptionell abgebildet:

- Ausführen der Frequent-Itemset-Analyse durch die Entwickler.
- Lesen der Daten aus der Datenbanktabelle.
- Schreiben der Ergebnisse der Frequent-Itemset-Analyse in eine andere Datenbanktabelle.
- Untersuchen dieser Ergebnisse durch die Entwickler.
- Repräsentation der Informationen in unterschiedlichen Views des SRM Plug-Ins.

Nachdem die Konzeptphase vollendet war, fand dann in der Implementierungsphase die Umsetzung dieses Konzeptes statt. Dabei wurde in erster Linie der Algorithmus, welcher die Frequent-Itemset-Analyse durchführen soll, transformiert. Diese Transformation umfasste das Lesen der Daten aus der Datenbanktabelle und Schreiben der Ergebnisse in eine andere Datenbanktabelle. Anschließend wurde dann das SRM Plug-In mit den notwendigen Views (Execution View, Coupled Changes, Commit View und Commit Message View) erstellt und der transformierte Algorithmus als externer Jar File in das SRM Plug-In integriert.

Die letzte Phase ist die Evaluationsphase. In dieser Phase wurde das SRM Plug-In durch die Entwickler getestet, um zu überprüfen, ob es ihren Anforderungen entspricht oder nicht. Diese Phase untergliedert sich wiederum in drei Phasen (siehe Abbildung 45). Die erste Phase ist die Vorbereitungsphase. In dieser Phase wurden alle für die Evaluation notwendigen Daten vorbereitet. Zu diesem Zweck wurden Testdaten generiert. Bei diesen Testdaten handelt es sich um Datenbanktabellen, die für die Frequent-Itemset-Analyse

benötigt werden. Im Anschluss daran wurde ein Testfallszenario erstellt (siehe Abbildung 46), um den Entwicklern vorzuschreiben, wie sie das Plug-In zu testen haben. Zuletzt wurde dann auch noch ein Fragebogen mit zehn Fragen über das SRM Plug-In erzeugt. In der Testphase fand das Testing des SRM Plug-Ins mittels des Testfallszenarios durch die Entwickler statt. Im Anschluss daran haben die Entwickler ihre Quoten zu dem SRM Plug-In in den Fragebögen kenntlich gemacht. An dem Testing haben insgesamt 10 Entwickler teilgenommen. Die letzte und somit auch die entscheidendste Phase der Evaluationsphase ist die Analyse- und Auswertungsphase. In dieser letzten Phase wurden die durch die Entwickler abgegebenen Fragebögen analysiert und ausgewertet. Die Abbildung 48 veranschaulicht die Ergebnisse der Evaluationsphase in einem Balkendiagramm graphisch. Dabei bestand der Fragebogen (siehe Abbildung 47) aus insgesamt 10 Fragen, wobei 6 Fragen über die funktionalen Eigenschaften und 4 über die nicht funktionalen (Design) Eigenschaften gestellt waren. Für die Antwortmöglichkeiten wurde die Likert-Skala angewendet. Die Ergebnisse in der Abbildung 48 veranschaulichen sehr deutlich, wie gut das SRM Plug-In durch die Entwickler bewertet wurde. Die Fragen 1, 2, 3, 4, 6 und 10 repräsentieren die funktionalen Eigenschaften des SRM Plug-Ins. Die Antworten der Entwickler zu diesen Fragen bekräftigen noch einmal ganz gut, dass das SRM Plug-In seinen Anforderungen gerecht wurde. Vor allem die positive Antwort der überwältigenden Mehrheit der Teilnehmer auf die Frage 10, ob sie das SRM Plug-In benutzen und weiterempfehlen würden, bestätigen die Zufriedenheit der Teilnehmer mit den Funktionalitäten des SRM Plug-Ins.

9.1 Ausblick

Da es sich bei diesem SRM Plug-In um einen Prototyp handelt, sind Erweiterungen in vielerlei Hinsicht auch möglich.

Das aktuelle SRM Plug-In verwendet Frequent Itemset Mining für die Durchführung der Analysen. Die eine Erweiterung des SRM Plug-Ins ist die Integration des Sequential Patten Mining in das SRM Plug-In. Im Kapitel 7 wurde bereits auf konzeptioneller Ebene beschrieben, wie die Integration von Sequential Pattern Mining in das SRM Plug-In verwirklicht werden kann. Dies würde den Vorteil mit sich bringen, dass Entwickler neben den Informationen über die gekoppelten Dateiänderungen zusätzlich auch noch die Informationen darüber bekommen würden, in welcher Reihenfolge diese gekoppelten Dateiänderungen zu ändern sind.

