Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 136

Synchronisierung der Dokumentation von Software-Modulen

Michael Happel

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Stefan Wagner
Betreuer:	Dipl.-Inf. Ivan Bogicevic
Beginn am:	8. Mai 2014
Beendet am:	7. November 2014
CR-Nummer:	D.2.7, H.2.7

Kurzfassung

Diese Bachelorarbeit beschäftigt sich mit der Frage, wie Entwickler bei der Aktualisierung ihrer Dokumentation von Computern unterstützt werden können. Dabei wird die Dokumentationssoftware UniMoDoc um eine Synchronisierungsfunktion erweitert, mit der Dokumentationsdaten automatisch mit Informationen, die im Versionsverwaltungssystem des dokumentierten Projekts zu finden sind, aktualisiert werden. Die Recherche zur Aufgabenstellung der Arbeit sowie die Implementierung der Synchronisierungsfunktion sind in dieser Arbeit dokumentiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

> Incorrect documentation is often worse than no documentation.
>
> *(Bertrand Meyer - Creator of the Eiffel programming language)*

1.1 Motivation

In der Softwaredokumentation wird zwischen integrierter und separater Dokumentation unterschieden. Integrierte Dokumentation ist die Dokumentation auf Codeebene, die vor allem durch Kommentare entsteht. Separate Dokumentation ist beschreibt den Dokumentationsstil, wo sich die Dokumentation außerhalb des Programmcodes befindet. [Lud10] Darunter fallen die Dokumentation des Gesamtsystems und die Dokumentation von einzelnen Softwaremodulen.

Bei der Dokumentation des Gesamtsystem wird das komplette Programm anhand seiner Funktionalität beschrieben. Auf die feineren Bestandteile des Programms wird bei dieser Dokumentationsart nicht detailliert eingegangen. Dokumentation auf Modulebene betrachtet die einzelnen Bestandteile, aus denen ein Softwaresystem aufgebaut ist. Darunter fallen zum Beispiel Codepakete, in denen das Benutzerinterface realisiert wird oder auch Strukturen in denen das Datenmodell einer Software untergebracht ist. Die folgende Bachelorarbeit beschäftigt sich mit dem Moduldokumentationswerkzeug UniMoDoc.

Eine Herausforderung der separaten Dokumentation ist es die Dokumentation aktuell zu halten. Wenn sich Teile des Programms ändern, oder neue Features in eine Software eingebaut werden, ist es grundsätzlich einfacher die integrierte Dokumentation aktuell zu halten, da Codekommentare einfach während der Implementierung angepasst werden können.

Da sich aber die separate Dokumentation in unterschiedlichen Dokumenten als der Code befindet, existiert eine Diskrepanz zwischen diesen beiden Dokumenten. Falls diese Abweichungen nicht behoben werden, wird die Dokumentation schnell wertlos und muss mit viel Mühe wieder auf den aktuellen Stand gebracht werden. Dies hat zur Folge, dass die Kosten für diese ohnehin sehr aufwendige und teure Aktivität noch weiter ansteigen.

Am Beispiel von UniMoDoc werden in der Paketdokumentation Eigenschaften des Codes dokumentiert, die für einen Menschen oft schwer und mühevoll zu erheben sind. Ein Beispiel dafür sind die Anzahl an Codezeilen in einem Paket. In diesen Fällen ist es wirksam die Metriken von einem Computer erheben zu lassen um so den Entwickler in seiner Pflicht, ständig eine aktuelle Dokumentation bereitzustellen, zu unterstützen.

1.2 Aufgabenstellung

Die Bachelorarbeit ist in zwei Teile aufgeteilt. Dabei umfasst der erste Teil der Arbeit eine Analyse inwiefern nützliche Informationen für die Dokumentation, die im Versionsverwaltungssystem eines Projektes gespeichert sind, automatisch erhoben werden können. Dabei soll zum einen erhoben werden, welche dieser Informationen für die Dokumentation in Frage kommen, weiterhin soll eine Möglichkeit gefunden werden, an diese Metriken zu gelangen. Dies kann Beispielsweise mithilfe eines Softwarewerkzeugs realisiert werden. Im zweiten Teil soll UniMoDoc um eine Synchronisierungsfunktion erweitert werden. Diese soll automatisch die ausgewählten Metriken aus dem ersten Teil aus dem Versionsverwaltungssystem in die Dokumentation synchronisieren und die gewonnenen Informationen dort anzeigen. Auftretende Konflikte zwischen der bestehenden Dokumentation und den Informationen im Repository sollen dabei gekennzeichnet werden.

1.3 Überblick über die Arbeit

In der folgenden Arbeit ist die Recherche aus Teil eins sowie die Implementierung aus dem zweiten Teil der Aufgabenstellung dokumentiert. Sie ist dabei in sechs weitere Kapitel unterteilt.

Kapitel 2 – UniMoDoc ist ein Überblick über die Dokumentationssoftware, die in dieser Arbeit erweitert wird.

Kapitel 3 – Das Versionsverwaltungssystem Git beschreibt auf welche Art und Weise die Informationen eines Projekts, in Git gespeichert werden.

Kapitel 4 – Die Extraktionswerkzeuge: Hier werden Werkzeuge vorgestellt mit denen Informationen aus einem Repository gewonnen werden können. Dabei wird ein Werkzeug für die Verwendung mit UniMoDoc ausgewählt. Weiterhin wird erklärt welche Informationen, das Werkzeug aus einem Projektarchiv gewinnen kann und wie diese in UniMoDoc für die Dokumetation verwendet werden können.

Kapitel 5 – Synchronisation erklärt den Ablauf der Synchronisation von den gewonnenen Informationen mit der Dokumentation in UniMoDoc mithilfe von Use-Cases.

Kapitel 6 – Die Implementierung ist die Dokumentation der Implementierungsarbeit in UniMoDoc. Hier wird die Umsetzung der, in der Aufgabenstellung geforderten, Programmfeatures vorgestellt. Weiterhin wird auf einige Schwierigkeiten, die während der Implementierung aufgetreten sind, eingegangen und anschließend aufgezeigt, wie diese gelöst werden konnten.

Kapitel 7 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte für zukünftige Erweiterungen der Synchronisierungsfunktion in UniMoDoc vor.

2 UniMoDoc

2.1 Einleitung

UniMoDoc ist ein Werkzeug zur Erstellung einer Dokumentationen auf Modulebene. Ausgeschrieben bedeutet der Programmname *Universal documentation tool for software modules*. Die ursprüngliche Version von UniMoDoc wurde im Rahmen einer Diplomarbeit als Java Swing Anwendung geschrieben, wurde aber von Ivan Bogicevic und Jan Strauß als Webanwendung neu entwickelt. Die Anwendung basiert heute auf dem Java Server Faces(JSF) Framework. UniMoDoc ist unter der Apache Licence Version 2.0 unter folgender Adresse erhältlich: `http://sourceforge.net/projects/unimodoc/`

2.2 Überblick über die Funktionsweise

Abb. 2.1 zeigt die Hauptseite von UniMoDoc, die in vier Bereiche Aufgeteilt ist. Diese sind wie folgt aufgebaut:

1. Modulübersicht: In diesem Bereich werden alle Softwaremodule, die von UniMoDoc dokumentiert werden, aufgelistet. Sie sind hierarchisch nach ihrem qualifizierenden Namen geordnet. Das erste Paket, mit einem Punkt als qualifizierenden Namen, gibt das Hauptverzeichnis des Projekts an. Mit einem Klick auf das Plus Icon im unteren Teil des Bereichs können neue Module hinzugefügt werden. Mit dem Zweig Icon ganz unten, lässt sich die Struktur der Beschreibungselemente aus dem nächsten Bereich ändern.

2. Beschreibungselemente: Hier werden die Dokumentationsdaten für ein, in der Modulübersicht ausgewähltes, Paket aufgelistet. Diese Daten werden in UniMoDoc Beschreibungselemente genannt. Ein Beschreibungselement dokumentiert einen bestimmten Aspekt eines Paketes. Die Beschreibungselemente sind auf sechs Tabs aufgeteilt und werden im nächsten Abschnitt genauer erklärt. Der Unterbereich in dem sich die Tabs befinden wird Tab Navigation genannt. Darin befinden zusätzlich noch ein Icon zum Bearbeiten sowie zum Löschen der Module. Mithilfe des Bearbeiten Icons, kann in einen Editiermodus gewechselt werden, worin der Inhalt für alle Beschreibungselemente manuell geändert werden kann. Mit einem Klick auf den Löschen Icon kann ein Modul gelöscht werden.

3. Seitenleiste: In der Seitenleiste werden dem Benutzer nützliche Hilfsinformationen angezeigt. Beispielsweise wird der in Abb. 2.1 in diesem Bereich erscheinende Hilfstext dann gezeigt, wenn der Benutzer mit dem Mauszeiger über das Beschreibungselement *Tasks* fährt.

