Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit Nr. 226

Automatisierte Transformation von Daten aus Software Repositories und ihre Vorbereitung für Data Mining

Simon Lehmann

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. rer. Nat. Stefan Wagner
Betreuer/in:	M.Sc. Jasmin Ramadani
Beginn am:	4. Mai 2015
Beendet am:	3. November 2015
CR-Nummer:	D.2.12, E.2

Kurzfassung

Bei dem Prozess der Softwareentwicklung werden viele verschiedene Dokumente und Daten erstellt, die wichtig für das Projekt sind, aber gleichzeitig nicht zu dem Programmcode gehören. Seien es die Arbeitspakete jedes einzelnen Entwicklers, die Dokumentationen zur Einarbeitung in einzelne Themengebiete und Testfälle oder auch Metadaten der Versionsverwaltung. Diese Daten sollen eingelesen, verarbeitet und in eine Datenbank gespeichert werden, damit sie danach analysiert und ausgewertet werden können. Der Prozess der Transformation soll mithilfe von Datenströmen durchgeführt werden, ohne dass weitere Dateien erstellt werden können.

Im Rahmen dieser Bachelorarbeit wurde ein Programm entwickelt, welches die Daten von drei bestehenden Formaten erfasst, verarbeitet und abspeichert. Dazu wurde analysiert mit welchem Verfahren die Transformationen durchgeführt wurden und wie der gewünschte Ablauf funktionieren sollte. Aus diesen Informationen ist ein Konzept für das Programm entstanden und aus diesem wurde die Software entwickelt wurde.

Das komplette Programm ist bisher für Datentransformation von Metadaten eines Repositories und Datenformate wie CSV-Dateien konstruiert. Dabei werden alle Transformationen in einer MySQL-Datenbank gespeichert. Ein wichtiger Aspekt des Konzepts ist die Erweiterbarkeit. Es sollen noch weitere Formate transformiert werden können und deswegen muss das Programm leicht erweitert sein.

Abstract

In the process of software development many different documents and sets of data are created. These are important for the project but at the same time do not belong in the program code. This may be the work of each single developer or the documentation for incorporation into individual topics and test cases or metadata of the software repository. These sets of data should be read, processed and stored in a database so they can be analyzed and evaluated. The process of transformation should be executed with data streams without creating new files or other external data while the process is running.

In the context of this thesis a program was developed that captured, processed and stored data out of three different formats. For this cause it was analyzed through which method the transformation was executed and how the desired process had to work. With this information a software concept was created and out of this the software itself was developed.

The complete program was developed for the data transformation of metadata of a repository and data format like CSV-files. With this all the transformations are stored in a MySQL database. An important aspect of the plan is the extensibility. There are other formats that should be transformed and because of that the software should be easy to extend.

Inhaltsverzeichnis

1 Einleitung

In der Einleitung wird die Problemstellung dieser Arbeit aufgezeigt sowie der Aufbau dieses Dokuments.

1.1 Motivation

Bei der Softwareentwicklung wird in den meisten Fällen eine Versionsverwaltung für den Prozess der Entwicklung bereitgestellt. Somit lässt sich zu jedem Zeitpunkt auf eine alte oder auch immer wieder auf die aktuelle Version der Software zugreifen. Damit ist gewährleistet, dass keine Daten sowie Dateien, die zu dem Projekt gehören, verloren gehen können. Diese Daten- und Versionsverwaltung nennt man „Repository“. Darüber hinaus liegen in einem Repository Metadaten, die Informationen enthalten, welche zeigen, zu welchem Zeitpunkt der Entwickler die Dateien verändert hat. Jede Veränderung der Daten bewirkt eine neue Version der Software im Repository.

Diese Metadaten wurden bisher meist nur als Informationsquelle für den Projektverlauf benutzt. Dabei können diese Informationen mithilfe der richtigen Analysealgorithmen Verlaufsinformationen und Problemfälle frühzeitig aufzeigen. Denn durch Verbindungen zwischen Entwickler und Dateien und ihren mehrfache Veränderungen ist Kohärenz zwischen den Daten zu entdecken.

Für die Algorithmen müssen die Informationen vorbereitet und in eine Datenbank transferiert werden. Hier beginnt die Aufgabe dieser Bachelorarbeit. Denn die Metadaten sind bisher Rohdaten, die als Textdokument abgelegt sind. Es soll ein neues Programm entwickelt werden, welches ohne Hilfe von Dateien die Informationen in eine Datenbank übertragen. Diese Informationen sind bisher „Commit“-Daten, wobei ein „Commit“ eine Änderung eines Entwicklers im Repository ist. Das gleicht „Issue“-Daten. Diese sind Arbeitspakete eines jeden Entwicklers, welche unabhängig von einem Repository in einer Datei gespeichert sind. Als letztes sind es die Dokumentationsdaten, die, sowie die „Issue“-

Daten, in einer Datei gehalten werden. Das Programm sollte so konzipiert sein, dass aber noch weitere Informationen in die Datenbank transformiert werden können.

1.2 Aufbau des Dokuments

Diese Arbeit ist folgendermaßen gegliedert:

Kapitel 2 – Grundlagen: beschreibt den fachlichen Hintergrund der Arbeit

Kapitel 3 – Analyse: beschreibt den aktuellen und den gewünschten Umgang mit den Daten, sowie den Ablauf der Transformation

Kapitel 4 – Entwurf und Konzept: stellt die entworfenen Lösungsansätze dar

Kapitel 5 – Implementierung: beschreibt die Umsetzung der Konzepte und wie diese implementiert wurden

Kapitel 6 – Validierung: stellt die Laufzeitanalyse des Programms vor

Kapitel 7 – Zusammenfassung: beschreibt die gewonnenen Ergebnisse sowie die mögliche Weiterentwicklung

2 Grundlagen

Um alle Aspekte dieser Arbeit zu verstehen, müssen einige Begrifflichkeiten geklärt werden. Diese werden im Folgenden beschrieben, um eine Grundlage für die Arbeit zu schaffen.

2.1 Git-Repository

„Git“-Repositories sind Software-Repositories die mit dem Programm „Git“ erstellt wurden. Damit geklärt wird, was ein Repository ist und was „Git“ von anderen Versionsverwaltungen unterscheidet, werden beide Punkte näher erläutert.

2.1.1 Software-Repository

Ein Repository ist eine Versionsverwaltung für ein Software-Projekt. Darin wird jede Änderung bei der Entwicklung eines Programms festgehalten. Bei einer Änderung, welche in einem Repository als „Commit“ bezeichnet wird, werden zusätzlich zu den veränderten neuen Dateien der Name des Entwicklers, welcher die neuen Dateien einpflegt, das Datum, eine Anmerkung zu der Änderung, sowie eine Identifikationsnummer für den neuen „Commit“ gespeichert.

Der Vorteil einer Versionierung von Softwareständen ist, dass jederzeit auf einen älteren Zustand des Programms zugegriffen werden kann. Dabei können auch gleichzeitig unterschiedliche Varianten einer Version entwickelt und später wieder zusammengeführt werden. Das ganze bringt große Flexibilität in den Prozess der Entwicklung und auch Datensicherung von Varianten und Versionen.

2.1.2 Git

In unserem Fall wird das Programm „Git“ benutzt. Es ist dadurch entstanden, dass die Entwickler des Linux-Systems bis im Jahr 2005 das Versionsverwaltungsprogramm

„BitKeeper“ benutzt haben, um ihr Open-Source-Programm zu verwalten [1]. In diesem Jahr hatte die freie Gemeinschaft von Entwicklern mit der Firma, welche „BitKeeper“ entwickelt hatte, Differenzen, sodass sich die Linux Entwickler dazu entschlossen haben, ein eigenes Versionsverwaltungsprogramm zu kreieren. Die fünf wichtigsten Eigenschaften, die das neue Programm enthalten soll, waren Schnelligkeit, Einfachheit, Unterstützung von mehreren tausenden Varianten einer Softwareentwicklung, vollständig verteilte Speicherung und große Projekte effizient zu handhaben, die ihrem eigenen Linus-System gleichen.

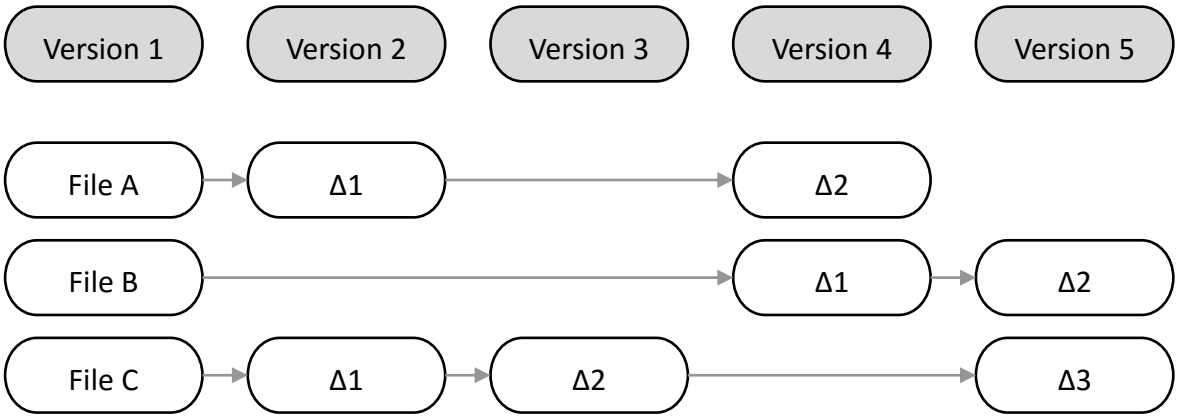

Abbildung 2.1.: Speichert Daten als Änderung der Ursprungsdatei [1]

Diese fünf Eigenschaften sind heute immer noch die Merkmale, welche „Git“ ausmachen. Um das zu verdeutlichen und die Unterschiede zu anderen Versionierungsprogrammen aufzuzeigen, wollen wir uns anschauen, wie die Versionsverwaltung aussieht. Die meisten Systeme betrachten ein zu versionierendes Projekt als eine Menge von Dateien, die im Laufe der Zeit verändert werden können. Dadurch entsteht eine umfassende Liste von Dateiänderungen.

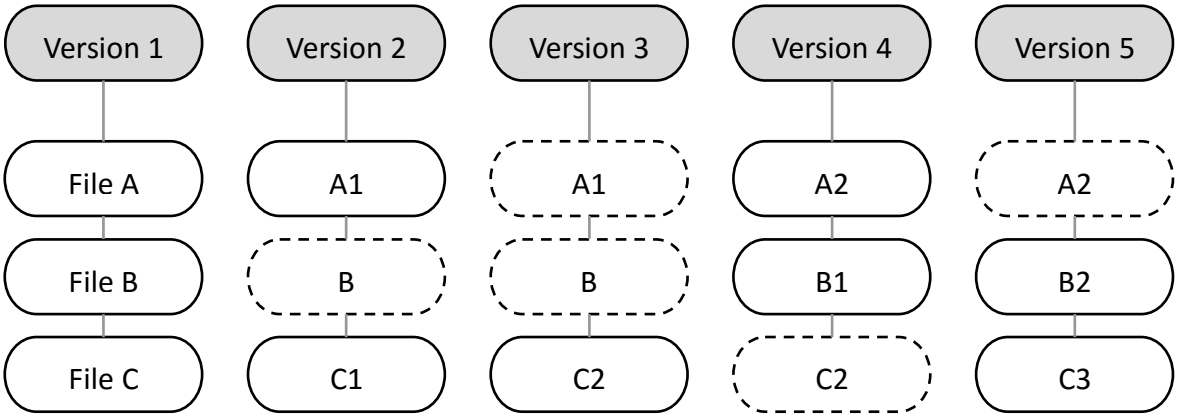

Abbildung 2.2.: Speichert jede Datei neu, oder erstellt einen Zeiger auf die vorherige Version [1]

„Git“ hingegen betrachtet die Dateien als eine Einheit jeder Änderung. Immer wenn ein „Commit“ erstellt wird, speichert „Git“ das komplette Projekt. Dabei werden Dateien, die nicht verändert wurden, nicht gespeichert, sondern es wird ein Zeiger auf die ursprüngliche Datei festgelegt. Somit entsteht hier eine Liste von Versionen.