Eine weitere Erweiterung betrifft das Design bzw. die Gebrauchstauglichkeit (*eng. usability*) des SRM Plug-Ins. Da im Rahmen dieser Diplomarbeit die Anforderungen an das Plug-In die funktionalen Eigenschaften anbetrafen, wurde hinsichtlich des Design und der Gebrauchstauglichkeit des Plug-Ins nicht so viel Zeit investiert. Der Schwerpunkt lag

vielmehr in der Realisation der funktionalen Anforderungen des SRM Plug-Ins. Die Fragen 5, 7, 8, 9 aus dem Fragebogen (siehe Abbildung 47) betreffen das Design und die Gebrauchstauglichkeit (*eng. usability*) des SRM Plug-Ins. Aus der Abbildung 48 ist ganz gut ersichtlich, dass diese Fragen zwar durch die Entwickler ganz gut bewertet wurden, aber einige Teilnehmer haben wiederum auch keine Stellung zu den Fragen genommen. Vor allem haben 40% der beteiligten Teilnehmer auf die Frage 5, ob Sie die Benutzeroberfläche sehr übersichtlich finden, die Antwort „weder noch“ angekreuzt. Diese Antwort besagt somit, dass es im Bereich der Benutzeroberflächengestaltung und der Gebrauchstauglichkeit des SRM Plug-Ins noch einige Erweiterungen möglich sind.

Literaturverzeichnis

[1]: R. Robbes, M. Lanza: A Change-based Approach to Software Evolution, Electronic Noted in Theoretical Computer Science, 166, S.93-109, 2007. Siehe: http://www.inf.usi.ch/faculty/lanza/Downloads/Robb2007a.pdf.

[2]: A.T.T. Ying, G.C. Murphy, R. Ng, M.C. Chu-Carroll: Predicting Source Code Changes by Mining Revision History. In IEEE Trasnaction on Software Engeneering 30(9). IEEE, S. 574-586, 2004.

[3]: H. Kagdi, M.L. Collard, J.I. Maletic: A survey and taxonomy of approaches for mining software repositories in the context of software evolution, Journal of Software Maintanance and Evolution: Research and Practice, 19(2), S.77-131, DOI: 10.1002/smr.344, 2007.

[4]: M. D'Ambros, M. Lanza, R. Robbes: On the Relationship Between Change Coupling and Software Defects. In WCRE '09. 16th Working Conference on Reverse Engeneering, 2009. IEEE, S.135-144, 2009.

[5]: J. Gonnet: Data Mining within Eclipse. Diplomarbeit, Universität Zürich, 2007.

[6]: J. Han, M. Kamber: Data Mining. Concepts and Techniques, Waltham, 2005, S.5-10.

[7]: T.A. Runkler: Data Mining. Methoden und Algorithmen intelligenter Datenanalyse, Wiesbaden, 2010, S.2-4.

[8]: H. Dürr: Anwendung des Data Mining in der Praxis. Seminararbeit, Universität Ulm WS 2003/2004, S.3-4.

[9]: H. Schwarz: Vorlesung Data-Warehouse-, Data-Mining- und OLAP-Technologien. WS 2010/2011, Universität Stuttgart.

[10]: B. Felix: SPSS 8. Professionelle Statistik unter Windows, Hamburg, 1998,S.691-700.

[11]: T. Srivastava: Getting your clustering right (Part1) (12.11.2013), URL:http://www.analyticsvidhya.com/blog/2013/11/getting-clustering-right/ (Letzter Zugriff am 07.04.2015).

[12]: I. Tudor: Association Rule Mining as a Data Mining Technique, BULETINULuniversitatii Petrol-Gaze din Ploiesti, Vol.LX No1/2008, S.49-56.

[13]: P. Helge: Data Mining: Verfahren, Prozesse, Anwendungsarchitektur. München, 2005.

[14]: E. Lüdecke: Ermittlung von Assoziationsregeln aus großen Datenmengen (14.04.2010),URL:http://www.fhschmalkalden.de/schmalkaldenmedia/Ermitlung_von_Assoziationsregeln_aus_gro%C3%A1en_Datenmengen-p-13278.pdf.
(Letzter Zugriff am 07.04.2015).

[15]: J. Mühle : Automatische Generierung von Assoziationsregeln, 2009, S.12.

[16]: M. Soldatova: Diskussion und Implementierung von Varianten des FP-Growth Algorithmus in relationalen Datenbanksystemen. Studienarbeit, Leibniz Universität Hannover, 2007.

[17]: F. Verhein: Frequent Pattern Growth (FP-Growth) Algorithm. An Introduction, Universität Sydney, 2008.

[18]: T. Bollinger: Assoziationsregeln – Analyse eines Data Mining Verfahrens. In Informatik-Spektrum, 19(5), Springer-Verlag, S. 257-261, DOI: 10.1007/s002870050036, 1996, URL: http://www.springerlink.com/content/katecd6pyyuevjfd/.
(Letzter Zugriff am 07.04.2015).

[19]: J. Han, Y. Pei, Y. Yin, R. Mao: Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. In Data Mining and Knowledge Discovery, 8(1), S.53-87, DOI: 10.1023/B, 2004.