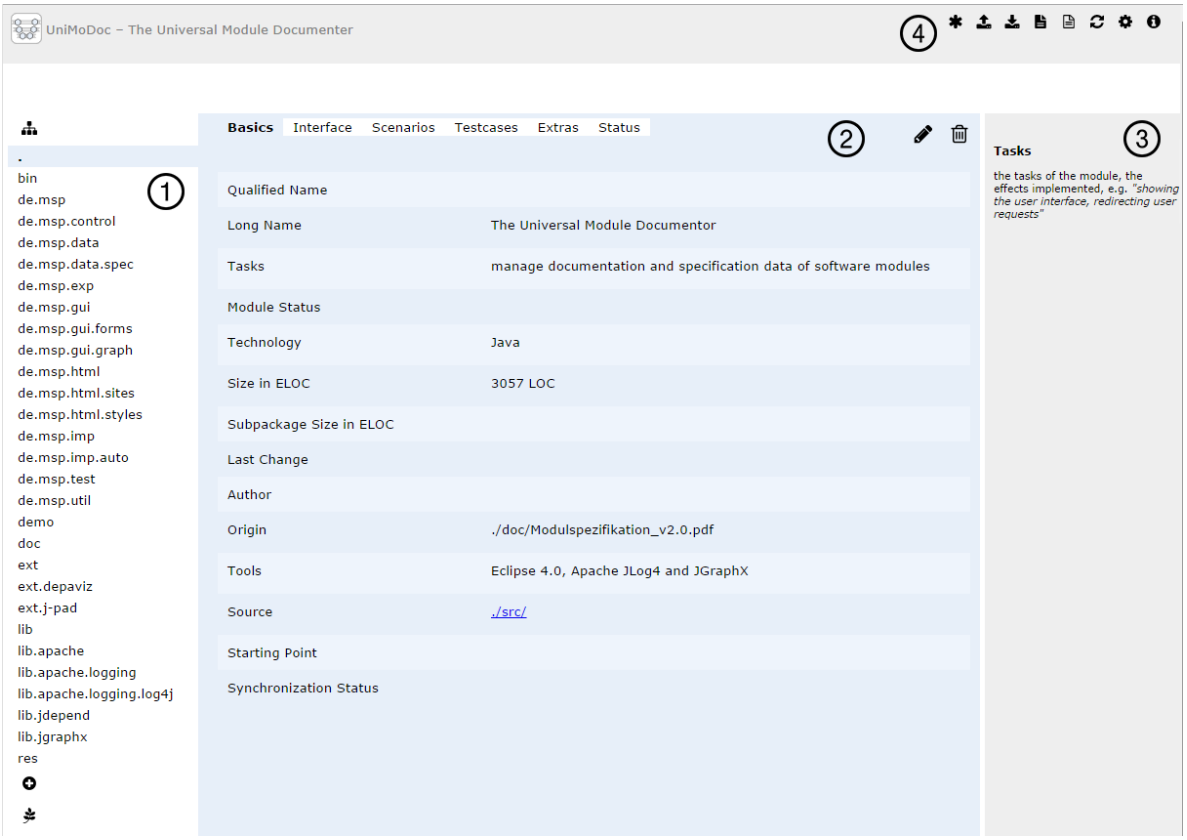


Abbildung 2.1: UniMoDoC Übersicht

4. Titelleiste: Im rechten Teil der Titelleiste befinden sich Icons, die durch anklicken bestimmte Operationen ausführen. Darunter gibt es die Option ein neues Projekt anzulegen. Weiterhin gibt es eine Import- und Exportfunktion für XML Dateien sowie weitere Exportfunktionen für PDF und RTF Dateien. Auch kann hier die, in dieser Arbeit implementierte Synchronisierungsfunktion gestartet werden. Zusätzlich gibt es noch eine Option für einen Einstellungs- sowie einen Aboutdialog.

2.3 Die Beschreibungselemente

2.3.1 Basics Tab

Im Basics Tab finden sich alle grundlegenden Informationen über das Paket. Die einzelnen Beschreibungselemente dieses Tabs werden im folgenden kurz erklärt.

Qualified Name: Der vollständig qualifizierte Name des Pakets, beispielsweise `org.example.gui`

Long Name: Der vollständige, ungekürzte Name des Pakets, beispielsweise *Graphical User Interface* für ein Modul mit dem Kurznamen `gui`.

Tasks: An dieser Stelle werden die Aufgaben des Pakets im Gesamtprogramm dokumentiert. Beispielsweise kann ein Paket für den Dateiexport zuständig sein, oder es implementiert das Benutzerinterface.

Module Status: In diesem Attribut wird der Entwicklungsstatus des Pakets durch die Angabe des derzeitigen Entwicklungsschritts angegeben. Entwicklungsschritte sind zum Beispiel: *Implementierung fertiggestellt* oder *Code im Review*.

Technology: Hier werden die Haupttechnologien und Programmiersprachen aufgelistet, die für das Paket verwendet wurden. Eine Technologie wäre zum Beispiel das JSF Framework, eine Programmiersprache Java.

Size: Dieses Attribut gibt die Größe des Pakets in Lines of Code(LOC) an.

Last Change: Der letzte Zeitpunkt, an dem Inhalte des Pakets geändert wurden.

Origin: Dieses Attribut gibt Referenzen an, die wichtig für das Paket sind. Wurden beispielsweise Bilder oder Schriftarten in diesem Paket verwendet, kann das Attribut die URL der Webseite enthalten, von der diese Ressourcen stammen. Weiterhin kann auf Spezifikation verlinkt werden, wo dieser Teil des Programms dokumentiert wurde.

Author: Hier werden die Personen aufgelistet, die bei der Erstellung von diesem Paket mitgearbeitet haben.

Tools: Dieses Attribut beschreibt die Werkzeuge die bei der Erstellung des Pakets verwendet wurden. Beispiele dafür sind das Entwicklungsframework Eclipse oder ein GUI-Builder.

Source: Dieses Beschreibungselement gibt den Pfad zu dem Ordner, in dem sich dieses Modul befindet, ausgehend vom Hauptverzeichnis aus an.

Starting Point: Hier wird der Einstiegspunkt im Code dokumentiert, wo das Programm als erstes beginnt Code auszuführen. Gibt es keinen direkten Einstiegspunkt kann hier auch der Teil des Pakets mit dem größten Umfang notiert werden.

In diesem Tab sind auch die neuen Beschreibungselemente untergebracht, die im Rahmen dieser Arbeit hinzugefügt werden. Sie werden in Abschnitt 4.5.1 erklärt.

2.3.2 Interface

In diesem Tab werden die, in diesem Modul angebotenen, Funktionen und Abhängigkeiten dokumentiert.

2.3.3 Scenarios

Hier werden typische Anwendungsfälle textuell beschrieben.

2.3.4 Testcases

Dieser Tab ist noch nicht fertig implementiert. Er soll zur Dokumentation von Testsuites, das sind thematisch gruppierte Testblöcke dienen.

2.3.5 Extras

Dieser Tab enthält ein großes Textfeld, wo zusätzliche Informationen über das Paket, die noch nicht dokumentiert wurden, beschrieben werden können.

2.3.6 Status

Dieser Tab bietet eine Übersicht zum Entwicklungsstatus des Pakets.

Document Status: Dieses Beschreibungselement gibt den Status des Dokuments an, welches das Paket dokumentiert.

Document Review Status: Der Status des Reviews von der Dokumentation zu diesem Paket.

Module Status: In diesem Attribut wird der Entwicklungsstatus des Pakets durch die Angabe des derzeitigen Entwicklungsschritts angegeben. Entwicklungsschritte sind beispielsweise: *Implementierung fertiggestellt* oder *Code im Review*.

Module Review Status: Hier wird der aktuelle Status der Reviews zu diesem Paket gespeichert.

3 Das Versionsverwaltungssystem Git

3.1 Einleitung

Git ist ein System zur verteilten Versionsverwaltung von Dateien. Es wurde ursprünglich für die Verwaltung des Quellcodes für den Linux Kernel entwickelt. Verteilte Versionsverwaltungssysteme zeichnen sich dadurch aus, dass jeder Client, mit dem das Repository geteilt wird ein Backup der gesamten Datenmenge besitzt. Somit ist das System nicht an einen zentralen Server gebunden. Im folgenden werden zunächst einige Grundbegriffe kurz umrissen, anschließend wird auf die interne Datenhaltung von Git eingegangen, die es möglich macht, dass für alle Dateien, die in das System eingebunden sind, die komplette Historie gespeichert wird, und bei Bedarf auf jeden vorherigen Stand zurückgesetzt werden kann [Cha09].

3.2 Drei Bereiche und drei Zustände

Dateien in Git befinden sich in einem von drei Zuständen: *Verändert*, *Gestaged* und *Commited*. Veränderte Dateien sind Dateien, die im Arbeitsverzeichnis neu erstellt oder modifiziert wurden und deren veränderter Zustand noch nicht in das Repository commited wurde. Diese Dateien können gestaged

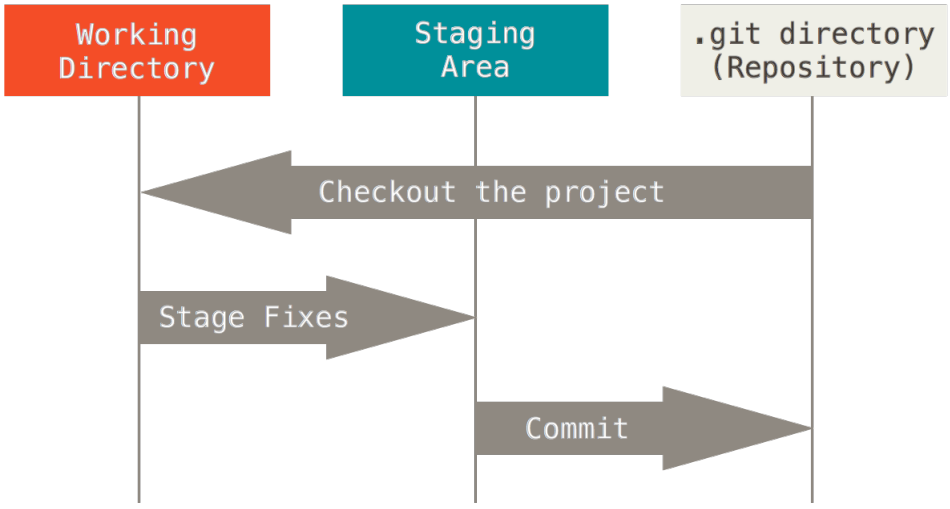


Abbildung 3.1: Die drei Bereiche von Git [Cha09]

werden. Man sagt zu diesen Vorgang auch, dass die Datei dem Index hinzugefügt wird. Sie werden dabei markiert um mit dem nächsten Commit in das Repository übernommen zu werden. Solange Änderungen noch nicht committed wurden, werden sie in der sogenannten *Staging Area* gespeichert. Dateien die committed sind, befinden sich sicher im Repository [Cha09].

3.3 Branchen und Mergen

Ein prominentes Feature von Git ist das Branching (verzweigen). Dies ermöglicht dem Benutzer vom Hauptentwicklungszweig abzuweichen und einen neuen Zweig zu erstellen, wo Änderungen commited werden können, ohne das der Hauptzweig dabei verändert wird. Dies ermöglicht paralleles Arbeiten sowie eine risikofreie Umgebung um größere Änderungen testen zu können. Durch Mergen können diese Zweige wieder zu einem Zweig zusammengeführt werden [Cha09].

3.4 Das Innenleben eines Git-Repositories

Wenn in einem Ordner ein Git Projekt mit dem Befehl `git init` angelegt wird, erstellt Git dort einen neuen Ordner mit dem Namen `.git`, in dem alle internen Daten für das Repository abgespeichert werden. Diese Daten lassen sich allgemein in zwei Gruppen unterteilen: *Git Objekte* und *Git Referenzen*. Die Funktion dieser Gruppen und deren Entitäten wird in den folgenden Abschnitten grob umrissen [Cha09].