2.2 Data Mining

Unter Data Mining versteht man allgemein die Analyse und Auswertung von Daten, im Besonderen von großen Datenmengen. Dabei wird versucht, Kohärenz zwischen den Daten zu ermitteln. Um diese Datenmengen auswerten zu können, müssen diese so konzipiert sein, dass Algorithmen die Daten durchlaufen können und Ergebnisse liefern. Dazu werden in unserem Fall alle Daten in einer Datenbank gespeichert, damit die Weiterverwendung für Data Mining unabhängig von dem Programm der Transformation zur Datenbank gestaltet werden kann.

2.3 Entity-Relationship-Model

Das „Entity-Relationship-Model“, oder auch ER-Diagramm, ist eine hochwertige Datenmodellierungstechnik [2]. Sie wurde dafür entwickelt, dass der Designer einer Datenbank diese so darstellen kann, dass seine Architektur leichter zu verstehen ist. Peter Chen hat das „entity-relationship-model“ aus den drei bis dorthin bestehenden Modellen, welche das “network model“, das „relational model“ und das „entity set model“ [3] sind, entwickelt. Ihm ist aufgefallen, dass Benutzer und Entwickler sich in Bezug auf Datenbanken oftmals uneinig sind oder Missverständnisse entstehen. Deswegen hat er diese Visualisierung.

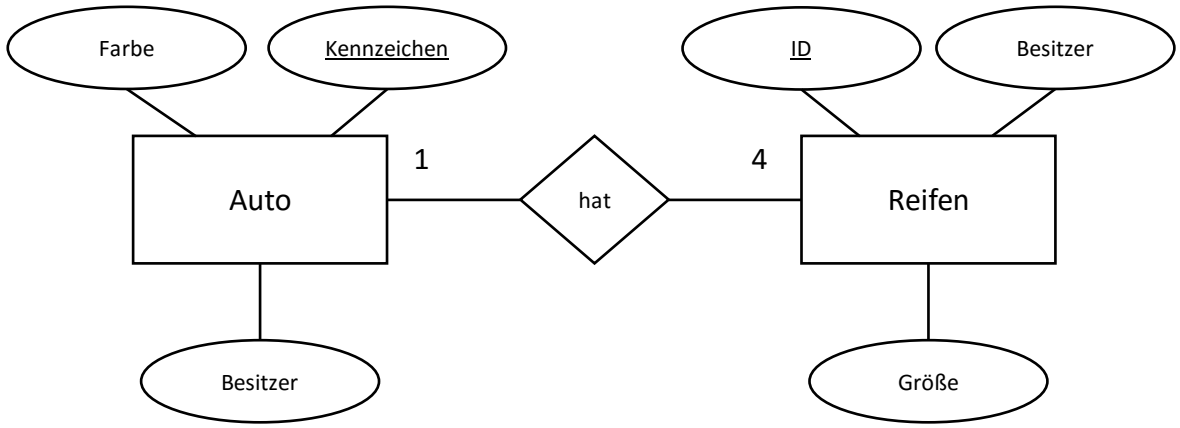

Abbildung 2.3.: Beispiel eines ER-Diagramms

Ein ER-Diagramm hat viele verschiedene Komponenten. Für das Verstehen des Datenbankkonzepts werden nur die wichtigsten benötigt. Deswegen werden nur diese hier erläutert. Das Modell besteht aus Entity-Klassen, die in Abbildung 2.3 die Komponente Auto und Reifen sind. Sie werden als Rechtecke dargestellt. Dazu besitzen sie Attribute. Das sind Ellipsen, die mit ihrer Entity-Klasse verbunden sind. Im Falle des Autos sind das die Farbe, der Besitzer und das Kennzeichen. Das unterstrichene Attribut ist der eindeutige Primärschlüssel dieser Entität. Stehen zwei Klassen in Beziehung zu einander, müssen beide mit einer Raute verbunden sein. Diese nennt man Relationstyp und sie trägt einen Namen. Das sollte immer ein Verb sein, damit die Beziehung zwischen den Entity-Klassen besser verstanden werden kann. An die Verbindungslinie der Klassen zu dem Relationstyp wird eine Kardinalität geschrieben. Sie sagt aus, wie viele Reifen ein Auto haben kann [2]. In Abbildung 2.3 könnte man somit einen Satz aus den gegeben Informationen machen: „Ein Auto hat vier Reifen“. Damit können leicht Verständnisprobleme überbrückt werden und deshalb wurde das Modell von Peter Chen entwickelt.

2.4 MySQL-Datenbank

In eine Datenbank werden Daten gespeichert. Dabei ist sie so aufgebaut, dass leicht aus einer Datenbank gelesen, geschrieben und Daten verändert werden können. Gleichzeitig können in einer Datenbank Einträge schnell gefunden und sortiert werden. Die Informationen, die gespeichert werden können, sind einfache Datentypen wie Zahlen und Zeichenfolgen, aber auch Bilder und Zeitdaten.

Die Datenbank „MySQL“ ist ein sehr schnelles, robustes, *relationales Datenbank-Managment-System* (RDBMS) [4]. Da in dieser Arbeit Daten für Data Mining vorbereitet werden müssen, muss eine Datenbank verwendet werden. „MySQL“ bietet sich, zu dem, dass sie frei verfügbar ist, an, weil sie leicht zu konfigurieren ist und auf verschiedenen Systemen funktioniert wie Windows und Linux.

2.4.1 JDBC

Das Programm „ATSR“ ist in Java geschrieben. Damit auf die MySQL-Datenbank zugegriffen werden kann, wird eine Schnittstelle zwischen den beiden Komponenten benötigt. Diese liefert die Programmierschnittstelle JDBC (Java Database Connectivity). Damit ist es möglich, aus Java heraus SQL-Befehle zur Datenbank zu senden und die Resultate zu erhalten [5].

3 Analyse

In der Analysephase wird der aktuelle Zustand des Systems, sofern es bereits ein System gibt, oder auch der aktuelle Zustand des Verfahrens, betrachtet und ausgewertet. Darüber hinaus wird das vom Kunden gewünschte System definiert.

3.1 Ist Zustand

Bisher werden alle Daten, die später für Data Mining gebraucht werden, von „Hand“ vorbereitet. Die Log-Daten, welche die Metadaten eines Repositories sind, werden durch die Befehle in der Kommandozeile in eine Datei ausgelagert und danach einzeln vorbereitet. Das Ganze funktioniert bei kleineren Softwareständen ohne weitere Probleme, allerdings benötigt man dafür viel Zeit. Bei großen Repositories ist es dagegen nicht mehr möglich, die Auswertung sowie die Vorbereitung für Data Mining auszuführen. Es gibt bisher noch keinen automatisierten Ablauf, um die Daten soweit vorzubereiten, um damit direkte Ergebnisse erlangen zu können.

3.2 Soll Zustand

Das zu implementierende Programm soll Metadaten eines Software-Repositories vorbereiten und diese in eine Datenbank schreiben. Der komplette Ablauf der Datentransformation soll nicht durch Dateienübertragung funktionieren, sondern mithilfe von „Streams“, also Datenströmen, sodass Daten nur durch einen „Stream“ eingelesen, verarbeitet und vorbereitet und zum Schluss durch einen „Stream“ in die Datenbank geschrieben werden. Dadurch sollte die Transformation eine gute Performanz haben sowie wenig Speicher verbrauchen. Ein Benutzer soll den geringsten Aufwand haben, um diese Transformation in Gang zu bringen. Deswegen wird der Vorgang mit nur einem Knopfdruck ausgeführt. Des Weiteren lassen sich bestimmte Arten von CSV-Dateien mit dem Programm transferieren. Zu Beginn sind diese Formate „Issue“-Dateien und „Docu“-Dateien. In der

Datei „Issues“ stehen alle Arbeitspakete der Entwickler für ein Projekt. Die Datei „Docu“ enthält die Pfade zu den Dokumentationen des Projekts. Alle Informationen sind am Ende in der Datenbank enthalten. Dazu soll die Datenbank einen Aufbau haben, der für das spätere Data Mining eine gute Performanz hervorbringt. Dazu müssen Backups für ältere Versionen, sowie andere Projekte die transformiert wurden, erstellt und gespeichert werden.

3.3 Datenformate der Transformation

Die Daten, welche für Data Mining vorbereitet werden müssen, sind unterschiedlich komplex für die Transformation. Das liegt daran, dass die Daten aus unterschiedlichen Quellen bezogen werden. Dieses Kapitel erläutert die Unterschiede, sowie jedes Datenformat in sich selbst.

3.3.1 Commit

Ein „Commit“ ist wohl das komplexeste Format. Das liegt daran, dass „Commits“ durch Log-Protokolle abgerufen werden müssen. Da wir das Versionsverwaltungsprogranmm „Git“ benutzen und dies ein eigenständiges Programm, ist muss ein Befehl gesendet werden, der alle „Commit“-Daten abruft. Für das spätere Data Mining sind aber nicht alle Informationen wichtig, deswegen muss nur ein Teil davon abgerufen werden. Das sind die Identifikationsnummer, der Name des Entwicklers, welcher diesen „Commit“ erstellt hat, das Datum, wann der Commit erstellt wurde und eine Anmerkung vom Entwickler dazu. Danach benötigen wir alle veränderten Dateien zu dem „Commit“. In Abbildung 3.1 werden alle Informationen beispielhaft angezeigt. Dafür wurde ein Log-Befehl an „Git“ gesendet und ein „Commit“ als Beispiel dargestellt.

```
5486558#Entwickler XYZ#Mon Dec 1 12:13:00 2014 +0100
#created astpa.extension
astpa.extension/.project
astpa.extension/META-INF/MANIFEST.MF
astpa.extension/build.properties
astpa.extension/plugin.xml
astpa.extension/pom.xml
astpa.extension/schema/stepedProcess.exsd
```

Abbildung 3.1.: Beispiel eines Log-„Commits“

3.3.2 Issue

Alle „Issues“ eines Projekts stehen in einer CSV-Datei. Sie sind dort aufgelistet und ein „Issue“ hat 22 Parameter. Davon benötigen wir für das spätere Data Mining nur drei, Status, Tracker und die Beschreibung. Ein „Issue“ bekommt in der Datenbank eine eindeutige Identifikationsnummer. Diese wird aber nicht von „Issue“-Dateien, sondern intern definiert.

3.3.3 Docu

Das Format des Docu-Dokuments ist noch nicht festgelegt, aber der Pfad und die Beschreibung sind als Parameter eines Docu-Eintrags enthalten. Sonstige Angaben können dazu noch nicht gemacht werden.

4 Entwurf und Konzept

In diesem Kapitel wird das entwickelte Konzept dieser Arbeit beschrieben. Dazu gehört die Architektur des Programms, sowie einzelne Aufgaben der „ATSR“, die im Vorhinein geplant und ausgearbeitet werden müssen.

4.1 Architektur

Die Idee hinter der Architektur ist, dass das Programm leicht erweiterbar und veränderbar ist und bleibt, denn es wird in näherer Zukunft weitere Transformationen durchführen müssen. Um diesen Ansprüchen zu genügen, wird das „Model-View-Controller“-Konzept als Architekturmodell benutzt.

4.1.1 Model-View-Controller Konzept

Beim „Model-View-Controller“-Konzept, auch „MVC“-Konzept genannt, gibt es drei separate Teile, in die das Programm unterteilt wird, siehe Abbildung 4.1. Alle haben unterschiedliche Aufgaben im Programm selbst und können, dadurch, dass sie untereinander getrennt sind, leicht ausgetauscht werden.