[20]: D. Gallardo, C. Aniszczyk: Get started with the Eclipse Platform (17.07.2007), URL:http://www.ibm.com/developerworks/opensource/library/os-eclipse-platform/.
(Letzter Zugriff am 07.04.2015).

[21]: A. Becker: Entwicklung eines JaMP-Editors für das Eclipse-Framework. Studienarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2007.

[22]: M. Witte: Portierung, Erweiterung und Integration des ObjectTeams/Java Compilers für die Entwicklungsumgebung Eclipse. Diplomarbeit, Technische Universität Berlin, 2003.

[23]: C. Wressnegger: Coco/R Eclipse Plug-In. Bachelorarbeit, Johannes Kepler Universität Linz, 2006.

[24]: H.-C. Frank: Implementierung eines Eclipse-Plugin zur Refaktorisierung „Replace Conditional with Polymorphism“. Masterarbeit, FernUniversität in Hagen, 2012.

[25]: The Eclipse Foundation. Extensions and Extension Points, URL: http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fconcepts%2Fextension.htm.
(Letzter Zugriff am 07.04.2015).

[26]: L.Vogel: Eclipse Extension Points and Extensions-Tutorial (27.08.2013), URL:http://www.vogella.com/tutorials/EclipseExtensionPoint/article.html
(Letzter Zugriff am 07.04.2015).

[27]: T. Zimmermann, A. Zeller, P. Weissberger, S. Diehl: Mining Version Histories to Guide Software Changes. In Proceedings of the 26th International Conference on Software Engineering. Washington, DC; USA. IEEE Computer Society, S. 563-572, 2004.

[28]: P. Kim, B. Tamersoy: Collaborativ Software Design & Development. Mining Software Repositories. Universität Texas, 2008.

[29]: A.T.T. Ying: Predicting Source Code Changes by Mining Revision History. Masterarbeit, University of British Columbia, 2001.

[30]: P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, V. S. Tseng (2014). SPMF: a Java Open-Source Pattern Mining Library. Journal of Machine Learning Research (JMLR), 15: 3389-3393.

[31]: P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, V. S. Tseng: SPMF. An Open-Source Data Mining Library. Startseite. URL: http://www.philippe-fournier-viger.com/spmf/
(Letzter Zugriff am 07.04.2015).

[32]: P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, V. S. Tseng: SPMF. An Open-Source Data Mining Library. Example 3: Mining Frequent Itemsets by Using the FP-Growth Algorithm. URL: http://www.philippe-fournier-viger.com/spmf/index.php?link=documentation.php#growth
(Letzter Zugriff am 07.04.2015).

[33]: The Eclipse Foundation. WindowBuilder - is a powerful and easy to use bi-directional Java GUI designer, URL: https://eclipse.org/windowbuilder/.
(Letzter Zugriff am 07.04.2015).

[34]: E. Clayberg: Building GUIs with WondowBuilder. In EclipseCon 2012, Reston, Virginia; USA. S.1-24, 2012.
URL:http://eclipsecon.org/europe2012/sites/eclipsecon.org.europe2012/files/Buiding-GUIs-with-WindowBuilder-EclipseCon-2012.pdf.
(Letzter Zugriff am 07.04.2015)

[35]: T. Schmidt: Überblick über den Eclipse Workbench (2002), URL: http://www.admin-wissen.de/tutorials/eclipse_workshop/ueberblick_workbench.html.
(Letzter Zugriff am 07.04.2015).

[36]: M.R. Hoffmann: Eclipse Workbench: Using the Selection Service (28.08.2008), URL: https://eclipse.org/articles/Article-WorkbenchSelections/article.html.
(Letzter Zugriff am 07.04.2015).

[37]: The Eclipse Foundation. Interface ITextSelection, URL: http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjface%2Ftext%2FITextSelection.html.
(Letzter Zugriff am 07.04.2015).

[38]: The Eclipse Foundation. Interface IMarkSelection, URL: http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjface%2Ftext%2FIMarkSelection.html.
(Letzter Zugriff am 07.04.2015).

[39]: The Eclipse Foundation. Interface IStructuredSelection, URL: http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjface%2Fviewers%2FIStructuredSelection.html.
(Letzter Zugriff am 07.04.2015).

[40]: P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, V. S. Tseng: SPMF. An Open-Source Data Mining Library. Example 54: Mining Frequent Sequential Patterns Using the PrefixSpan Algorithm. URL: http://www.philippe-fournier-viger.com/spmf/index.php?link=documentation.php#examplePrefixSpan.
(Letzter Zugriff am 07.04.2015).

[41]: The Eclipse Foundation. Eclipse Public License (EPL) Frequently Askes Questions, URL: https://eclipse.org/legal/eplfaq.php.
(Letzter Zugriff am 07.04.2015).

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum Unterschrift: Mehmet Fatih Cicek