3.5 Git Objekte

Die Git Objekte sind für die Erstellung eines internen Datenmodells verantwortlich, welches die Historie aller Dateien in einem Projekt über dessen gesamten Lebenszeitraum abbildet. Dadurch ist es möglich alle Änderungen die im Projekt durchgeführt wurden rückgängig zu machen. Im folgenden werden die drei wichtigsten Git-Objekte vorgestellt. Diese werden Blob-, Tree- und Commitobjekte genannt [Cha09].

3.5.1 Das Blob Objekt

Blob Objekte werden verwendet um die verschiedenen Dateien, die im Projekt vorkommen zu speichern. So entsteht aus jeder einzelnen Text-, Bild- und Codedatei jeweils ein Blob Objekt. Wird im Verlauf des Projekts eine dieser Dateien bearbeitet, wird eine neues Blob Objekt erstellt, in dem die veränderte Datei gespeichert wird. Blob Objekte werden im Repository angelegt, sobald eine Datei zum Index hinzugefügt wird. Dabei wird anhand des Inhalts der Datei ein 40- Stelliger SHA-1 Hash erzeugt, der als zukünftige Referenz für das Objekt dient. Im Ordner `.git/objects` wird anschließend, falls noch nicht vorhanden, ein neuer Ordner erstellt. Der Name des Ordners besteht aus den ersten zwei Stellen des Hashs. In diesem Ordner wird die Datei abgelegt. Dabei wird sie zunächst komprimiert

und erhält anschließend die letzten 38 Stellen des Hashs als Dateinamen. Der gesamte Hash wird anschließend in der Index Datei im `.git` Ordner hinterlegt [Cha09].

3.5.2 Das Tree Objekt

Das Tree Objekt existiert um die vielen Blob Objekte eines Projekts zu organisieren und ihnen eine Struktur zu geben. Die Beziehung zwischen Tree- und Blob Objekten funktioniert wie die von Ordnern und Dateien in einem Dateisystem. Dort können sich in einem Ordner Dateien sowie Unterordner befinden. Analog können Blob- sowie weitere Tree Objekte zu einem Tree Objekt gehören. Wenn Dateien zum Index eines Repositories hinzugefügt werden, wird aus jedem Ordner, der zu der hinzugefügten Dateistruktur gehört, ein Tree-Objekt erzeugt. Dieses Objekt erhält, wie das Blob Objekt, einen 40- Stelligen Hash, welches das Objekt intern adressiert. In jedem Tree Objekt werden die Objekt-Typen, Hashes und Namen der Objekte gespeichert, welche sich im dazugehörigen Ordner befinden. Auch Tree Objekte werden im `.git/objects` Ordner gespeichert [Cha09].

3.5.3 Das Commit Objekt

Für jeden Commit, der im Repository durchgeführt wird, wird ein Commit Objekt erstellt. Dieses Objekt enthält Informationen über den Zustand des Repositories zum Zeitpunkt des Commits. Ein Commit Objekt enthält üblicherweise die folgenden Informationen: Der Name und die Emailadresse des Authors. Eine Referenz auf ein Tree Objekt, welches den Zustand der Dateien und Ordner zum Commitzeitpunkt widerspiegelt. Ein 40- Stelliger Hash, welcher diesen Commit eindeutig identifiziert. Einen Zeitstempel von dem Zeitpunkt des Commits. Einen Kommentar, der die Änderungen, die mit diesem Commit in das Versionsverwaltungssystem eingecheckt werden kommentiert. Außerdem enthält ein Commit Objekt einen Verweis auf das vorherige Commit Objekt, aus dem es hervorgegangen ist. Jedes Commit Objekt, bis auf das Objekt, welches beim initialen Commit erstellt wurde, enthält mindestens eine dieser Referenzen. Im Fall, dass es sich bei dem Commit um einen Merge handelt, werden alle Branches angegeben, aus welchen sich der Commit zusammensetzt. Erst durch diese Referenzen ist es möglich Änderungen im Projekt rückgängig machen zu können und die Versionshistorie einzelner Dateien nachzuverfolgen. Die Commit Objekte werden wie alle anderen Hauptobjekte auch, im `.git/objects` Ordner gespeichert [Cha09].

3.5.4 Beispiel

Abb. 3.2 zeigt einen Graphen, der die Beziehungen dieser 3 wichtigsten Git Objekte verdeutlicht. Gelbe Knoten repräsentieren Commit- , türkisfarbene Knoten Tree- und graue Knoten stellen Blob Objekte dar. Alle Knoten besitzen eine Adresse, welche aus den ersten sechs Stellen ihres Hashs besteht. In diesem Beispiel wurde im ersten Commit ein Ordner erstellt, in welchem sich die Datei *test.txt* befindet. Im zweiten Commit wurde diese Datei verändert. Dabei wurde ein neuer Blob erstellt, der die neue Version der Datei *test.txt* beinhaltet. Weiterhin wurde im zweiten Commit eine neue Datei mit dem Namen *new.txt* erstellt. Im dritten Commit wurden die Dateien *text.txt* und *new.txt* nicht verändert. Das Tree Objekt des Commits zeigt weiterhin auf die selben Blob Objekte wie der Tree aus dem zweiten Commit. Allerdings wurde die ursprüngliche Version von *text.txt* aus Commit

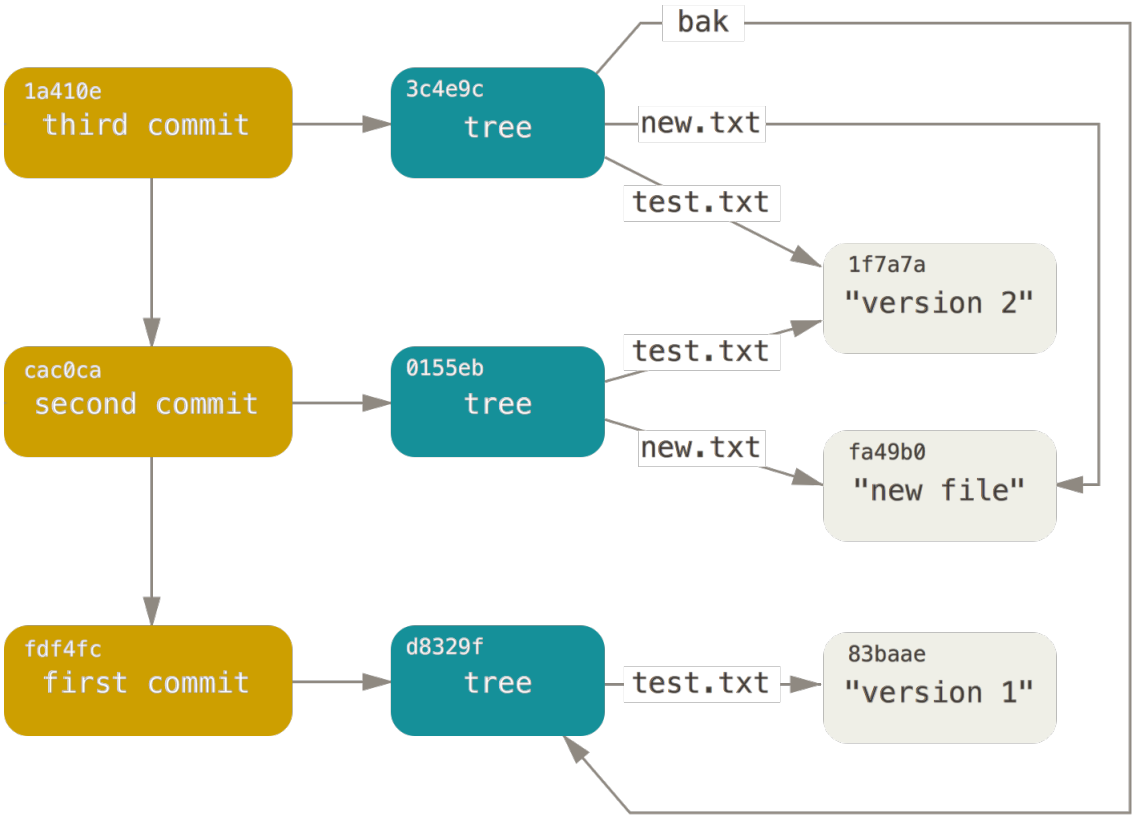


Abbildung 3.2: Git Objekte [Cha09]

eins im Ordner *bak* abgespeichert. Dafür wurde im Tree Objekt des dritten Commits eine Referenz auf das Tree Objekt von Commit eins hinterlegt [Cha09].

3.6 Git Referenzen

Referenzobjekte helfen bei der Navigation durch den Objektgraphen. Sie sind im Ordner `.git/refs` hinterlegt. Im folgenden werden die wichtigsten dieser Objekte kurz angesprochen [Cha09].

3.6.1 Heads

Heads sind Referenzen zu aktuellen Commit Objekten. Jeder Branch eines Repositories enthält eine Head Referenz neusten Commit Objekt dieses Branchs [Cha09].

3.6.2 Remotes

Remotes sind Referenzen zu Adressen, wo Versionen des Repositories hinterlegt sind. Auf diese Remotes kann das Repository hochgeladen (gepusht) oder von ihnen heruntergeladen (gepullt) werden [Cha09].

3.6.3 Tags

Obwohl Tags im `.git/refs` Ordner gespeichert sind, gehören sie zu der Gruppe der Git Objekte. Ein Tag zeigt auf ein Commit Objekt. Sie erleichtern die Interaktion mit Commit Objekten, da um an das Commit Objekt zu gelangen nicht die ganze Commitkette ausgehend von der Head Referenz durchlaufen werden muss [Cha09].

4 Die Extraktionswerkzeuge

4.1 Recherche

Der vorherige Abschnitt zeigte wie die Metadaten von Git organisiert und aufgebaut sind. Bei einer Recherche nach Analysewerkzeugen, die diese Daten analysieren und aus ihnen Informationen ableiten, zeigte sich relativ schnell, dass der Markt für solche Analysewerkzeuge relativ klein ist. Im folgenden werden drei dieser Werkzeuge vorgestellt: StatsSVN für Svn Repositories sowie GitStats und Gitinspector für Git Versionsverwaltungssysteme.

4.2 StatSVN

StatSVN [Sta] ist ein Open- Source- Werkzeug, welches aus den Informationen eines Subversion Repositories eine große Anzahl an Statistiken, Tabellen und Grafiken generiert. Das Projekt befindet sich seit 2006 in Entwicklung, die letzte Veröffentlichung der Betaversion 0.7 stammt allerdings aus dem Jahr 2010. Soweit erkennbar ist, wird im Moment an dem Projekt nicht mehr aktiv weiterentwickelt.