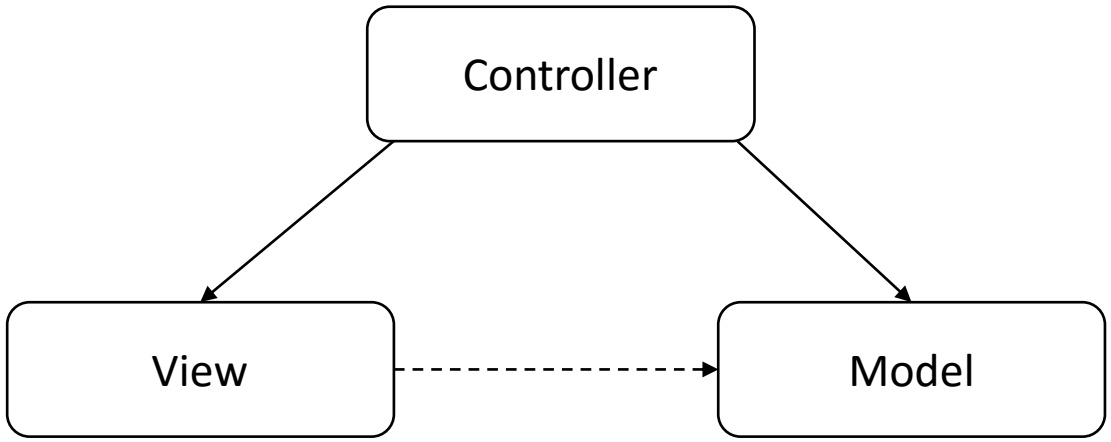

Abbildung 4.1.:Darstellung des Model-View-Controller Konzepts

Das erste ist die Präsentationsebene. Bei diesem Konzept „View“ genannt, ist alles enthalten, was zur Darstellung gehört. In unserem Fall ist dies im Hauptsächlichen die Oberfläche für den Benutzer, um das Programm einfach und handlich zu bedienen. Im optimalen Fall „weiß“ die Oberfläche nichts und reagiert nur auf Eingaben des Benutzers. Es kann aber durchaus sein, dass die Präsentationsebene eine Klasse aus dem „Model“ kennen muss für eine leichtere Darstellung von Daten.

Das „Model“ hält alle möglichen Arten von Daten, was bedeutet, dass keinerlei Funktionalitäten darin stecken. Die Modellebene ist als einzige unabhängig von den anderen beiden Teilen und ist deswegen sehr leicht zu verwalten.

Die Steuerungsebene, hier „Controller“ genannt, ist wohl die komplizierteste der drei Schichten. Sie ist zuständig für die Logik in dem Programm selbst. Wird über die Oberfläche eine Eingabe getätigt und vom Benutzer eine Aktion des Programms verlangt, leitet die „View“ die Anforderung direkt weiter an den „Controller“. Da die Präsentationsschicht keine Verbindung zur Steuerungsebene hat, gibt es von dem „Controller“ mehrere Beobachtungsmethoden, welche direkt angesprochen werden, falls die Benutzeroberfläche betätigt wird. Dadurch können die Aktionen und Vorgänge leicht geändert werden.

4.1.2 Erweiterbarkeitskonzepte

Das gesamte Programm dient als Hilfsprogramm für die Verarbeitung und Vorbereitung von Daten. Da es noch ganz am Anfang seiner Entwicklung steht, werden mit Sicherheit noch einige Änderungen vorgenommen werden. Da die Erfahrungen mit dem Programm weitere Erkenntnisse liefern, werden sie zeigen, in welchem Aufgabenfeld es noch verwendet werden kann. Deswegen soll die Erweiterbarkeit ein wichtiger Aspekt der Arbeit sein.

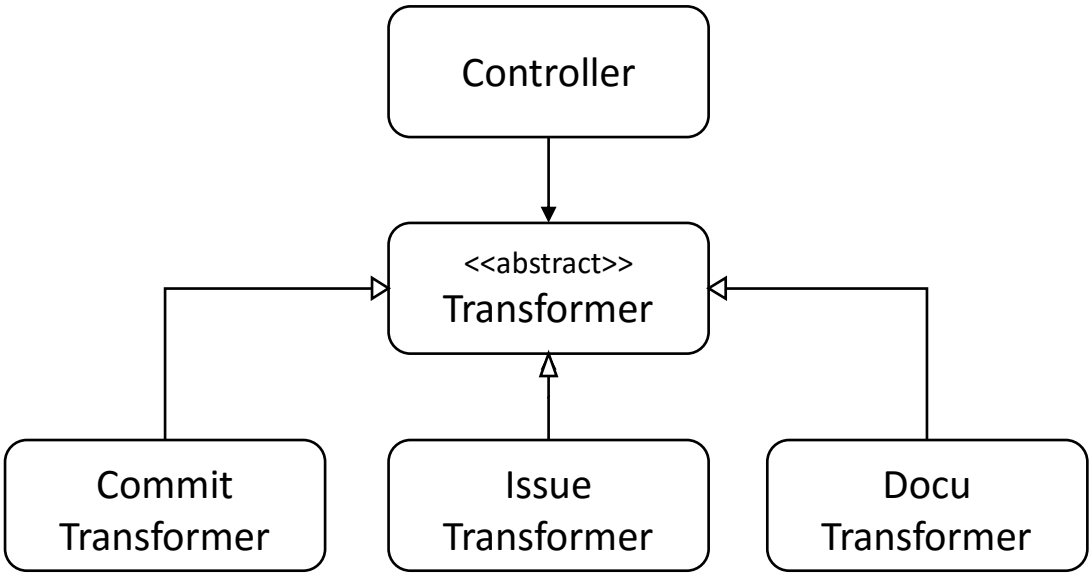

Abbildung 4.2.: Konzept der Transformatoren

Für das Programm müssen deswegen zwei Funktionalitäten besonders betrachtet werden. Dies ist zum einen die Transformation der Metadaten, denn das sind bis jetzt die „Commits“ aus den Log-Daten des Repository und die „Issues“ und „Docus“ aus den CSV-Dateien. Dort kann sich immer wieder etwas ändern und noch eine Transformation hinzukommen. Deswegen gibt es eine abstrakte Klasse mit Namen „Transformer“, von welcher alle Transformationsklassen abstammen müssen. Dadurch ist gegeben, dass jede „Transformer“-Klasse bereits den richtigen „DatabaseWriter“ anspricht und auch der Ablauf einer Transformation korrekt ist.

Der zweite Punkt ist das Schreiben in die Datenbank. Es ist möglich, dass eine andere Datenbank verwendet werden soll. Bisher ist die verwendete Datenbank eine MySQL-Datenbank. Die abstrakte Klasse „Transformer“ kennt nur das „Interface“ für die Datenbankbeschreibung. Die Implementierung ist somit leicht ersetzbar und wieder verwendbar.

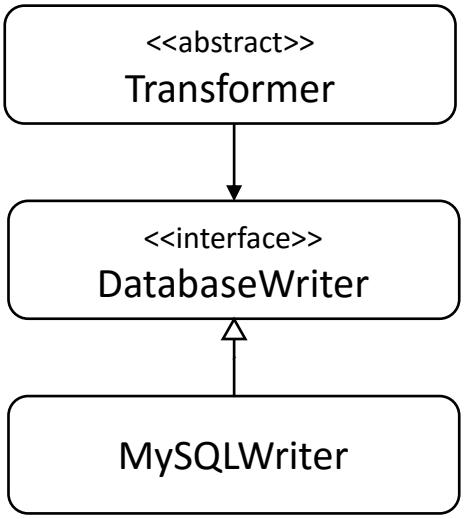

Abbildung 4.3.: Konzept der Architektur von Komponenten zur Beschreibung der Datenbank

Das ganze erlaubt deswegen eine einfache Erweiterbarkeit der beiden Funktionalitäten. Dadurch ist auch eine große Flexibilität gewährleistet, um die Dateien zu ergänzen oder zu verändern. Dazu wird der Code nicht so oft dupliziert und wenn an einer Stelle etwas verändert wird, muss dies nicht noch an mehreren anderen Stellen getan werden.

4.2 Datenbankkonzept

In der Datenbank werden alle transformierten Daten abgelegt. Da die Daten für Data Mining vorbereitet sein sollen, muss die Struktur der Datenbank für die Weiterverarbeitung ausgelegt sein. Bei den Tabellen für die „Issues“ und „Docus“ ist dies leicht gehandhabt, denn beide benötigen nur eine Tabelle mit ihren Werten siehe Abbildung 4.4.

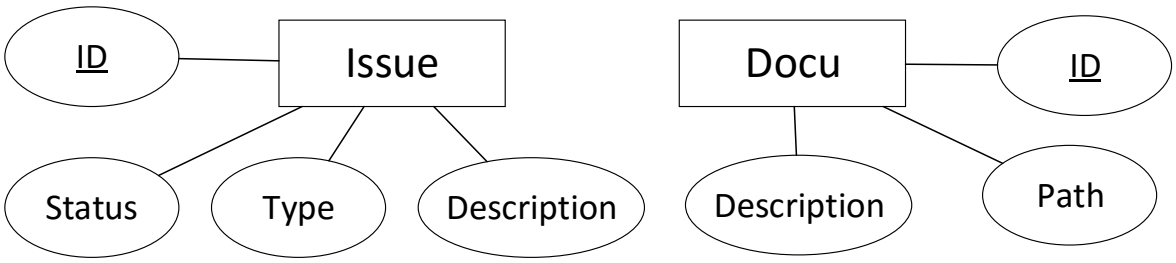

Abbildung 4.4.: ER-Diagramm für "Issue"- und "Docu"-Tabellen

Die „Commit“-Daten benötigen mehrere Tabellen, da sie aus mehreren Einheiten zusammengesetzt ist. Das Hauptgerüst ist der „Commit“ mit Identifikationsnummer, Name des Autors, das Erstellungsdatum und die zugehörige Nachricht. Diese vier Parameter können in eine Tabelle untergebracht werden, wobei die Identifikationsnummer der Primärschlüssel ist, wie in Abbildung 4.5 zu sehen. Ein „Commit“ hat aber auch noch Dateien die zu ihm gehören. Eine Datei kann aber auch zu mehreren „Commits“ gehören. Deswegen muss hier ein Relations-Typ eingefügt werden. Daraus entstehen drei Tabellen. Die Tabelle der Dateien, der „Commits“ und die Verbindungstabelle in dieser stehen der anderen Tabellen Primärschlüssel falls es eine Relation gibt.

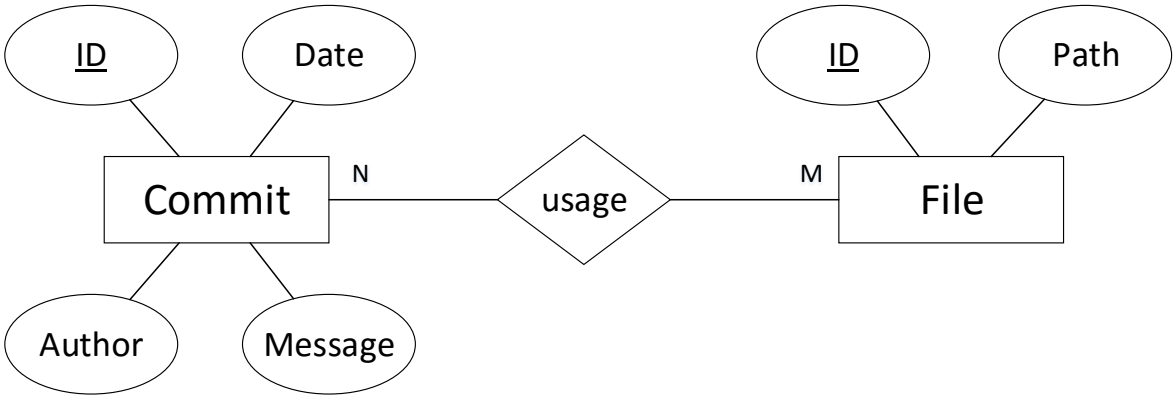

Abbildung 4.5.: ER-Diagramm der "Commits"

4.3 Transformationskonzept

Die Transformation soll bei jedem Datenformat gleich ablaufen. Dafür wird sie in drei Schritte unterteilt. Es beginnt mit dem Erfassen der Daten. danach werden die Daten verarbeitet und zum Schluss in der Datenbank gespeichert.

4.3.1 Datenerfassung

Die Daten können aus verschiedenen Dateien und Strukturen erfasst werden. Bis jetzt gibt es zwei Quellen. Dazu gehören ein Repository und CSV-Dateien. Bei beiden können die nötigen Informationen mithilfe eines Streams erhalten werden. Dies ist einer der wichtigen Aspekte dieser Arbeit, denn es soll eine Datenerfassung sein, ohne zusätzliche Dateien sowie andere externe Formate zu erstellen. Das zu entwickelnde Programm soll bei der Transformation von allein diesen Stream öffnen und die gewünschten Daten abrufen. Die nötigen Eingaben vom Benutzer sollen schon davor im Programm eingegeben sein.