4.2.1 Probleme

Zu Testzwecken wurden zwei Open Source Projekte mit dem Tool analysiert: Das Repository der Programmiersprache Ruby sowie Joda-Time, eine Java Bibliothek die die vorhandene Datum- und Zeitfunktionalität verbessert und erweitert. Beim Testen offenbarte sich eine grundlegende Schwäche des Werkzeugs, nämlich dass nicht alle Commits im Repository analysiert werden konnten, was dazu führte, dass in den Statistiken einige Dateien nicht berücksichtigt wurden und somit die daraus resultierenden Daten nicht genau genug waren.

4.3 GitStats

Auch GitStats [Gitb] ist ein Open- Source Programm, welches Informationen und Statistiken aus einem Git Repository ableiten kann. Es ist seit 2007 in Entwicklung und erhält auch heute noch regelmäßig Updates.

4.3.1 Probleme

Leider hat das Programm für die Verwendung für diesen Anwendungsfall zwei große Nachteile. Zum einen besitzt das Tool als Ausgabeformat für die Statistiken nur HTML, was die Weiterverarbeitung der Daten in diesem Anwendungsfall schwieriger macht. Außerdem sind viele der generierten Daten für den Anwendungsfall nicht relevant, während wichtige Features für UniMoDoc fehlen. So werden Dateien im Projektarchiv beispielsweise nicht ihren Autoren zugeordnet, ein Feature, welches, wie in Abschnitt 4.5 gezeigt wird, für die Dokumentation sehr relevant sein kann. Die wenigen Daten die relevant sind, können mit dem, im nächsten Abschnitt vorgestellten Tool Gitinspector ebenfalls extrahiert werden. Somit schied auch dieses Tool für die Verwendung im Projekt aus.

4.4 Gitinspector

Gitinspector [Gita] ist ein, unter der GNU GPL v3 Lizenz, angebotenes Tool zur Erstellung von Statistiken aus Git Repositories. Es wurde ursprünglich für die Analyse von Repositories der Studenten aus der Chalmers University of Technology sowie der Gothenburg University entwickelt. Auch heute erhält das Tool regelmäßig Updates, die aktuelle Version 0.3.2 wurde am 15. Januar 2014 veröffentlicht. Gitinspector besitzt Git und Python als Abhängigkeiten.

4.4.1 Features

Gitinspector ist von den betrachteten Analysewerkzeugen das einzige, welches auf mehreren Threads arbeitet. Dies sorgt dafür, das auch die Analyse von sehr großen Repositories mit vielen Commits relativ schnell durchgeführt wird. Als Ausgabeformate bietet Gitinspector HTML, XML und Klartext. Davon ist vor allem XML relevant, da XML Dateien leicht von Computerprogrammen weiterverarbeitet werden können. Weiterhin wird eine mächtige Filterfunktion angeboten. So ist es beispielsweise möglich bestimmte Dateitypen im Repository zu ignorieren, beispielsweise Dateien die beim Compilieren erzeugt werden. So fließen diese Dateien nicht in die generierte Statistik ein und verfälschen sie nicht.

4.4.2 Bericht

Im folgenden wird die HTML Version eines mit Gitinspector erzeugten Berichts für das Projektarchiv des Open- Source Editors Atom [Ato] gezeigt. Dieser komprimiert angezeigte Bericht ist in fünf Teile aufgeteilt.

Eincheck Historie pro Entwickler

In Abb. 4.1 sind alle Autoren nach ihrem Anteil an Codeänderungen im Repository aufgelistet. Weiterhin sind dort ihre Anzahl an Commits und die Anzahl an Dateiänderungen und Löschungen zu finden.

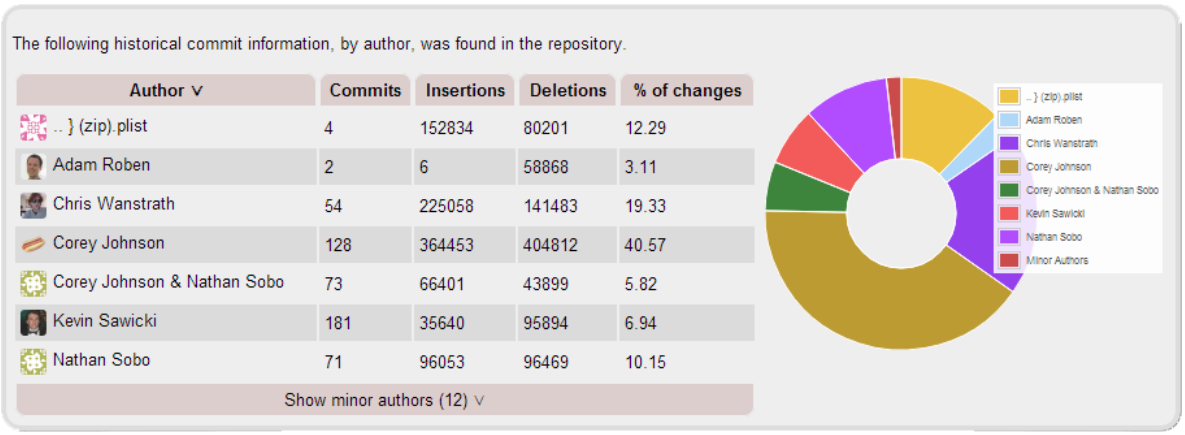


Abbildung 4.1: Eincheck Historie pro Entwickler

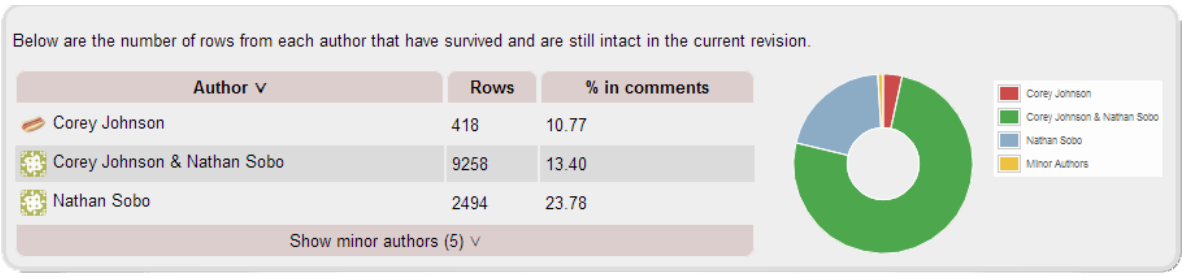


Abbildung 4.2: Code eines Entwicklers im aktuellen Commit

Code eines Entwicklers im aktuellen Commit

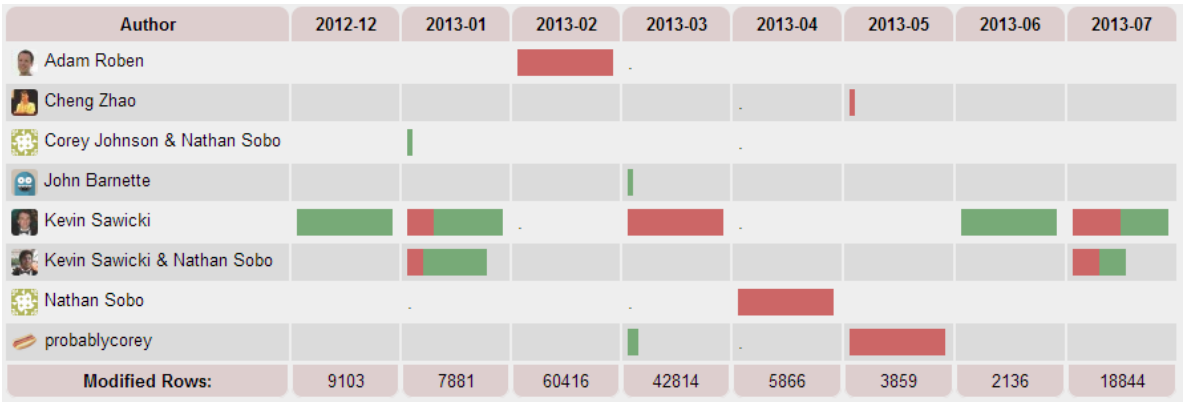
In Abb. 4.2 sind die Anzahl an Codezeilen pro Author aufgelistet, die noch in der aktuellen Revision vorhanden sind. Es wird auch gezeigt, wie viel Prozent dieser Codezeilen aus Kommentaren besteht.

Aktivitätenprotokoll

In Abb. 4.3 sind alle Monate aus dem Projektzeitraum aufgelistet. Für jeden Monat wird angegeben, welcher Author wieviel am Projekt gearbeitet hat. Die roten Anteile sind Löschungen, die grünen Einfügungen und Änderungen. Weiterhin wird die Anzahl der Codezeilen angegeben, die in diesem Monat verändert wurden.

Dateiverantwortlichkeiten

Der nächste Abschnitt im Bericht in Abb. 4.4 listet für jeden Author im Projekt für jede Datei, in der er Code eingefügt hat, auf wie viele Zeilen ausführbaren Code davon ihm zugeordnet werden.



Author	2012-12	2013-01	2013-02	2013-03	2013-04	2013-05	2013-06	2013-07
Adam Roben								
Cheng Zhao								
Corey Johnson & Nathan Sobo								
John Barnette								
Kevin Sawicki								
Kevin Sawicki & Nathan Sobo								
Nathan Sobo								
probablycorey								
Modified Rows:	9103	7881	60416	42814	5866	3859	2136	18844

Abbildung 4.3: Aktivitätenprotokoll

Abbildung 4.4: Dateiverantwortlichkeiten

Dateitypen

In Abb. 4.5, dem letzten Teil des Berichts werden alle Dateitypen die sich im Projektarchiv befinden aufgelistet. Die fett gekennzeichneten Dateitypen wurden für den Bericht berücksichtigt. Alle anderen wurden ignoriert.

4.4.3 Modifikation

Die im Projekt verwendete Version 0.3.2 von Gitinspector musste noch für den Anwendungsfall angepasst werden. So wurden für den Abschnitt, wo die Dateiverantwortlichkeiten der einzelnen Entwickler angegeben wurden nur die 10 Dateien, an denen die meisten Zeilen verändert wurden angezeigt. Um alle Dateien, an denen der Entwickler mitgearbeitet hatte anzuzeigen musste in der

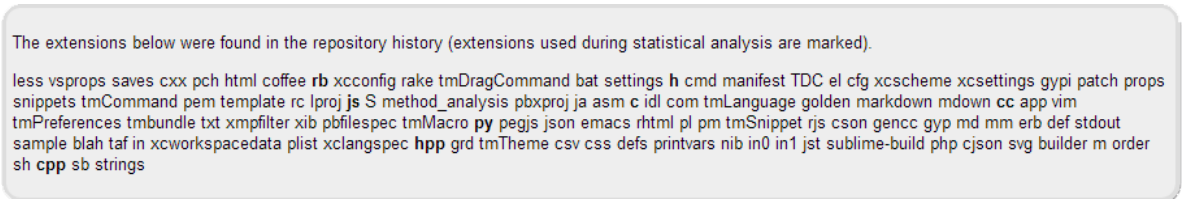


Abbildung 4.5: Dateitypen

Datei `responibilities.py` in Zeile 122 und 123 die Abbruchbedingung der for Schleife entfernt werden.

4.4.4 Performanz

Testweise wurde das Werkzeug auf einem System mit einem NovaBench Score von 1671 ausgeführt [Nov]. Als Testrepositories dienten zum einen das Git-Repository des Atom Editors (Stand 06.06.2014) mit 16018 commits und ca. 400 Dateien [Ato]. Außerdem wurde das Repository der Web Frontend Framework Bootstrap (Stand 06.06.2014) mit 9232 Commits und 328 Dateien getestet [Boo]. Zur Analyse der ersten Frameworks benötigte das System 20 Sekunden um eine XML Datei mit den Ergebnissen zu erstellen. Für die Analyse des Bootstrap Frameworks wurden 13 Sekunden gebraucht.

4.5 Verwendung der Daten in UniMoDoc

Von den von Gitinspector erzeugten Daten sind die folgenden relevant für eine Verwendung in UniMoDoc.

- Die Dateitypen, die im Projekt verwendet werden können benutzt werden, um die Daten für das Beschreibungselement Technology im globalen Modul zu ermitteln.
- Die Größe der einzelnen Pakte in eloc (Executable lines of code). Dies kann zum einen verwendet werden um das Beschreibungselement Size für alle Module zu bestimmten, weiterhin können die Daten für ein neues Beschreibungselement verwendet werden, welches in Abschnitt 4.5.1 vorgestellt wird.
- Die Autoren pro Datei. Mit diesen Analysedaten kann ermittelt werden wer von den Autoren an welcher Datei mitgearbeitet hat. Somit kann das Beschreibungselement Autor für jedes einzelne Modul ermittelt werden.
- Der Pfad jeder einzelnen Datei. Da für jede Datei im Projekt ihr absoluter Pfad bekannt ist, kann das Beschreibungselement Source bestimmt und überprüft werden, welches den absoluten Pfad eines jeden Moduls angibt.

- Weiterhin kann der Pfad, der für jede Datei im Projekt bekannt ist, dazu verwendet werden um zu überprüfen ob ein Modul überhaupt noch im Projekt vorhanden ist. Dies kann in einem weiteren neuen Beschreibungselement *Synchronization Status*, welches im nächsten Abschnitt beschrieben wird, dokumentiert werden.

4.5.1 Neue Beschreibungselemente

Durch die ermittelten Daten können zwei neue Beschreibungselemente für die Module mit Informationen gefüllt werden. Diese werden in den folgenden Abschnitten vorgestellt.

Größe der Unterpakete (Subpackage Size)

Da das Beschreibungselement Size für jedes Modul ermittelt werden kann, ist es möglich für jedes Modul auch die Gesamtgröße seiner Unterpakete zu ermitteln. Diese Gesamtgröße setzt sich aus der Größe des Moduls plus der Größe aller seiner Unterpakete zusammen. Zum Beispiel existieren zwei Module: `de.test.data` und `de.test.data.constants`. Die Größe des Beschreibungselements Subpackage Size für `de.test.data` setzt sich aus der Größe von `de.test.data` sowie der von `de.test.data.constants` zusammen. Subpackage Size von `de.test.data.constants` entspricht seiner eigenen Größe, da dieses Modul keine weiteren Unterpakete hat.

Synchronization Status

Ein weiteres neues Beschreibungselement ist Synchronisation Status. In diesem Element wird der Zustand des Paktes, wie ihn die Synchronisation ermittelt hat, dokumentiert. Er besitzt drei mögliche Zustände: *Module not found*, *Module freshly added* und *leer*. Ein Modul hat den Synchonization Status von *Module not found* wenn das Modul während des Synchronisationsvorgangs nicht im Repository gefunden werden konnte. Das kann zum Beispiel daran liegen, dass der Ordner, in dem sich der Inhalt dieses Moduls befand, bereits gelöscht wurde, diese Änderung aber noch nicht dokumentiert wurde. Wenn ein Modul einen Synchronization Status von *Module freshly added* besitzt, wurde es während der letzten Synchronisation neu in der Dokumentation hinzugefügt. Dies kann dann vorkommen, wenn ein neues Modul im Projekt hinzugefügt wurde, dies aber nicht in der Dokumentation vermerkt wurde. Wenn ein Modul einen leeren Synchronization Status besitzt, wurde nichts auffälliges am Modul während der letzten Synchronisation bemerkt. Das Modul wurde gefunden und alle Beschreibungselemente die synchronisiert werden können befinden sich auf dem neusten Stand. Für jeden Zustand, den dieses Beschreibungselement besitzen kann, können auf dem Modul verschiedene zusätzliche Operationen durchgeführt werden. Diese Operationen werden im nächsten Kapitel vorgestellt.

5 Synchronisation

5.1 Allgemeiner Ablauf

In Abb. 5.1 wird dargestellt wie die im Rahmen der Bachelorarbeit implementierte Synchronisation funktioniert. Im linken Teil der Grafik wird der Zustand der Dokumentation vor der Synchronisation dargestellt. Dort gehören zu einer Dokumentation Liste von Modulen. Nachdem die Synchronisation durchgeführt wurde, wird das Beschreibungselement Synchronisation Status einer von den drei Klassen zugeordnet, die in Abschnitt 4.5.1 vorgestellt wurden. Für die neu aus dem Repository hinzugefügten Module ist es möglich die Markierung zu entfernen. Für die nicht mehr im Repository gefundenen Module ist es zusätzlich auch noch möglich die Module zu löschen. Die Icons für diese Operationen werden neben dem Edit und Delete Icon in der Tab Navigation angezeigt. Wenn die Synchronisation neu gestartet wird, werden alle Module erneut einer Klasse zugeordnet. Die Anwendungsfälle dieser Operationen werden in diesem Kapitel vorgestellt.

Bei genauerer Betrachtung der Daten, die zur Verwendung in UniMoDoc von Gitinspector ermittelt wurden und in Abschnitt 4.5 vorgestellt wurden, wird klar, dass sich diese von einem Computer genau

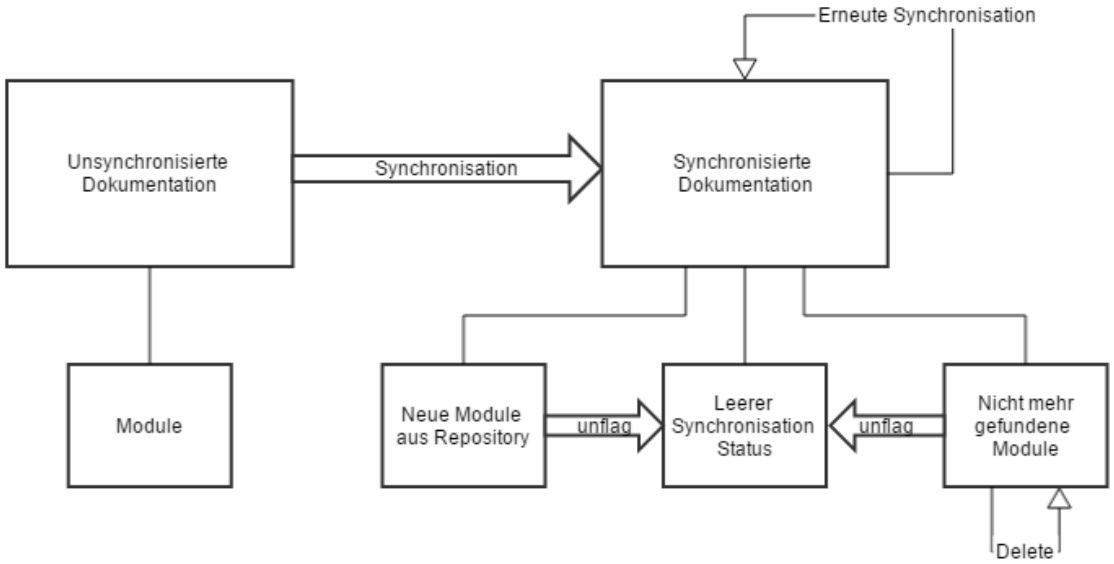


Abbildung 5.1: Allgemeiner Synchonsationsablauf

ermitteln lassen. Wenn es keine Fehler bei der Ermittlung dieser Daten gibt, kann sich der Benutzer darauf verlassen, dass die Daten korrekt sind und muss sich nicht weiter damit beschäftigen. Auch lassen diese Daten keinen Interpretationsspielraum zu, da sie entweder stimmen können oder nicht. Aus diesem Grund ist es nicht nötig dem Benutzer alle Veränderungen die in der Synchronisation durchgeführt wurden einzeln aufzulisten und ihm die Möglichkeit zu geben die Aktualisierungen bei einzelnen Beschreibungselementen rückgängig zu machen. Somit sind dafür keine Anwendungsfälle nötig.

5.1.1 Voraussetzungen für die Erstellung einer Gitinspector XML Datei

Aus Gründen, die im nächsten Kapitel genauer erläutert werden, findet die Analyse eines Repositories mit Gitinspector außerhalb von UniMoDoc statt. Folgende Voraussetzungen sind dabei zu beachten.

- Die modifizierte Version von UniMoDoc, zusammen mit den Skripten zur Ausführung auf der Konsole, müssen unter `https://sourceforge.net/projects/gitinspector032m/` bezogen werden
- Git muss installiert und in der PATH Variable hinterlegt und somit von der Konsole aus ausführbar sein. (Download unter: `http://git-scm.com/downloads`)
- Python ab der Version 2.6 muss installiert sein. (Download unter: `https://www.python.org/downloads/`)

5.2 Anwendungsfälle

5.2.1 Einleitung

Abb. 5.2 zeigt Übersicht mit den sieben Use-Cases, die in diesem Kapitel vorgestellt werden. Da UniMoDoc neben dem Benutzer, der die Dokumentation durchführt, keine zusätzlichen Benutzerrollen verwendet, ist dieser in der Lage auf alle hier vorgestellten Funktionen zuzugreifen.