4.3.2 Datenverarbeitung

Bei der Datenverarbeitung geht es darum, die erhaltenen Daten aufzuteilen und in ein Format zu packen, das das Interface, welches für die Speicherung in die Datenbank zuständig ist, versteht. Da die Daten sehr unterschiedlich sein können, soll jedes Format seine eigene Verarbeitung bekommen. Dadurch kann flexibel auf die jeweiligen Anforderungen eingegangen werden. Da es bisher drei verschiedene Formate gibt, wie in Absatz Datenformate der Transformation erwähnt, sollen drei Transformationen mit ihren Verarbeitungen zunächst entwickelt werden.

4.3.3 Datenspeicherung

Die Speicherung verläuft, was auch hier wieder bei jedem Format gleich ist, nach einem bestimmten Schema. Das beschreiben führt die implementierte Klasse des Interfaces „DatabaseWriter“ durch. Der Ablauf, siehe **Fehler! Verweisquelle konnte nicht gefunden**

erden., startet mit dem Auftrag, die Datenbank zu beschreiben. Zuerst wird überprüft, ob die Tabellen, die für die gerade laufende Transformation benötigt werden, bereits existieren.

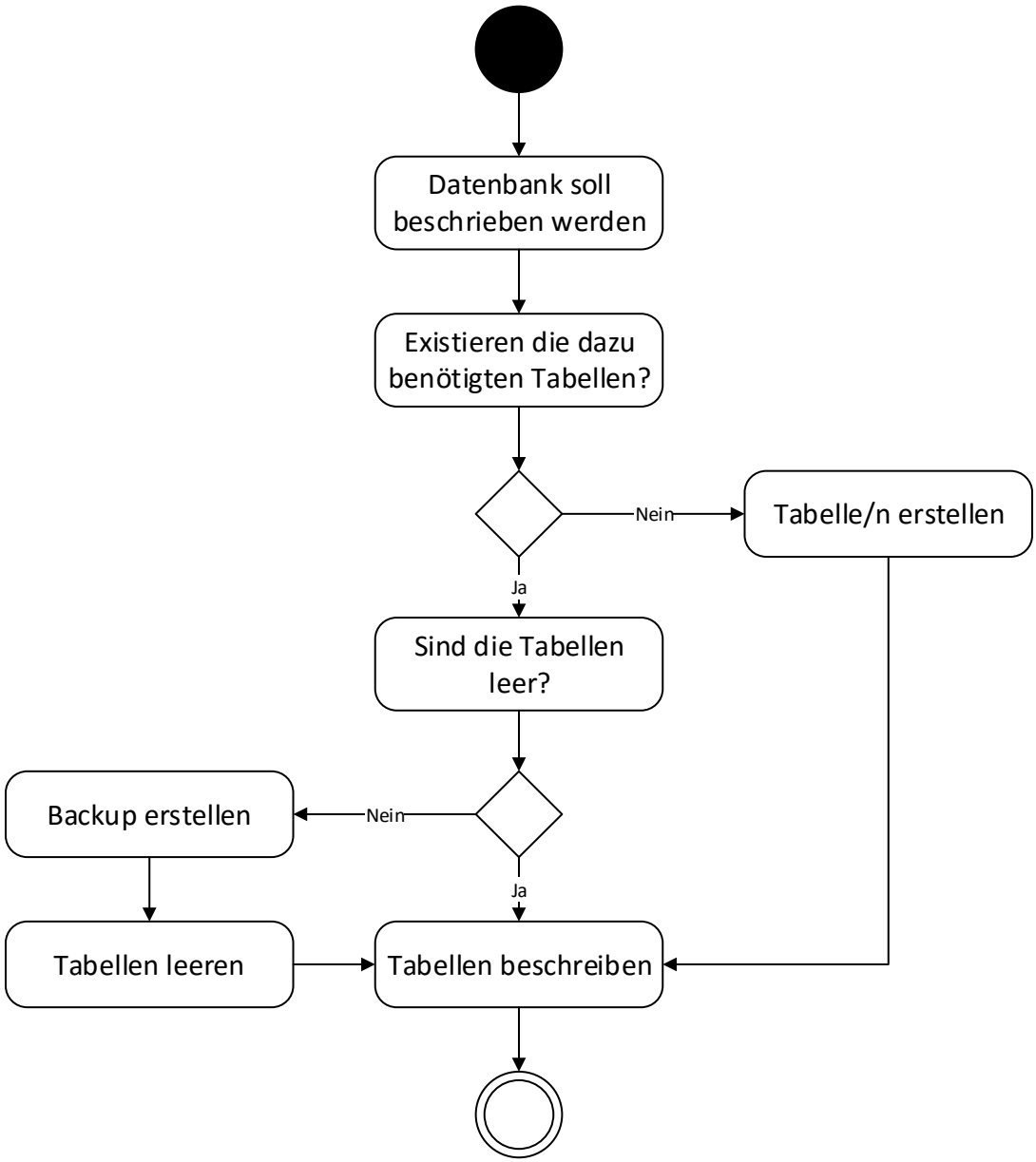

Abbildung 4.6.: Ablauf der Speicherung von Daten in die Datenbank

Ist dies nicht der Fall, sollen die Tabellen erstellt und beschrieben werden. Damit wäre die Arbeit abgeschlossen. Gibt es die benötigten Tabellen bereits, muss überprüft werden, ob sie leer sind. Falls sie leer sind, können sie genauso beschrieben und die Transformation beendet werden. In den meisten Fällen sind die Tabellen aber nicht leer, sondern von vorherigen Transformationen gefüllt. Dann muss ein Backup der bereits existierenden Datenbanktabellen erstellt werden. Damit die Tabellen unterscheidbar sind, bekommen sie,

sobald aus ihnen eine Backup-Tabelle wird, einen Zeitstempel des aktuellen Datums und Uhrzeit. Der Zeitstempel wird im Namen eingetragen. Danach kann die Transformationstabelle geleert und neu beschrieben werden. Somit sind alle möglichen Zustände der Datenbank abgedeckt und es können keine Fehler bei der Speicherung bezüglich der Tabellen entstehen.

5 Implementierung

In diesem Kapitel wird die Implementierung des Programms beschrieben. Dies kann in vier Unterpunkte unterteilt werden. Dazu gehört die Benutzeroberfläche. Das betrifft die Schnittstelle zwischen Benutzer und dem Programm selbst. Die Verwaltung und die Handhabung der Einstellungen ist ein weiterer Abschnitt. Der wichtigste Punkt wird aber die Transformation beinhalten, da dies die Hauptfunktion des „ATSR“ ist. Und zum Schluss wird die Datenbankverbindung erläutert.

5.1 Benutzeroberfläche

Die Benutzeroberfläche besteht aus mehreren Komponenten. Diese sind die Hauptansicht, die Ansicht um die Einstellungen zu ändern oder auch einzusehen und mehrere Benachrichtigungen für den Benutzer. Dazu gehört zusätzlich die Verbesserung der Benutzbarkeit.

5.1.1 Hauptansicht

Beim Starten des Programms gelangt der Benutzer zur Hauptansicht, welche im Programmcode „TransformationView“ genannt wird. Das „TransformationView“ ist ein „JFrame“ und wird zur Anzeige von Elementen und Komponenten verwendet.

In Abbildung 5.1 ist die Ansicht in ihre fünf Teile aufgegliedert. Am oberen Rand wird die Menüleiste (a) angezeigt. Von dort aus kann das Programm beendet oder in das Menü gewechselt werden, um die Einstellungen zu ändern. Direkt darunter ist die Anzeige für den Zustand der Datenbankverbindung (b). Ist diese, wie in der Abbildung, grün und die Schrift lautet „connected to database“, dann ist eine Verbindung zu der Datenbank, welche in die Einstellungen parametriert wurde, möglich. Falls dies nicht möglich ist, ist das Feld rot und lautet „not connected to database“. Es bedeutet, dass mit den gegebenen Einstellungen keine Verbindung zu einer Datenbank hergestellt werden kann. Die nächste Komponente der

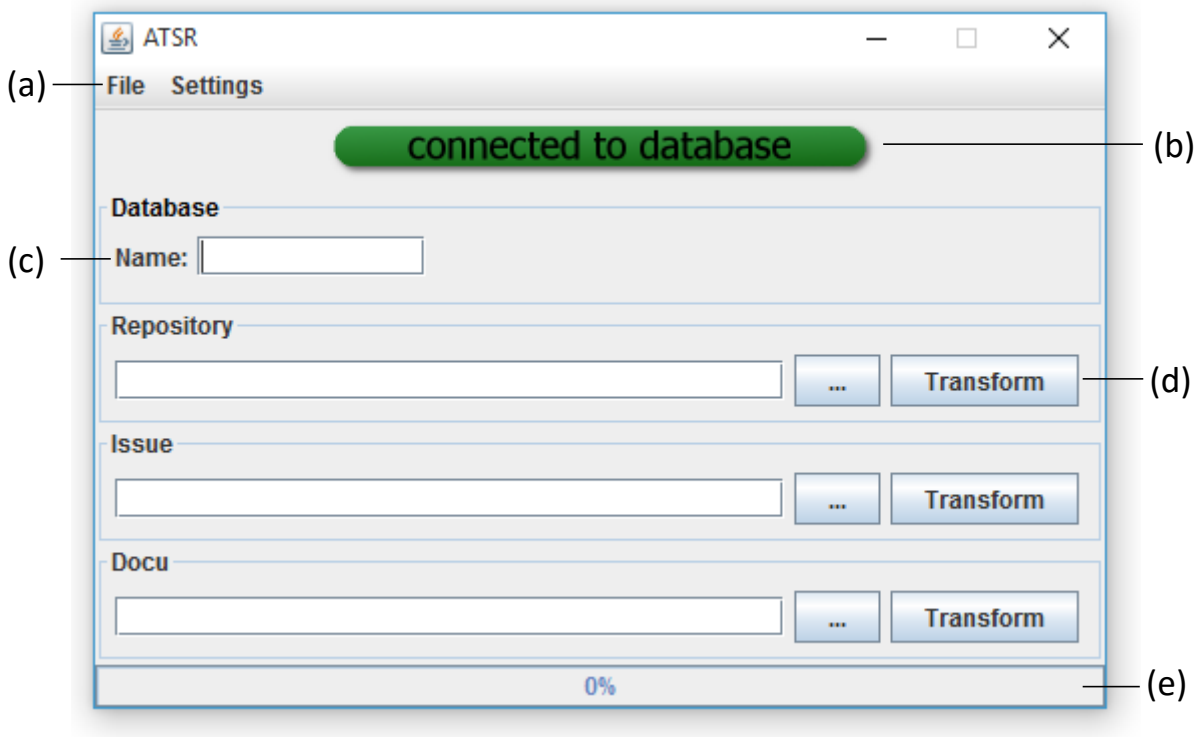

Abbildung 5.1.: Hauptansicht des Programms „ATSR“

Hauptansicht ist das Eingabefeld des Datenbanknamen (c). Dort kann der Name eines bestehenden Datenbankschemas eingetragen werden oder auch ein neues Schema erstellt werden, in dem ein Name eines Datenbankschemas eingetragen wird, das es noch nicht gibt. Der größte Bereich sind die Transformationsfelder (d). Jedes einzelne steht für ein bestimmtes Format aus dem Abschnitt 3.3 „Datenformate der Transformation“. Die Funktionalitäten hinter den Buttons, mit der Aufschrift „…“, die angelegt sind, um die Dateien auszuwählen, sind bei allen gleich. Nur die Datei-Art, welche ausgewählt werden kann, ist unterschiedlich. Für ein Repository muss ein Ordner eingefügt werden. Dieser muss alle Daten des „Git“-Repositories enthalten. In Abbildung 5.2 ist das Projekt „A-STPA“

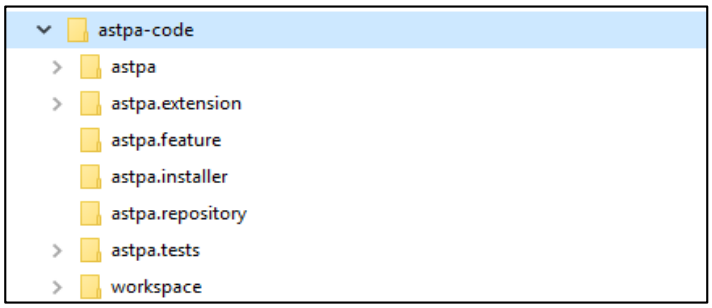

Abbildung 5.2.: Ordnerstruktur eines Software-Repository

dargestellt. In diesem Fall müsste der Pfad des Ordners, welcher in der Abbildung blau hinterlegt ist, in das Textfeld eingetragen werden.