Use Cases

Abb. 5.3 bis Abb. 5.9 zeigen die in Abb. 5.2 vorgestellten Use-Cases im Detail.

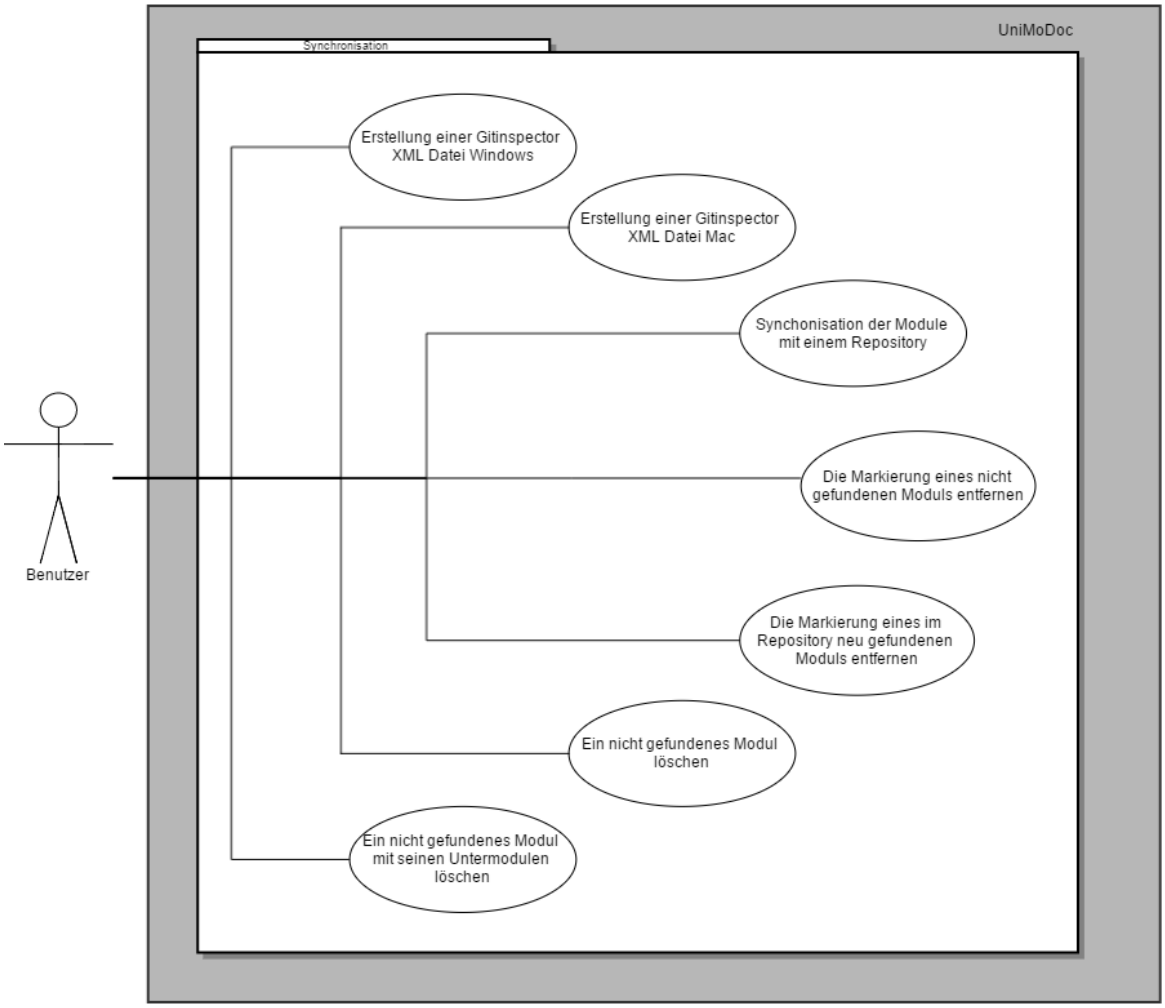


Abbildung 5.2: Übersicht der Use-Cases

Ziel:		Der Benutzer möchte unter Windows eine Gitinspector XML Datei für die Verwendung in UniMoDoc erstellen.	
Akteur:		**Benutzer**	
Beschreibung:		Der Benutzer startet das für Windows vorbereitete Skript, wählt ein Repository zur Analyse aus und erhält zum Abschluss eine von Gitinspector erstellte XML Datei für das gewählte Repository.	
Normalablauf			
Vorbedingung:		Die Voraussetzungen aus Abschnitt 5.1.1 müssen erfüllt sein. Der Benutzer befindet sich in einem Windows Betriebssystem. Der Benutzer hat die Datei windows.bat mit einem Doppelklick gestartet.	
1	Skript	Das Skript erwartet vom Benutzer eine Pfadeingabe zu einem Git-Repository.	
		Bedingung für Sonderfall: Der Benutzer bricht den Vorgang durch Schließen des Fensters ab.	Alternativablauf 1a
2	Benutzer	Der Benutzer gibt einen Pfad an und bestätigt die Eingabe durch Drücken der Enter Taste.	
3	Skript	Das Skript überprüft die Pfadeingabe und analysiert das Repository.	
		Bedingung für Sonderfall: Der eingegebene Pfad zeigt nicht zu einem validen Git-Repository.	Alternativablauf 1b
Nachbedingung:		Das Skript erstellte die Datei statistics.xml im selben Ordner wo sich die Datei windows.bat befand.	
Alternativablauf 1a			
Vorbedingung:		Der Benutzer bricht den Vorgang durch Schließen des Fensters ab.	
Nachbedingung:		Das Fenster wurde geschlossen. Es wurde keine Analyse vorgenommen.	
Abbruch			
Alternativablauf 1b			
Vorbedingung:		Der eingegebene Pfad zeigt nicht zu einem validen Git Repository.	
1b1	Skript	Das Skript gibt eine Warnung aus und schließt das Fenster.	
Nachbedingung:		Das Fenster wurde geschlossen. Es wurde keine Analyse vorgenommen.	
Abbruch			

Abbildung 5.3: Erstellung einer Gitinspector XML Datei unter Windows

Ziel:		Der Benutzer möchte unter Mac OS eine Gitinspector XML Datei für die Verwendung in UniMoDoc erstellen.	
Akteur:		**Benutzer**	
Beschreibung:		Der Benutzer startet das für Mac OS vorbereitete Skript, wählt ein Repository zur Analyse aus und erhält zum Abschluss eine von Gitinspector erstellte XML Datei für das gewählte Repository.	
Normalablauf			
Vorbedingung:		Die Voraussetzungen aus Abschnitt 5.1.1 müssen erfüllt sein. Der Benutzer befindet sich in einem Mac Betriebssystem. Der Benutzer hat das Terminal geöffnet und ist zum Ordner, wo sich die Datei mac.sh befindet, navigiert. Der Benutzer startete das Skript mit dem Befehl: sh mac.sh	
1	Skript	Das Skript erwartet vom Benutzer eine Pfadeingabe zu einem Git-Repository.	
		Bedingung für Sonderfall: Der Benutzer bricht den Vorgang durch Schließen des Fensters ab.	Alternativablauf 2a
2	Benutzer	Der Benutzer gibt einen Pfad an und bestätigt die Eingabe durch Drücken der Enter Taste.	
3	Skript	Das Skript überprüft die Pfadeingabe und analysiert das Repository.	
		Bedingung für Sonderfall: Der eingegebene Pfad zeigt nicht zu einem validen Git-Repository.	Alternativablauf 2b
Nachbedingung:		Das Skript erstellte die Datei statistics.xml im selben Ordner wo sich die Datei mac.sh befand.	
Alternativablauf 2a			
Vorbedingung:		Der Benutzer bricht den Vorgang durch Schließen des Fensters ab.	
Nachbedingung:		Das Fenster wurde geschlossen. Es wurde keine Analyse vorgenommen.	
Abbruch			
Alternativablauf 2b			
Vorbedingung:		Der eingegebene Pfad zeigt nicht zu einem validen Git Repository.	
2b1	Skript	Das Skript gibt eine Warnung aus und schließt das Fenster.	
Nachbedingung:		Das Fenster wurde geschlossen. Es wurde keine Analyse vorgenommen.	
Abbruch			

Abbildung 5.4: Erstellung einer Gitinspector XML Datei unter Mac OS

Ziel:		Der Benutzer möchte eine Gitinspector XML Datei in UniMoDoc einlesen und den Stand des Repositories mit dem Stand der Dokumentation synchronisieren.
Akteur:		**Benutzer**
Beschreibung:		Der Benutzer mit dem Synchronisationsdialog den Pfad der XML Datei aus und startet die Synchronisation. Nachdem diese beendet wurde, werden die Änderungen in UniMoDoc angezeigt.
Normalablauf		
Vorbedingung:		Der Benutzer hat UniMoDoc in einem Browser geöffnet.
1	Benutzer	Der Benutzer klickt auf den Synchronisationsbutton, der sich im rechten Teil der Titelleiste befindet.
2	UniMoDoc	UniMoDoc zeigt dem Benutzer ein Dialogfeld mit einem Dateiauswahldialog und einem Upload Button.
		Bedingung für Sonderfall: Der Benutzer bricht den Vorgang durch Schließen des Popup Dialogs ab. — Alternativablauf 3a
3	Benutzer	Der Benutzer klickt auf den Button mit der Aufschrift „Datei auswählen“, wählt eine, durch Gitinspector erzeugte, XML Datei aus und startet die Synchronisation durch Klicken auf den Button mit der Aufschrift „Upload XML File“.
4	UniMoDoc	UniMoDoc überprüft die Pfadeingabe und startet die Synchronisation. Für die Synchronisationsdauer erscheint ein Dialog, der den Benutzer über die laufende Synchronisation informiert.
		Bedingung für Sonderfall: Der eingegebene Pfad zeigt nicht zu einer XML Datei. — Alternativablauf 3b
		Bedingung für Sonderfall: Der eingegebene Pfad zeigt nicht auf eine valide Gitinspector XML Datei. — Alternativablauf 3c
Nachbedingung:		Die Synchronisation wurde erfolgreich abgeschlossen. Die Module wurden mit den aktuellen Informationen aus dem Repositoy befüllt. Es wird eine Statistik über die gemachten Änderungen angezeigt.
Alternativablauf 3a		
Vorbedingung:		Der Benutzer bricht den Vorgang durch Schließen des Popup Dialogs ab.
Nachbedingung:		Das Fenster wurde geschlossen. Es wurde keine Synchronisation durchgeführt.
Abbruch		
Alternativablauf 3b		
Vorbedingung:		Der eingegebene Pfad zeigt nicht zu einer XML Datei.
3b1	UniMoDoc	UniMoDoc gibt eine Warnung im Browser aus und beendet den Synchronisationsvorgang.
Nachbedingung:		Die Warnung wird im Browser gezeigt, es wurde keine Synchronisation durchgeführt.
Abbruch		
Alternativablauf 3c		
Vorbedingung:		Der eingegebene Pfad zeigt nicht auf eine valide Gitinspector XML Datei.
3c1	UniMoDoc	UniMoDoc gibt eine Warnung auf der Konsole aus und beendet den Synchronisationsvorgang.
Nachbedingung:		Die Warnung wird auf der Konsole angezeigt, es wurde keine Synchronisation durchgeführt.
Abbruch		