Bei den „Issues“ und „Docus“ sind es CSV-Dateien. Bei den Buttons für die Transformation wird die des entsprechenden Formats ausgeführt. Solange das Programm arbeitet und eine Übertragung, Verarbeitung und Speicherung neuer Daten durchführt, können keine neuen Transformationen gestartet werden. Damit der Benutzer mitbekommt, wie weit die Arbeit fortgeschritten ist, wird in der Fortschrittsanzeige (e) prozentual angezeigt, wie weit der Prozess durchgeführt wurde.

5.1.2 Einstellungsansicht

Die Ansicht für die Einstellungen ist ein „JDialog“. Das bedeutet, es ist ein untergeordnetes Fenster der Hauptansicht. Von hier aus können Werte geändert werden und nach dem Schließen des Fensters werden die Änderungen, je nach Buttonbetätigung, übernommen oder verworfen. In dem Feld „Database“, wie in Abbildung 5.3 zu sehen ist, werden alle Parameter zur Datenbankverbindung angezeigt und sind veränderbar. Um eine Verbindung herzustellen, wird ein Benutzername mit einem passenden Passwort benötigt und eine IP-Adresse mit dem richtigen Port. Damit kann eine Verbindung zu einer Datenbank aufgebaut werden, ohne direkt auf ein Schema oder eine Tabelle zuzugreifen. Der untere Bereich des

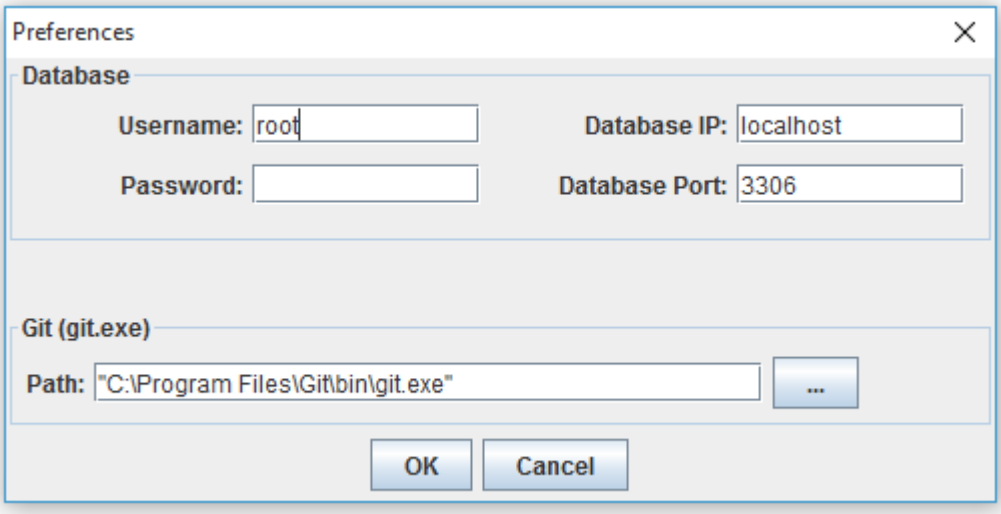

Abbildung 5.3.: Einstellungen mit Standardwerten

Fensters ist für die Einstellungen von „Git“. Um die Log-Daten von „Git“ abrufen zu können, ist der Dateipfad der EXE-Datei erforderlich. Dieser muss hier eingetragen sein. Für alle fünf Werte der Einstellungen, außer dem Passwort, gibt es Standardwerte. Das soll dem Benutzer helfen zu verstehen, was in den Feldern eingetragen werden kann, aber auch Arbeit ersparen, falls er selbst alle anderen Programme standardmäßig konfiguriert hat. Um die Änderungen zu bestätigen oder aufzuheben, müssen die beiden Buttons am unteren Rand betätigt werden. Wird der Bestätigungsbutton gedrückt, werden die Einstellungsparameter nach 5.2 „Verwaltung der Einstellungen“ gesichert. Bei dem Button um abzubrechen, werden alle Änderungen verworfen. Nach dem Schließen kommt der Benutzer wieder auf die Hauptansicht des „ATSR“ zurück.

5.1.3 Benachrichtigungen

Durch Betätigen von Buttons werden Transformationen oder andere Aktionen ausgeführt. Hier kann es auch vorkommen, dass der Benutzer Fehler bei den Eingaben gemacht hat oder außerhalb des Programms, also im System, unerwartete Fehler aufgetreten sind. Damit der Anwender weiß, warum ein Problem aufgetreten ist, gibt es verschiedene Arten von Benachrichtigungen. Dazu gehören Fehlermeldungen und informative Meldungen, die in diesem Abschnitt erläutert werden. Alle Nachrichten haben einen Bestätigungsbutton um die Meldung zu schließen. Danach muss der Benutzer selbst das Problem lösen.

Informationsmeldungen

Diese Meldungen werden angezeigt, wenn der Benutzer etwas vergessen hat auszufüllen oder er auf etwas Bestimmtes hingewiesen werden soll, ohne dass bisher ein Fehler gemacht wurde.

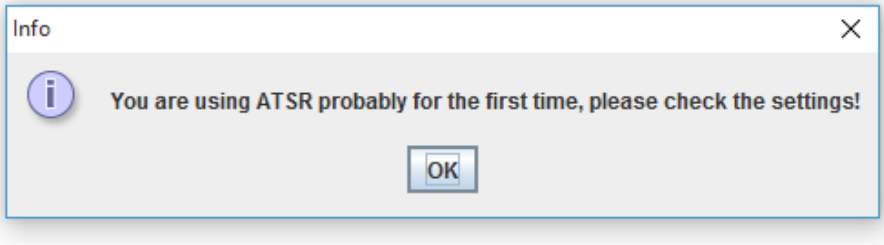

Abbildung 5.4.: Nachricht beim ersten starten des "ATSR"

Wenn das Programm das erste Mal auf dem System gestartet wird, wird, wie in Abbildung 5.4 zu sehen ist, eine Nachricht für den Anwender angezeigt. Darin wird darauf hingewiesen, dass das Programm zum ersten Mal gestartet wird und deswegen die Einstellungen überprüft und ausgefüllt, werden bevor eine Transformation gestartet wird. Diese Mitteilung wird direkt am Beginn angezeigt, weil im Normalfall die Standardwerte in den Einstellungen abgespeichert sind und diese in den meisten Fällen nicht die richtigen Parameter für die Datenbankverbindung oder des „Git“-Pfad sind.

Die Informationsmeldungen, dass ein Eintrag fehlt, sind nur für die Hauptansicht. Dort wird mitgeteilt ob das Feld der auszuführenden Transformation leer ist. In Abbildung 5.5 ist die Meldung über das Fehlen des Repository-Pfades zu sehen. Des Weiteren kann im Startfenster das „TransformationView“, das Textfeld für den Namen der Datenbank, nicht beschrieben sein. Auch hier gibt es eine kurze Benachrichtigung.

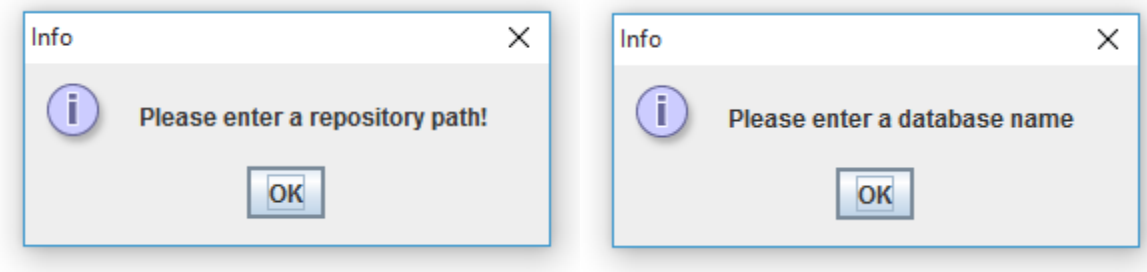

Abbildung 5.5.: Meldungen wenn Felder ausgelassen wurden

Fehlermeldungen

Fehlermeldungen sind Probleme, bei denen der Benutzer zusätzlich eingreifen muss, um das Problem zu beheben. Davon gibt es drei Arten, die nur beim Starten einer Transformation auftreten können. In Abbildung 5.6 sind die jeweiligen Arten dargestellt.

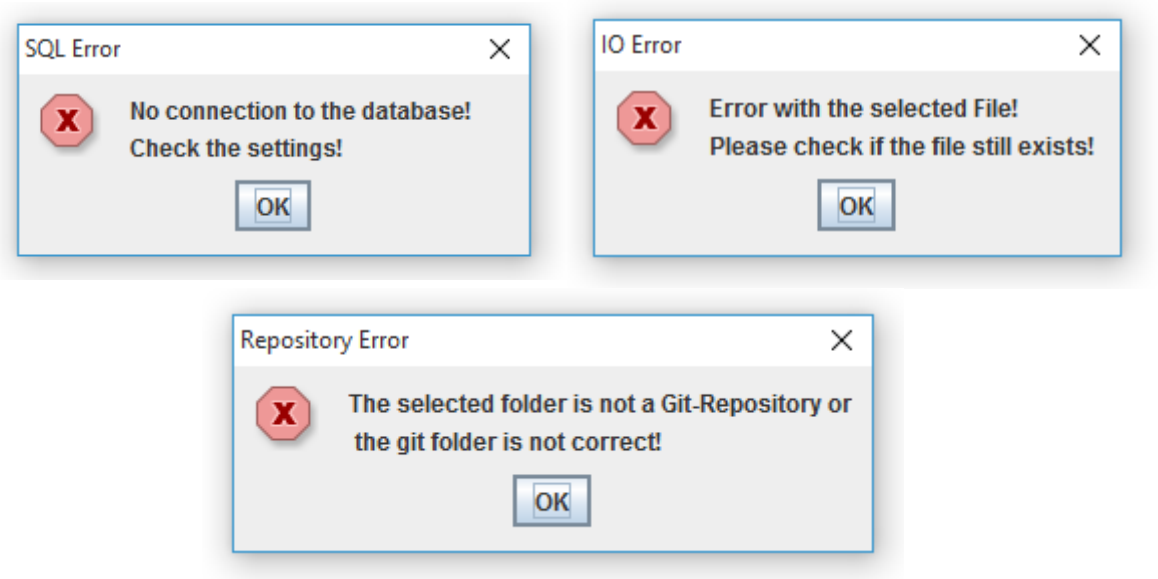

Abbildung 5.6.: Fehlermeldungen des Programms "ATSR"

Ein Repository Error ist der einzige, der nur bei der Transformation von „Commits“ auftreten kann. Dieses Problem tritt auf, wenn der Ordner, um die Daten zu erfassen, nach einem „Git“-Befehl, der sich alle „Commit“-Daten auslesen lässt, keine Werte in der Kommandozeile, wie auch in Abschnitt 5.3 „Transformation“ erläutert wird, schreibt. Denn nur wenn in dem Fall Daten ausgegeben werden, ist der Ordner ein „Git“-Repository, andernfalls ist es nur ein Ordner des Systems.

Die beiden anderen Fehlermeldungen können bei jeder Transformation entstehen, denn zum einen bekommt man einen SQL Error, wenn mit der Datenbankverbindung oder bei der Beschreibung der Datenbank ein Problem entstanden ist. Dies kann verschiedene Gründe haben. Von dem Verlieren der Datenbankverbindung während des Schreibens, bis hin zu fehlerhaften Daten. Zum anderen gibt es IO Errorsm, die immer dann auftreten können, wenn eine Verbindung zu einem Format über einen Datenstrom hergestellt wird. In unserem Fall sind das Dateien, die per „Inputstream“ eingelesen werden und bei Repositories wird

das Programm „Git“ als Verbindungsglied verwendet, welches die Daten über einen Datenfluss in unser Programm einspeist.