Abbildung 5.5: Synchronisation der Module mit einem Repository

Ziel:		Der Benutzer möchte das Markierungssymbol eines Moduls in der linken Seitenleiste löschen.	
Akteur:		**Benutzer**	
Beschreibung:		Der Benutzer wählt den „Modul nicht gefunden“ Dialog durch Klicken auf das Icon mit dem Fragezeichen im rechten Teil des Tab Navigationsbereichs aus. Anschließend entfernt er die Markierung durch einen Klick auf den Button mit der Aufschrift „Unflag“.	
Normalablauf			
Vorbedingung:		Der Benutzer hat UniMoDoc in einem Browser geöffnet. Der Benutzer hat ein Modul, welches während einer Synchronisation nicht gefunden wurde, und somit mit einem Fragezeichensymbol versehen ist, ausgewählt.	
1	Benutzer	Der Benutzer klickt auf das Icon mit dem Fragezeichen im rechten Teil des Tab Navigationsbereichs.	
2	UniMoDoc	UniMoDoc zeigt dem Benutzer ein Dialogfeld mit der Information, dass das Modul bei der Synchronisation nicht gefunden worden ist.	
		Bedingung für Sonderfall: Der Benutzer bricht den Vorgang durch einen Klick auf den Button mit der Aufschrift „Cancel“ ab.	Alternativablauf 4a
3	Benutzer	Der Benutzer klickt auf den Button mit der Aufschrift „Unflag“	
4	UniMoDoc	UniMoDoc entfernt die Markierung und schließt das Dialogfenster.	
Nachbedingung:		Die Markierung wurde entfernt. Das Dialogfenster wurde geschlossen. Das Icon mit dem Fragezeichensymbol, mit dem dieser Vorgang gestartet wurde, ist nicht mehr sichtbar. Der Inhalt des Beschreibungselement „Synchronisation Status“ wurde entfernt.	
Alternativablauf 4a			
Vorbedingung:		Der Benutzer bricht den Vorgang durch einen Klick auf den Button mit der Aufschrift „Cancel“ ab.	
Nachbedingung:		Das Dialogfenster wurde geschlossen. Die Markierung wurde nicht entfernt.	
Abbruch			

Abbildung 5.6: Die Markierung eines nicht gefundenen Moduls entfernen

Ziel:		Der Benutzer möchte ein Modul, welches bei der Synchronisation nicht gefunden wurde, entfernen.	
Akteur:		**Benutzer**	
Beschreibung:		Der Benutzer wählt den „Modul nicht gefunden“ Dialog durch Klicken auf das Icon mit dem Fragezeichen im rechten Teil des Tab Navigationsbereichs aus. Anschließend entfernt er das Modul mit einem Klick auf den Button „Delete“	
Normalablauf			
Vorbedingung:		Der Benutzer hat UniMoDoc in einem Browser geöffnet. Der Benutzer hat ein Modul, welches während einer Synchronisation nicht gefunden wurde, und somit mit einem Fragezeichensymbol versehen ist, ausgewählt.	
1	Benutzer	Der Benutzer klickt auf das Icon mit dem Fragezeichen im rechten Teil des Tab Navigationsbereichs.	
2	UniMoDoc	UniMoDoc zeigt dem Benutzer ein Dialogfeld mit der Information, dass das Modul bei der Synchronisation nicht gefunden worden ist.	
		Bedingung für Sonderfall: Der Benutzer bricht den Vorgang durch einen Klick auf den Button mit der Aufschrift „Cancel“ ab.	Alternativablauf 5a
3	Benutzer	Der Benutzer klickt auf den Button mit der Aufschrift „Delete“	
4	UniMoDoc	UniMoDoc entfernt das Modul und schließt das Dialogfenster.	
Nachbedingung:		Das Modul wurde gelöscht. Das Modul, welches sich in der linken Seitenleiste vor dem gelöschten Modul befand ist nun ausgewählt. Es wird eine Infomeldung angezeigt.	
Alternativablauf 5a			
Vorbedingung:		Der Benutzer bricht den Vorgang durch einen Klick auf den Button mit der Aufschrift „Cancel“ ab.	
Nachbedingung:		Das Dialogfenster wurde geschlossen. Das Modul wurde nicht gelöscht.	
Abbruch			

Abbildung 5.7: Ein nicht gefundenes Modul löschen

Ziel:		Der Benutzer möchte ein Modul, welches bei der Synchronisation nicht gefunden wurde, zusammen mit seinen Untermodulen entfernen.	
Akteur:		**Benutzer**	
Beschreibung:		Der Benutzer wählt den „Modul nicht gefunden“ Dialog durch Klicken auf das Icon mit dem Fragezeichen im rechten Teil des Tab Navigationsbereichs aus. Er bestätigt mit einem Klick auf die Option „Delete submodules“ dass er auch die Untermodule löschen möchte. Anschließend entfernt er das Modul mit seinen Untermodulen mit einem Klick auf den Button „Delete“	
Normalablauf			
Vorbedingung:		Der Benutzer hat UniMoDoc in einem Browser geöffnet. Der Benutzer hat ein Modul, welches während einer Synchronisation nicht gefunden wurde, und somit mit einem Fragezeichensymbol versehen ist, ausgewählt.	
1	Benutzer	Der Benutzer klickt auf das Icon mit dem Fragezeichen im rechten Teil des Tab Navigationsbereichs.	
2	UniMoDoc	UniMoDoc zeigt dem Benutzer ein Dialogfeld mit der Information, dass das Modul bei der Synchronisation nicht gefunden worden ist.	
		Bedingung für Sonderfall: Der Benutzer bricht den Vorgang durch einen Klick auf den Button mit der Aufschrift „Cancel“ ab.	Alternativablauf 6a
3	Benutzer	Der Benutzer klickt auf die Option „delete submodules“ Anschließend klickt er auf den Button mit der Aufschrift „Delete“	
4	UniMoDoc	UniMoDoc entfernt das Modul mit seinen Untermodulen und schließt das Dialogfenster.	
Nachbedingung:		Das Modul wurde zusammen mit seinen Untermodulen gelöscht. Es wird eine Informationsmeldung angezeigt.	
Alternativablauf 6a			
Vorbedingung:		Der Benutzer bricht den Vorgang durch einen Klick auf den Button mit der Aufschrift „Cancel“ ab.	
Nachbedingung:		Das Dialogfenster wurde geschlossen. Das Modul und seine Untermodule wurden nicht gelöscht.	
Abbruch			

Abbildung 5.8: Ein nicht gefundenes Modul mit seinen Untermodulen löschen

Ziel:		Der Benutzer möchte das Markierungssymbol eines Moduls in der linken Seitenleiste löschen.	
Akteur:		**Benutzer**	
Beschreibung:		Der Benutzer wählt den „Modul neu hinzugefügt“ Dialog durch Klicken auf das Icon mit dem Kreissymbol im rechten Teil des Tab Navigationsbereichs aus. Anschließend entfernt er die Markierung durch einen Klick auf den Button mit der Aufschrift „Unflag“.	
Normalablauf			
Vorbedingung:		Der Benutzer hat UniMoDoc in einem Browser geöffnet. Der Benutzer hat ein Modul, welches während einer Synchronisation neu hinzugefügt wurde, und somit mit einem Kreissymbol versehen ist, ausgewählt.	
1	Benutzer	Der Benutzer klickt auf das Icon mit dem Kreissymbol im rechten Teil des Tab Navigationsbereichs.	
2	UniMoDoc	UniMoDoc zeigt dem Benutzer ein Dialogfeld mit der Information, dass das Modul bei der Synchronisation neu hinzugefügt worden ist.	
		Bedingung für Sonderfall: Der Benutzer bricht den Vorgang durch einen Klick auf den Button mit der Aufschrift „Cancel“ ab.	Alternativablauf 7a
3	Benutzer	Der Benutzer klickt auf den Button mit der Aufschrift „Unflag“	
4	UniMoDoc	UniMoDoc entfernt die Markierung und schließt das Dialogfenster.	
Nachbedingung:		Die Markierung wurde entfernt. Das Dialogfenster wurde geschlossen. Das Icon mit dem Kreissymbol, mit dem dieser Vorgang gestartet wurde, ist nicht mehr sichtbar. Der Inhalt des Beschreibungselement „Synchronisation Status“ wurde entfernt.	
Alternativablauf 7a			
Vorbedingung:		Der Benutzer bricht den Vorgang durch einen Klick auf den Button mit der Aufschrift „Cancel“ ab.	
Nachbedingung:		Das Dialogfenster wurde geschlossen. Die Markierung wurde nicht entfernt.	
Abbruch			

Abbildung 5.9: Die Markierung eines im Repository neu gefundenen Moduls entfernen

6 Die Implementierung

6.1 Java Server Faces

Die Java Server Faces Technologie Framework zur Erstellung von Benutzeroberflächen für Java basierte Webanwendungen [Jav]. Mit diesem Framework ist es mit relativ wenig Aufwand möglich:

- User Interface Komponenten auf einer Website rendern zu lassen
- Events, die von diesen Komponenten ausgelöst werden auf mit Anwendungscode, der sich auf dem Server befindet, interagieren zu lassen
- UI Komponenten an Daten auf dem Server zu binden

Die JSF Komponenten können mit sogenannten Beans interagieren. Diese stellen eine Verbindung zwischen dem Präsentationslayer und dem Layer, wo die Geschäftslogik realisiert wird, dar. Beans sind Java Klassen, die dem *JavaBeans*- Standard genügen müssen [Mar]. Innerhalb einer Bean können dann andere Teile des Programms aufgerufen werden, die dann zum Beispiel Daten zur Befüllung von UI- Komponenten an das Präsentationslayer der Anwendung liefern.