5.1.4 Benutzbarkeit

Durch die vielen Eingabefelder wird das Programm für den Einstieg sehr schwierig zu verstehen. Darum mussten Methoden erarbeitet, werden um die Benutzbarkeit des Programms zu erhöhen. Nach dem Buch Usability engineering gibt es fünf wichtige Attribute zur Benutzbarkeit eines Programms [6]. Als ersten Punkt wird die Erlernbarkeit aufgezählt. Das heißt, die Software soll einfach zu verstehen sein, damit der Benutzer keine größere Einarbeitungszeit benötigt. Der zweite Punkt ist die Effizienz hinsichtlich des Anwenders. Dabei ist wichtig, dass, wenn der Benutzer verstanden hat, wie das Programm funktioniert, die gewünschten Funktionalitäten schnell ausgeführt werden können. Die Einprägsamkeit ist der dritte Punkt auf der Liste wichtiger Attribute der Benutzbarkeit. Bei diesem Begriff ist die Schnelligkeit gemeint, sich an Erlerntes, im Hinblick auf die Software, zu erinnern. Auch wenn ein längerer Zeitraum vergangen ist. Als vierten Punkt werden Fehler angesprochen. Hier soll die Fehlerrate des Benutzers niedrig gehalten werden und leicht händelbar sein, damit der Benutzer seine eigenen Fehler leicht korrigieren kann. Der letzte Punkt ist die Zufriedenheit, was einen sehr subjektiven Aspekt der Benutzeroberfläche anspricht. Der Anwender soll sich wohl fühlen wenn er das Programm benutzt.

Um jedem dieser Benutzbarkeitsattribute zu entsprechen, hat das Programm „ATSR“ bestimmte Funktionen oder Komponenten, die dem Benutzer helfen und das Benutzten erleichtern sollen. Beim ersten Starten der Software wird eine Benachrichtigung angezeigt, die besagt, dass der Anwender die Einstellungen überprüfen soll, bevor er seine erste Transformation startet. Das betrifft den ersten Punkt der Benutzbarkeit, die Erlernbarkeit eines Systems. Für diesen Aspekt sind im „ATSR“ noch weitere Hilfen eingebaut. Zunächst gibt es in der Hauptansicht eine Anzeige, wie in Abbildung 5.7 zu sehen ist, wobei der Benutzer direkt erkennen kann, ob eine Verbindung zur Datenbank besteht oder nicht. Falls die rote Anzeige mit „not connected to database“ angezeigt wird, muss der Anwender zuerst in den Einstellungen die Parameter ändern oder neu eintragen, damit eine Verbindung zur

Abbildung 5.7.: Anzeige zur Datenbankverbindung in der Hauptansicht

Datenbank aufgebaut werden kann. Auch hier sind Hilfestellungen für den Benutzer angebracht. Zum einen sind hinter den Textfeldern der Datenbank Symbole angebracht. Bei jeder Änderung der Angaben in den Feldern werden die Parameter überprüft. So weiß der Benutzer zu jedem Zeitpunkt, ob seine Eingaben richtig waren oder nicht. Dazu ist eine textliche Information unterhalb der Felder angegeben. Somit kann der Benutzer genau identifizieren, wo ein Fehler in den Angaben ist. In Abbildung 5.8 wurde ein falsches Passwort eingetragen. Da von den vier Parametern immer zwei verknüpft, sind werden

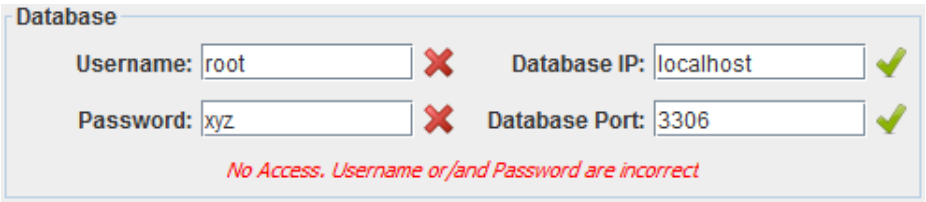

Abbildung 5.8.: Einstellungen mit falschem Passwort

immer beide Felder als Fehlerhaft angezeigt und der entsprechende Text unten angefügt. Die Effizienz des Programms wird dadurch erhöht, dass mehrere Parameter gespeichert werden und beim erneuten Starten diese wieder eingebunden werden. Somit muss der Benutzer seine Angaben nicht immer wieder neu einfügen sondern kann direkt eine Transformation, starten sofern er beim letzten Bedienen alle notwendigen Textfelder ausgefüllt hat. Dazu gehören alle Parameter, die in der Ansicht für die Einstellungen eingetragen werden können. Sobald in diesem Fenster die Werte bestätigt werden, werden sie abgespeichert. Der einzige weitere Eintrag ist der Name der Datenbank. Dieser wird nach jeder Transformation gesichert. Wie das Sichern der Werte funktioniert, wird in 5.2 Verwaltung der Einstellungen erklärt. Für den dritten Punkt, die Einprägsamkeit, gibt es keine spezielle Funktion, aber dadurch, dass der Benutzer an vielen Stellen Rückmeldungen über seine Eingaben bekommt und die meisten Werte die er eingibt gespeichert werden, wird dieser Aspekt gleichzeitig Unterstützt. Um die Fehler möglichst gering zu halten, gibt es entsprechende Mitteilungen. Diese wurden im vorherigen Abschnitt 5.1.3 beschrieben. Sie sind dafür da, um dem Benutzer, zusätzlich zu den Symbolen, Rückmeldung über Fehleinträge zu geben. Des Weiteren bekommt der

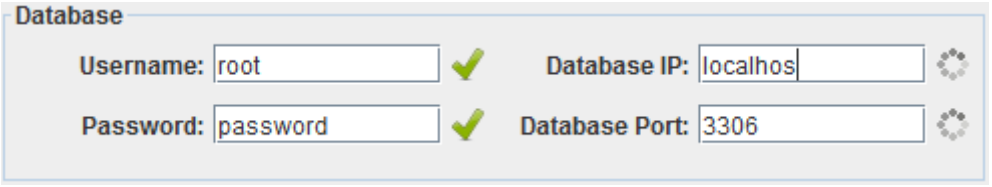

Abbildung 5.9.: Überprüfung der Datenbankverbindung nach Änderung des Inhalts der IP-Adresse

Anwender Statusmeldungen innerhalb von Prozessen, die das Programm zu erledigen hat. In den Einstellungen wird nach jeder Änderung in einem Textfeld der Status der Verbindung abgefragt. Bei den Parametern Port und IP-Adresse können diese Statusabfragungen bis zu drei Sekunden dauern. Solange kann der Bestätigungsbutton nicht gedrückt werden und hinter den Textfeldern wird ein Bearbeitensymbol eingeblendet, sowie in Abbildung 5.9 verdeutlicht. Während der Transformation von Daten muss der Anwender warten, bis diese vollständig abgearbeitet ist. Er kann nicht gleichzeitig zwei Transformationen starten. Damit auch hier keine Fehler entstehen können, bekommt der Benutzer anhand eines Fortschrittsbalkens mitgeteilt, wie weit der Prozess durchgeführt wurde. Die Buttons, um eine neue Transformation zu starten, sind, solange die Übertragung von Daten läuft, deaktiviert wie in Abbildung 5.10 zu sehen ist. Dann kann der Benutzer selbst keine Fehler machen. Alle anderen Funktionalitäten können dennoch ausgeführt werden. Die

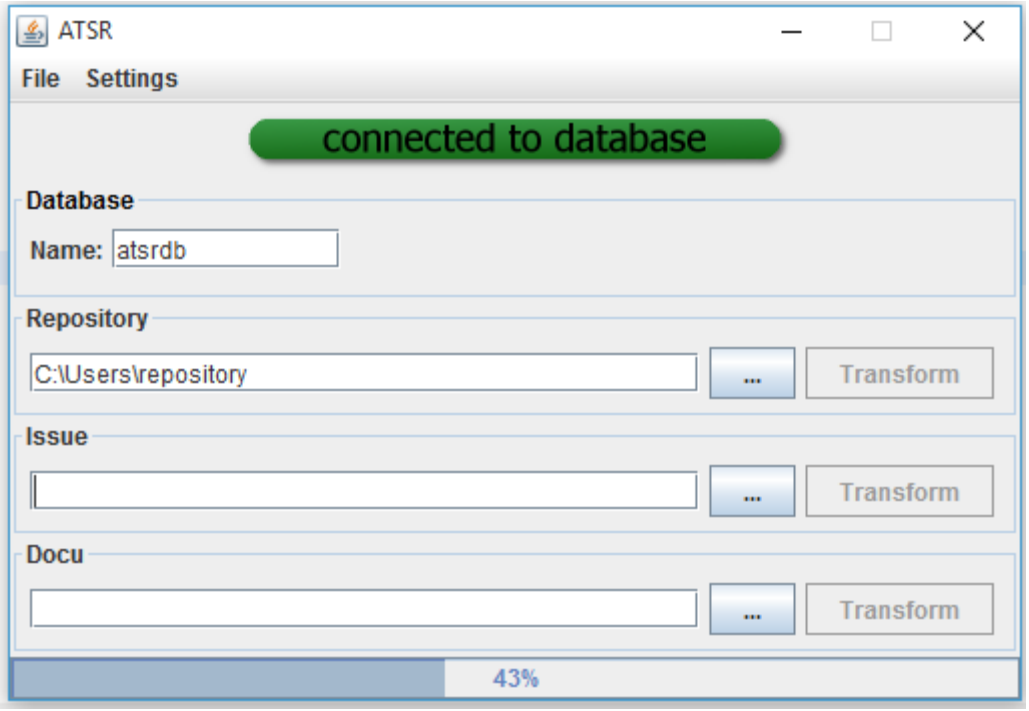

Abbildung 5.10.: "ATSR" während einer Transformation in der Hauptansicht

Zufriedenheit für den Anwender zu optimieren ist die undefinierbarste Aufgabe, denn dieser Punkt kann nicht so einfach aus einem objektiven Standpunkt heraus gewertet werden, da das „ATSR“ ein sehr kleines Programm ist, es nur zwei Ansichten gibt und in diesen beiden Fällen die Informationen kompakt gehalten sind. Da die Hilfestellung von der Software ausreichend ist, ist dieser Punkt auch zufriedenstellend.

5.2 Verwaltung der Einstellungen

Um die Anwendung des Programms zu erleichtern, gibt es für die Einstellungen eine automatisierte Speicherungs- und Ladefunktion. Das heißt, werden die Textfelder in den Einstellungen gesetzt und vom Benutzer bestätigt, speichert die Anwendung die Werte in einer extern liegenden Datei. Sobald diese Datei existiert, wird sie beim Starten geladen und die Parameter in die Textfelder des „ATSR“ geschrieben. Somit kann der Benutzer auch nach einer längeren Pause sofort eine vorherig durchgeführte Transformation ausführen. Das einzige, was nicht gespeichert wird, sind die Pfade, die im Hauptbildschirm für die einzelnen Transformationen eingetragen werden müssen. Das muss immer wieder neu eingetragen werden. Es gibt genau sechs Parameter, die mit diesem Verfahren gespeichert werden. Die Datenbankverbindungsdaten, Benutzername, Passwort, IP-Adresse und Port, der „Git“-Pfad, welcher den Dateipfad der Ausführungsdatei von der Software „Git“ beinhaltet und zuletzt den Namen der Datenbank, was als einziger Parameter außerhalb der Einstellungsansicht einzutragen ist.

Listing 5.1.: Importiere die Datei, welche die Einstellungsparameter enthält, in die Klasse „Settings“.

```
Settings settings = new Settings();
FileInputStream fis = new FileInputStream(settingsFile);
ObjectInputStream in = new ObjectInputStream(fis);
settings = (Settings) in.readObject();
in.close();
fis.close();
```

Das gesamte Verfahren verläuft mithilfe des Java Interfaces „Serializable“. Dafür wird eine Klasse erstellt, die dieses Interface implementiert und Werte enthält. Bei uns ist dies die Klasse „Settings“. Da es nur Parameter gibt, die mit einfachen Datentypen definiert werden können, bietet sich das Interface an. Wenn eine Klasse serialisiert, wird heißt das in diesem Kontext, dass diese Klasse sich in einen Byte-Datenstrom transformieren lässt und auch umgekehrt aus einem Datenstrom von Bytes in eine Klasse zurück wandeln lässt. Damit lassen sich mit wenig Aufwand die gewünschten Informationen auslagern und einlesen. In dem Listing 5.1 ist der Code-Ausschnitt für das Importieren einer Datei in das Programm.

Es muss ein Einlesedatenstrom zu der Datei geöffnet werden, dann kann mit einem Befehl die komplette Datei eingelesen und den richtigen Attributen der Klasse zugeordnet werden. Für das Exportieren der Datei ist Listing 5.2 ein Ausschnitt des Codes. Auch hier muss ein Datenstrom geöffnet werden und über den „write“-Befehl wird die gesamte Klasse als Bytes in die Datei übertragen. Die Datei, welche beschrieben wird, trägt die Endung „SER“. Dies ist ein Indikator für eine serialisierte Klasse.

Listing 5.2.: Exportieren der Klasse „Settings in eine externe Datei.

```
FileOutputStream fos = new FileOutputStream(settingsFile, false);
ObjectOutputStream out = new ObjectOutputStream(fos);
out.writeObject(settings);
out.close();
fos.close();
```

5.3 Transformation

Bei dem Kapitel über das Transformationskonzept wurde bereits erklärt, dass eine Transformation im Allgemeinen in drei Schritte zu unterteilen ist. Abbildung 5.11 zeigt, wie das Konzept bei dem „ATSR“ umgesetzt wurde. Das Einlesen erfolgt anhand eines Datenflusses, in der Abbildung als Kanal dargestellt, was die geforderten Daten von einer Datenquelle außerhalb des Programms bezieht. Die Verarbeitung verläuft synchron zum Einleseverfahren. Der „ATSR“ manipuliert die Daten insofern, dass diese vorbereitet und in das richtige Format gepackt werden, damit das Ausgeben an einem Stück durchgeführt werden kann. Am Ende werden die gesammelten und verarbeiteten Daten in ein Datensystem außerhalb der Software gepackt. Dieses System ist eine MySQL-Datenbank. Der Aufbau sowie das Konzept davon wurde in 4.2 Datenbankkonzept beschrieben.

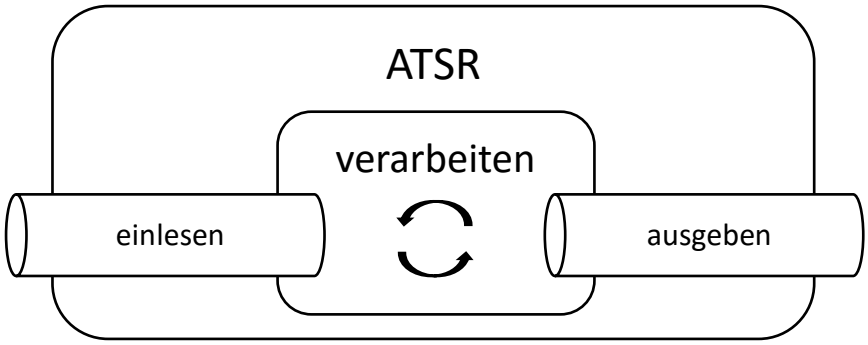

Abbildung 5.11.: Transformationsüberblick des "ATSR“

Was zusätzlich bei allen Transformationen gleich ist, ist die Fortschrittsanzeige in der Hauptansicht. Damit diese immer aktuell bleibt, wurde ein Entwurfsmuster verwendet. Das Muster heißt „Observer“. Die Transformer werden dadurch zu Objekten, die beobachtet werden. Somit kann der Controller den Fortschritt erfassen und die Information an den Fortschrittsbalken weiterleiten. Durch dieses Entwurfsmuster müssen keine manuell erstellten „Listener“, die rückläufig Informationen zum Controller schicken, implementiert werden.

5.3.1 Commit-Transformation

Um die „Commits“ in das Programm einzulesen, muss der entsprechende „Git“-Befehl, welcher die Log-Daten abruft, im richtigen Pfad in der Kommandozeile ausgeführt werden.

Die Kommandozeile bietet sich für Anfragen an „Git“ an, denn auf sie kann von dem „ATSR“ aus über Datenströme darauf zugegriffen werden. Das heißt, es können sehr leicht Eingaben gesendet und Ausgaben ausgelesen werden. Da aber die Kommandozeile ein externes Programm ist, welches „Git“ aufrufen soll, muss das Programm aus einem neuen Prozess angestoßen werden. Genau dafür sind die Zeilen aus Listing 5.3 zuständig. Des Weiteren werden dort direkt die Datenströme abgegriffen und für die weitere Verarbeitung instanziiert. Somit lassen sich über den „PrintWriter“ Kommandos auf die Konsole ausgeben und die empfangenen Zeilen können direkt über den „InputStream“ eingelesen und verarbeitet werden.

Listing 5.3.: Eröffnen der Verbindung zu der Kommandozeile

```
String[] command = { "cmd" };
Process process = Runtime.getRuntime().exec(command);
PrintWriter cmdOut = new PrintWriter(process.getOutputStream());
InputStream cmdIn = process.getInputStream();
```

Nachdem dies ermöglicht ist, muss der Pfad in der Kommandozeile geändert werden. Dafür muss der Pfad des Ordners, welcher in der Hauptansicht für die Transformation eingefügt wurde, mit dem Befehl „change directory“ ausgeführt werden. Danach ist der Ausgangspunkt der Konsole der gewünschte Ordner.

Nun ist der „Git“-Befehl auszuführen. Da „Git“ selbst ein eigenständiges Programm ist, muss bevor der eigentliche Befehl erfolgt „Git“ selbst geöffnet werden. Da aber die

```
[git] log --pretty=format:"%h#%an#%ad#%s" --name-only
```

Abbildung 5.12.: „Git“-Befehl um alle „Commits“ zu erfassen

Parameter mit dem Öffnen kombiniert werden müssen, hat der Befehl zwei Teile. Zum einen der Dateipfad von „Git“, in diesem ist die EXE-Datei anzugeben und zum anderen „Git“ spezifische Kommandos. In Abbildung 5.12 ist dieser Teil das eckig umklammerte „Git“. In den Einstellungen ist es das Textfeld, in dem der Benutzer diesen Parameter einzugeben hat. Der nächste Parameter bestimmt die Art des Befehls. In unserem Fall ist es ein Log-Befehl. Dieser gibt für den aktuellen Systempfad die Log-Daten aus. Wie die Formatierung und die

Struktur bei der Ausgabe aussehen soll, kann mit zusätzlichen Parametern definiert werden. Mit der „pretty“-Option lassen sich bestimmte Formatierungen generieren. Da wir den Parameter „format“, mit einem Nachfolgenden „String“ hinzugefügt haben, können wir eine komplett individuelle Darstellung der Daten generieren lassen. Die Parameter, welche wir erhalten wollen, müssen mit einem Prozentzeichen voran in dem „String“ enthalten sein. Wir rufen vier Werte ab, die mit einem Hash-Symbol getrennt sind. Der erste Platzhalter steht für den abgekürzten „Commit“ Hash-Code. Dieser ist eindeutig für jeden „Commit“ und kann zur Identifikation benutzt werden. Der nächste Wert gibt den Namen des Autoren, der die Änderung vollzogen und somit den „Commit“ generiert hat, wieder. Der Platzhalter an der dritten Stelle steht für das Erstellungsdatum des „Commits“ und zuletzt wird die Nachricht, die zu dem „Commit“ gemacht wurde abgefragt. Damit ist die Angabe zur Formatierung der „Commit“-Daten abgeschlossen. Als letztes wird noch die Struktur der Ausgabe der Dateien zu dem „Commit“ definiert. Mit dem Parameter „name-only“ werden die Daten mit ihren Pfaden untereinander angegeben. Damit sind alle Daten, die für das spätere Data Mining wichtig sind, erfasst [1].

All die Informationen, die im vorherigen Abschnitt aufgelistet wurden, werden beim Ausführen des Befehls in die Konsole ausgegeben und dadurch, dass wir diese Ausgabe als Datenstrom erfasst haben, in den „ATSR“ eingelesen. Die erhaltenen Daten werden von dem Programm in die einzelnen Komponenten aufgespalten und in „Commit“-Objekte transferiert. Diese Objekte werden zum Schluss als Liste dem „DatabaseWriter“ übergeben. Der wiederum speichert alle Daten in die Datenbank. Der Ablauf hierfür wird in dem Abschnitt 5.4 Datenübertragung in die MySQL-Datenbank beschrieben.

5.3.2 Issue- und Docu-Transformaton

Die Transformationen von „Issues“ und „Docus“ können zusammenfassend erläutert werden, denn bei beiden ist der Ablauf identisch, weil die Datenquelle, aus der alle Daten bezogen werden, dasselbe Format hat. Die CSV-Dateien, die für beide Transformationen die Daten enthalten, sind für jede Spalte mit einem Semikolon getrennt. Der einzige Unterschied für die einzelnen Transformationen sind die Spalten, welche ausgelesen werden.

Bei den „Issues“ sind das die Identifikationsnummer, wobei in der Datei diese Spalte eine fortlaufende Nummerierung ist, der Status, der Typ, auch hier gibt es eine Besonderheit in der Datenquelle, denn diese Spalte wird dort „Tracker“ genannt, und die Beschreibung des „Issue“. Bei den „Docus“ gibt es bisher keine definierte CSV-Datei, aber die drei Spalten, die mit Sicherheit darin vorkommen und auch von dem Programm eingelesen werden, sind die Identifikationsnummer, der Pfad für die entsprechende Dokumentation und eine Beschreibung dazu.

Genauso wie bei der Transformation von „Commits“, wird ein Datenstrom zur Datenquelle geöffnet, hier eine einfache Datei, und diese eingelesen. Während dem Einlesen werden die einzelnen Zeilen verarbeitet und in Objekte des Transformationsformats gepackt. Nachdem das Verarbeitet aller eingehenden Daten beendet ist, können die Objekte als Liste dem „DatabaseWriter“ übergeben werden und dieser speichert die Daten in der Datenbank wie in Abschnitt 5.4 beschrieben wird.

5.4 Datenübertragung in die MySQL-Datenbank

Das Konzept der Datenbankbeschreibung wurde in 4.3.3 Datenspeicherung beschrieben. In Listing 5.4 ist der Programmcode für das Speichern von „Commits“ anhand des Konzepts zu sehen. Der „DatabaseWriter“ bekommt eine Liste der fertigen „Commits“ und schreibt diese mittels JDBC in die Datenbank.

Listing 5.4.: Konzept der Speicherung, in die Datenbank, implementiert bei der Transformation von „Commits“

```
public void writeCommits(List<Commit> commits) {
	if (commitTablesExist()) {
		if (!commitTablesAreEmpty()) {
			// Create backup of the tables
			createCommitBackup();
			// Clear tables
			clearCommitTables();
		}
	} else {
		// Create tables
		createCommitTables();
	}
	// Describe tables
	writeInCommitTables(commits);
}
```

Dafür muss eine Verbindung mithilfe der Parameter aus den Einstellungen aufgebaut werden. Wenn dies erfolgreich abgeschlossen wurde und die Speicherung starten soll, wird der Ablauf gestartet. Zuerst wird überprüft, ob die Tabellen bereits existieren. Bei einer „Issue“- oder „Docu“-Transformation gibt es nur eine Tabelle in der Datenbank, da genügt die Überprüfung dieser einen Tabelle. Bei einer „Commit“-Transformation sind drei Tabellen vorhanden. Nur wenn alle drei bereits existieren, wird die Rückgabe der Funktion wahr. Danach wird überprüft ob die Tabellen bereits beschrieben wurden. Genau wie im ersten Schritt müssen bei den „Commits“ drei Tabellen geprüft werden und nur wenn alle drei leer sind, wird auch zurückgegeben, dass die Tabellen leer sind. Bei „Issues“ und

„Docus“ muss nur eine Tabelle überprüft werden. Sind sie noch befüllt, müssen Backups erstellt werden. Diese bekommen den aktuellen Zeitstempel in den Namen und werden als separate Tabelle gespeichert. Das geschieht mit jeder einzelnen Tabelle. Erst nach diesem Punkt können die Tabellen geleert werden. Falls die Tabellen im allerersten Schritt noch nicht existiert haben, müssen diese natürlich noch erstellt werden, bevor sie beschrieben werden. Die letzte Aufgabe ist das eigentliche Beschreiben der Datenbanktabellen.