6.2 Programmierstil

In der Implementierung wurde darauf geachtet, das Best Practices beim Programmierstil eingehalten wurden, um lesbaren, erweiterbaren und wiederverwendbaren Code zu erzeugen. Konkret wurde unter anderem auf aussagekräftige Kommentierung, gute Codeformatierung, konsistente Benennung der Bezeichner, die Verwendung von JavaDoc Kommentaren und gutes Logging geachtet um die Lesbarkeit des Codes hoch zu halten. Die von JSF vorgegebene Trennung von Präsentation und Applicationlogic sorgt weiterhin für eine gut strukturierte Code Base.

6.3 Implementierung

Bei der Implementierung von UniMoDoc wurde eine Bean mit dem Namen *SyncBean* erstellt, durch die die Interaktion mit dem User Interface geschieht. In dieser Klasse werden Methoden zur Ausführung der Synchronisation (Abb. 5.5) sowie dem Entfernen der Markierungen (Abb. 5.6 und Abb. 5.9) bereitgestellt. Die anderen Use Cases finden, wie bei der Erstelltung der Gitinspector XML Dateien, außerhalb des Programms statt oder es wurden Funktionalitäten verwendet, die schon an anderen Stellen im Programm implementiert wurden.

Die Funktionalität für die Entfernung der Markierung ist relativ simpel. Als Parameter wird der Methode ein String mit der ID des zu modifizierenden Moduls mitgegeben. Mit dessen Hilfe kann im Modulbaum das Beschreibungselement *syncStatus* des entsprechenden Moduls zurückgesetzt werden. Das führt dazu, dass die entsprechende Markierung nicht mehr angezeigt wird, da diese abhängig von dem Wert des Beschreibungselements angezeigt werden.

Die Synchronisation findet ihn mehreren Abschnitten statt, die im folgenden kurz erläutert werden.

1. Upload der Gitinspector XML Datei: Die Gitinspector XML Datei kann im User Interface über einen Dateiauswahldialog hochgeladen werden. Bevor die Klasse weiter verarbeitet werden kann, prüft die Klasse `FileUploadValidatorSync` aus dem Validatorpaket die Dateigröße, den Dateityp sowie die Länge des Dateinamens. Die Datei darf maximal 5MB groß sein und nur einen MIME-Type von `text/xml` oder `application/xml` besitzen. Der Dateiname darf maximal 50 Zeichen lang sein. Ist die Überprüfung erfolgreich, wird mit dem nächsten Schritt fortgefahren, bei einer nicht erfolgreichen Validierung wird dem Benutzer auf dem Browser eine entsprechende Fehlermeldung ausgegeben.
2. Überprüfung der XML Datei: Im nächsten Schritt wird überprüft ob sich die übergebene XML Datei tatsächlich um eine Gitinspector XML Datei handelt. Dabei wird zum einen die Struktur der XML Datei grob überprüft, weiterhin wird die Versionsnummer der Gitinspector Version ausgelesen, die die Statistik erstellt hat um Versionskonflikte zu vermeiden. Schlägt die Überprüfung fehl, wird auf der Konsole eine entsprechende Fehlermeldung ausgegeben.
3. Traversierung der XML Datei und Ermittlung der Informationen: In diesem Schritt wird die XML Datei durchlaufen und es werden die Beschreibungselemente aktualisiert. Es wird nach neuen und nicht gefundenen Dateien gesucht und es werden die Technologien ermittelt, die im Projekt verwendet werden. Die Statistiken werden gesammelt und in einer Nachricht im Browser ausgegeben. Die Updates in den Beschreibungselementen werden auch im Browser direkt angezeigt.

6.4 Probleme

Die Implementierung der Synchronisationsfunktion lief relativ Problemfrei ab. Die wenigen Probleme, die dennoch auftraten, sind in den folgenden Abschnitten zusammen mit ihren Lösungen dokumentiert.

6.4.1 Parsing der Gitinspector XML Datei

Ursprünglich war geplant die XML Datei aus Gitinspector so zu parsen, wie das bei der Verarbeitung des XML Imputs für UniMoDoc Projektdateien in der Klasse `XMLImporter` im Paket `de.modulspezifikation.importer` durchgeführt wurde. Dort wurde die Baumförmige Knotenstruktur der mit dem `DocumentBuilder` geparsten XML Datei manuell traversiert. Bei der Anaylse der von Gitinspector erzeugten XML Datei wurde deutlich, dass sich viele der logisch zusammenhängenden Informationen in verschiedenen Teilen des XML Baums befanden. Dadurch musste, dieser

Baum sehr oft auf unterschiedliche Arten traversiert werden. Dies erzeugte eine große Menge an unübersichtlichem Code, welcher sehr schwer zu lesen war.

Um das Problem zu lösen, wurde die aufwendige XML Knotentraversierung von der in Java integrierten Bibliothek XPath übernommen. Mit dieser Abfragesprache können auch sehr komplexe Baumtraversierungen übersichtlich und mit wenigen Codezeilen durchgeführt werden. Dadurch wird dass die Logik für die Repositoryanalyse nicht mit Unmengen an Traversierungscode verschleiert.

6.4.2 Automatische Analyse durch Gitinspector innerhalb des Programms

In Abschnitt 5.1.1 wurde bereits darauf hingewiesen, dass die Gitinspector XML Datei semiautomatisch mithilfe von Konsolenskripten erstellt werden muss. Die Gründe warum die diese Erstellung nicht innerhalb von UniMoDoc stattfindet werden im folgenden Aufgelistet.

Komplexe Installation: Als JSF Web Anwendung läuft UniMoDoc auf einem Server, der die vom Browser aufrufbaren Webseiten generiert und an diesen sendet. Für das UniMoDoc Projekt ist das der Apache Tomcat Server. Die zusätzliche Installation der in Gitinspector verwendeten Abhängigkeiten auf dem Server ist sehr komplex und war im Zeitrahmen dieser Bachelorarbeit nicht zu erledigen.

Lizenzkonflikte: Die in Gitinspector verwendete GNU GPL v3 Lizenz ist mit der in UniMoDoc verwendeten Apache Licence 2.0 kompatibel. Dies sorgt dafür, dass die modifizierte Version von Gitinspector separat von UniMoDoc angeboten werden muss.

Erhöhte Laufzeit der Synchronisation: Wenn die Repositoryanalyse direkt im Programm ausgeführt wird, würde sich die Ausführungzeit der Synchronisation um die Zeitdauer erhöhen, die Gitinspector benötigt um die XML Datei zu erstellen. Abschnitt 4.4.4 zeigte, dass dafür bei größeren Repositories Zeiten im Bereich von 10-15 Sekunden benötigt werden. Somit würde sich die Zeit während der Synchronisation, wo der Benutzer das Programm nicht bedienen kann noch weiter verlängern.

7 Zusammenfassung und Ausblick

Diese Bachelorarbeit zeigte nun, wie Entwickler bei der Aktualisierung ihrer Dokumentation von Computern unterstützt werden können. Dabei können Maschinen vor allem bei der Erhebung von Metriken helfen, die durch ihre Art von Menschen nur sehr langwierig und schwer zu ermitteln sind. Es wurde die Moduldokumentationssoftware UniMoDoc vorgestellt und erklärt wie das Programm mithilfe von Beschreibungselementen die Eigenschaften von Softwaremodulen dokumentiert. Anschließend anhand des Versionskontrollsystems Git erklärt, wie diese Systeme die, an sie übergebenen, Information abspeichern und es wurde ein Werkzeug ausgewählt, das diese Informationen aus Git extrahieren kann und damit ausgewählte Beschreibungselemente von UniMoDoc automatisch aktualisieren kann. Weiterhin wurde gezeigt, dass es zusätzlich noch möglich ist zu ermitteln, welche von den in UniMoDoc verwalteten Pakete im Repository noch exisieren und welche davon nicht mehr vorhanden oder neu hinzugekommen sind.

Im zweiten Teil der Bachelorarbeit wurde gezeigt, wie diese Veränderungen in UniMoDoc eingebaut wurden. Hierzu wurden die implementierten neuen Features mithilfe von Use-Case Diagrammen dargestellt. Weiterhin wurde erklärt wie die Implementierung umgesetzt wurde und gezeigt wie Probleme, die dort aufgetreten sind behandelt wurden.

7.1 Ausblick

Bei der Entwicklung der neuen Features in UniMoDoc sind folgende Ideen für Entwicklungen in der Zukunft aufgekommen:

- Die Möglichkeit alle Markierungen für im Repository nicht gefundenen und in der Dokumentation neu hinzugefügten Module auf einmal zu entfernen.
- Einen Absoluten Pfad in den Programmeinstellungen festzulegen, von dem aus die Beschreibungselemente *qName* und *src* angegeben werden.
- Für jede Datei im Repository bei der Synchronisation überprüfen, wann sie das letzte mal verändert wurde und diese Information im Beschreibungselement *Last Change* für die zuletzt veränderte Datei im Modul angeben.
- Weitere Beschreibungselemente, deren Informationen nicht im Repository liegen aber von einem Computer ermittelt werden können in UniMoDoc hinzufügen. Ein Beispiel dafür ist die Testabdeckung der Tests in den einzelnen Modulen.

Literaturverzeichnis

[Ato] Atom Editor - Project Website. URL https://atom.io/. (Zitiert auf den Seiten 22 und 25)

[Boo] Bootstrap - Project Website. URL http://getbootstrap.com/. (Zitiert auf Seite 25)

[Cha09] S. Chancon. *Pro Git.* Apress, 1. Auflage, 2009. (Zitiert auf den Seiten 7, 15, 16, 17, 18 und 19)

[Gita] Gitinspector - Project Website. URL https://code.google.com/p/gitinspector/. (Zitiert auf Seite 22)

[Gitb] GitStats - Project Website. URL https://sourceforge.net/projects/gitstats/. (Zitiert auf Seite 21)

[Jav] The Java EE5 Tutorial - Chapter 10 Java Server Faces Technology. URL http://docs.oracle.com/javaee/5/tutorial/doc/bnaph.html. (Zitiert auf Seite 37)

[Lud10] H. L. Ludewig, J. *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken.* dpunkt.verlag, 2. Auflage, 2010. (Zitiert auf Seite 9)

[Mar] M. P. K. S. Marinschek, Kurz. Kapitel 2.4 Managed Beans. URL http://jsfatwork.irian.at/. (Zitiert auf Seite 37)

[Nov] Novabench - Website. URL http://novabench.com/. (Zitiert auf Seite 25)

[Sta] StatSVN - Project Website. URL http://www.statsvn.org/. (Zitiert auf Seite 21)

Alle URLs wurden zuletzt am 29. 10. 2014 geprüft.

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