Um das Verfahren zu beschleunigen, auch wenn große Mengen an Daten übertragen werden, werden vorbereitete „SQL-Statement“ erstellt und diese dann pro „Commit“, „Issue“ oder „Docu“ gefüllt. Dabei müssen nur noch die Werte der Daten in den Befehl eingebunden und ausgeführt werden.

Listing 5.5.: Beispiel eines vorbereiten SQL-Befehl und befüllen mit Werten anhand eines „Commits“

```
String commitQuery = "insert into committable (id, author, date, message)
values (?,?,?,?)";
PreparedStatement commitStmt = conn.prepareStatement(commitQuery);
commitStmt.setString(1, commit.getId());
commitStmt.setString(2, commit.getAuthor());
commitStmt.setTimestamp(3, new Timestamp(commit.getDate().getTime()));
commitStmt.setString(4, commit.getMessage());
```

6 Validierung

Eine Transformation wird immer über das komplette Repository ausgeführt. Das kann dazu führen, dass die Übertragungszeit sehr lange dauert. Um darüber mehr Informationen zu erhalten, wurden Tests zur Erfassung der Laufzeit gemacht, die in diesem Kapitel präsentiert werden.

6.1 Performanz

Für den Test wurde von der Seite „github.com“ mehrere „Open Source“-Software-Repositories heruntergeladen und auf dem lokalen System gespeichert. Von dort konnten diese verschiedenen Projekte in Bezug auf „Commits“, Größe des Projekts und Anzahl an beteiligten Entwicklern verglichen werden. Dadurch können Aussagen anhand der Laufzeit und dieser drei Parameter getroffen werden.

Das Diagramm in Abbildung 6.1 zeigt für jedes getestete Repository hundert „Commits“ pro Einheit, die Größe des Repsoitories in Megabytes, die Anzahl der Entwickler und die Dauer einer Transformation, wobei nur Transformationen durchgeführt wurdenn bei denen das „ATSR“ eine neue Datenbank erstellen musste. Damit sind die Laufzeiten besser miteinander zu vergleichen, da bei allen der Ablauf der Datenbankbeschreibung gleich war. Wenn man die Werte einzeln mit der Transformationszeit vergleicht, kann man erkennen, dass die Größe des Projektes, also wie viele Bytes benötigt werden, keinen Einfluss darauf hat, wie schnell eine Transformation abgearbeitet wird. Betrachtet man hier im Genaueren „toaruos“ mit „nim“, kann man erkennen, dass die Größe des Repositories sich um 13,3 Megabytes unterscheidet. Dabei ist die Dauer der Übertragung von „nim“ mehr als viermal so hoch wie bei „tuaruos“. Ein ähnliches Verhalten kann zwischen „hextris“ und „textmate“ erkannt werden. Die beiden anderen Parameter haben ein ähnlicheres Verhalten zur Transformationszeit. Dennoch sind die Entwickler nicht ganz so auschlaggebend wie die Anzahl der „Commits“. Wenn man dafür „neovim“ und „nim“ vergleicht, bei denen beide Projekte mehr als 150 verschiedene Entwickler haben, unterscheidet sich der Wert der Dauer

stark. Die Anzahl der „Commits“ spiegelt die Dauer einer Transformation am besten wieder, wobei auch hier definiert werden kann, dass ein „Commit“ diese Transformationszeit benötigt.

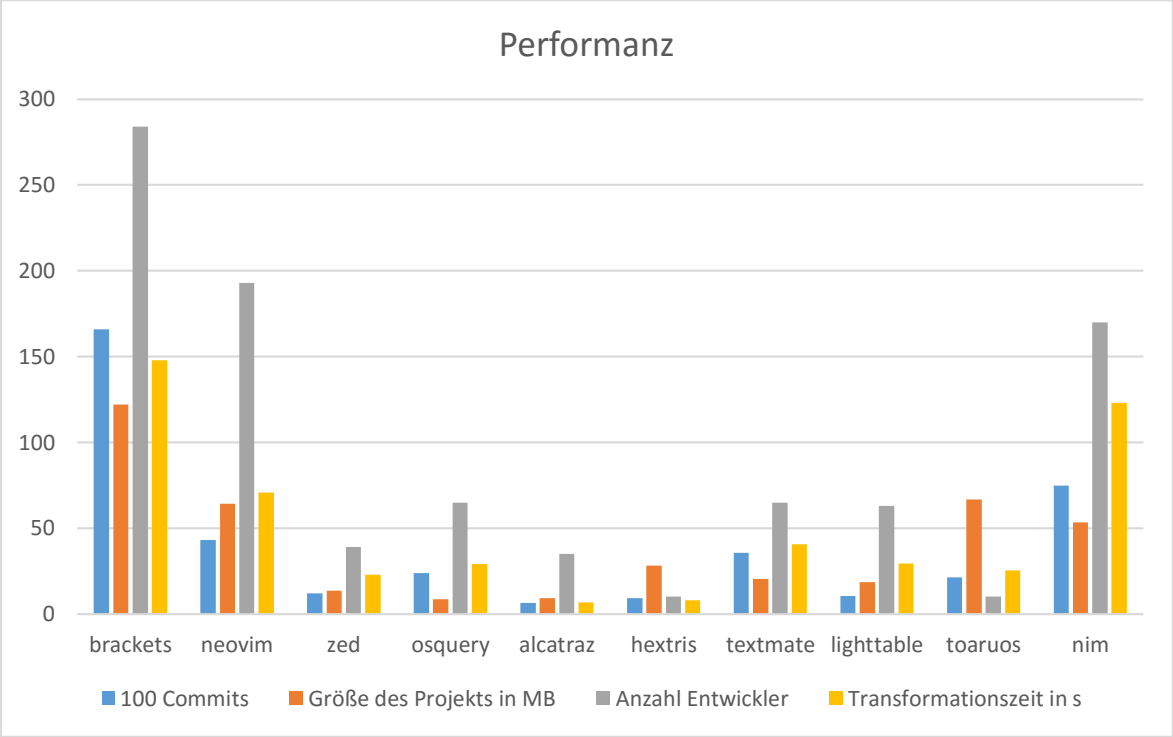

Abbildung 6.1.: Test der Transformationszeit anhand von „Commits“, Größe des Projekts und Anzahl Entwicklern. Alle drei Werte sind von „github.com“

Aus diesen Daten kann zusammenfassend gesagt werden, dass die zwei auschlaggebenden Parameter, die Anzahl der „Commits“, sowie die Entwickleranzahl die Übertragungszeit am meisten beeinflussen. Die Entwickler für sich selbst nicht direkt, aber ein Projekt, das viele Entwickler hat, hat meistens auch viele „Commits“. Und dazu kommen nicht nur kleine Änderungen am Projekt, sondern auch große, die viele Dateiänderungen pro „Commit“ beinhalten. Das führt dann zu längeren Laufzeiten bei der Transformation. Die Größe des Projekts in Bytes spielt dagegen keine Rolle.

7 Zusammenfassung

In diesem Kapitel sind die Ergebnisse zusammengefasst und die möglichen Verbesserungen oder Erweiterungen beschrieben.

7.1 Fazit

Die Aufgabe dieser Arbeit war es, ein Programm zu entwerfen und zu implementieren, welches Daten von Software-Repositories in eine Datenbank überträgt. Dabei soll die gesamte Transformation mithilfe von Datenströmen funktionieren. Damit ist gemeint, dass keine Dateien für die Zwischenspeicherung erstellt werden sollen. Genau dafür ist das Resultat dieser Arbeit ausgelegt und entwickelt worden. Es kann nun Daten aus „Git“ extrahieren, für Data Mining vorbereiten und diese in eine MySQL-Datenbank abspeichern. Dabei werden keine temporären Dateien für die Übertragung generiert. Deshalb kann festgehalten werden, dass die Aufgabe erfüllt wurde.

Dazu wurde die Benutzbarkeit des Programms mehrfach erweitert, damit der Benutzer sich leicht in das „ATSR“ einarbeiten kann. Dies wurde in der Analysephase nicht mit dem Stellenwert, den es haben sollte, betrachtet. Dadurch kam Arbeit hinzu, die den Zeitplan am Ende der Thesis, verzögert hat. Aber es waren sehr gute und wichtige Verbesserungen, wie sich im Nachhinein feststellen lässt, denn damit sind die meisten Unklarheiten beim Benutzen aus dem Weg geräumt worden und es können wesentlich weniger Fehler auftreten.

Was noch unklar ist, ist die Effizienz des Programms in dieser Hinsicht zur Erweiterbarkeit mit weiteren Formaten. Bisher gibt es drei Formate, von denen zwei eine identische Datenquelle haben. Bei diesem Punkt kann nicht sicher festgehalten werden, ob es leicht wird, weitere Formatmodelle hinzuzufügen, denn es gab keine Testmöglichkeiten im Bezug darauf. Genauso ist das Datenformat „Docu“ noch nicht komplett definiert. Das heißt, es gibt keinen spezifizierten Aufbau dieser CSV-Datei und das konnte nur teilweise getestet und geprüft werden. Dabei können natürlich noch Probleme entstehen, da die Schnittstellen immer ein hohes Fehlerpotential enthalten.

7.2 Weitere Schritte

Programme, die ganz neu entwickelt wurden, haben meistens großes Verbesserungspotential. Nicht weil sie noch viele Fehler enthalten, sondern da erst bei der häufigen Benutzung Ideen entstehen, für welche weiteren Aufgaben das Programm nützlich wäre. Für das „ATSR“ gibt es durch Analysen und Tests bisher schon ein paar Ideen für Erweiterungen, die aber auf Grund von der begrenzten Zeit nicht mehr umgesetzt werden konnten.

Der Ablauf der Datenbankbeschreibung könnte am meisten verändert werden. Bisher wird ein komplettes Software-Repository übertragen, wenn die Transformation gestartet wird. Dabei kann es vorkommen, dass dasselbe große Repository immer wieder übertragen werden soll und dieser Vorgang dann viel Zeit kostet. Wenn hingegen vor der Transformation die Datenbank überprüft wird, ob Daten von dem Projekt in der Datenbank sind und nur noch die neuen „Commits“ übertragen werden, könnte man hier viel Zeit für den Anwender sparen. Dann sind aber natürlich mehrere Punkte zu beachten. Wenn nun die Übertragung unterbrochen wird, das kann durch den Verlust der Datenbankverbindung oder abstürzten des eigenen Systems entstehen, dann ist nicht klar, wie konsistent die Daten noch sind. Hier müssten mehrere Sicherheitsmechanismen eingebaut werden, damit keine Daten verloren gehen. Das kann bisher nicht passieren, denn wenn ein Fehler bei der Übertragung passiert, muss die Transformation von neuem gestartet werden. Als weitere Erweiterungsmöglichkeit der Datenbankverbindung könnte das komplette Repository vor der Verarbeitung geklont und lokal abgespeichert werden. Damit kann verhindert werden, dass der „Git“-Befehl bei großen Repositories aus anderen Netzwerken zu viel Zeit in Anspruch nimmt. Und bei einem Verbindungsabbruch würde das Programm selbst solange keine Probleme bekommen, bis das komplette Projekt geklont wurde.

Alle Erweiterungen sind bisher dafür da, um das Programm effizienter zu machen. Der letzte Punkt ist eine Idee, die abhängig vom Einsatz des „ATSR“ ist. Es wäre auch eine Möglichkeit das Programm als Service einzurichten, ohne eine Benutzerfläche. Dann könnte man täglich Transformationen starten und der Benutzer müsste sich nur noch mit den

Tabellen in der Datenbank beschäftigen. Diese Umstrukturierung wäre aber nur ein Fortschritt, wenn dies der Standardfall der Benutzung wäre.

Literaturverzeichnis

[1] S. Chacon und B. Straub, Pro Git, Apress, 2009.

[2] Riccardi, „Data Modeling with Entity-Relationship Diagrams,“ p. 61, 2004.

[3] P. Chen, „The entity-relationship model - toward a unified view of data,“ pp. 9-36, 1976.

[4] L. Welling und L. Thomson, PHP and MySQL Web development, Sams Publishing, 2003.

[5] Oracle, „www.oracle.com,“ Oracle Corporation, [Online]. Available: http://www.oracle.com/technetwork/java/javase/jdbc/index.html. [Zugriff am 29 10 2015].

[6] J. Nielsen, Usability engineering, Elsevier, 1994.

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