Bachelorarbeit

Extraktion von Daten aus Versionskontrollsystemen zur anschließenden Visualisierung in einer CodeCity

Felix Gaebler

30. Dezember 2021

Betreuung

Prof. Dr. Rainer Koschke

Dr. Rene Weller

Universität Bremen

Fachbereich 3 – Mathematik und Informatik

Bibliothekstraße 1

28359 Bremen

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

II Abkürzungsverzeichnis

CI Continuous Integration

CVCS Centralized Version Control System

DTD Document Type Definition

DVCS Distributed Version Control System

GXL Graph eXchange Language

LOC Lines of Code

RFG Resource Flow Graph

SEE Software Engineering Experience

SUS System Usability Scale

SVN Subversion

UI User Interface

VCS Version Control System

VR Virtual Reality

XML eXtensible Markup Language

1 Einleitung

"The number of transistors that can be placed on an integrated circuit doubles every two years."

— Gordon Moore, 1965

Gordon Moore sagt mit diesem Zitat vor 56 Jahren die Entwicklung der Prozessoren und damit auch die Entwicklung der Rechenleistung unserer Computer für die letzten Jahrzehnte vorher. Mit steigender Leistung der Computer kam auch die Fähigkeit zur Ausführung größerer und komplexerer Programme, welche unseren digitalisierten Alltag begleiten.

Unsere steigenden Ansprüche an den Funktionsumfang und die ständige Weiterentwicklung von Softwarepaketen sorgt dafür, dass Software über die Zeit immer größer wird. [1] Die „Größe" einer Software wird bemessen an den Lines of Code (LOC). Beispielhaft für diese Entwicklung ist der Linux-Kernel, wie in Abbildung 1 zu sehen.

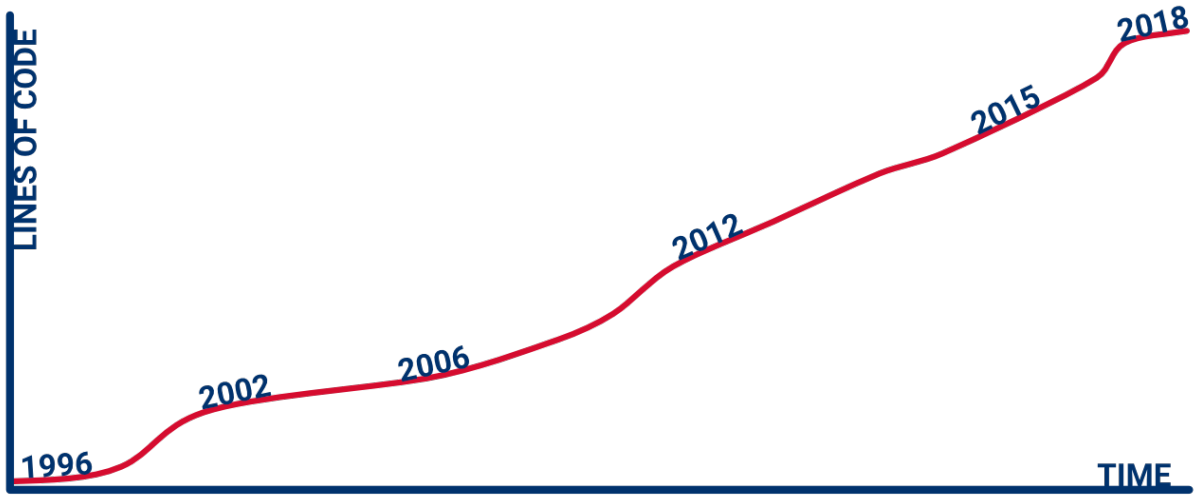


Abbildung 1: Entwicklung der LOC vom Linux Kernel 1991-2015 [2]

Mit der Anzahl der LOC steigt die Komplexität der Software [1] und damit auch der Wartungsaufwand. Das Einlesen in komplexe Programme ist zeitintensiv und schwierig.

Abhilfe bei diesem Problem soll ein Ansatz von Richard Wettel schaffen, bei welchem eine Software mithilfe einer Stadtmetapher visualisiert wird - sogenannte „CodeCities"[3].

Die Arbeitsgruppe Softwaretechnik der Universität Bremen setzt, angeleitet von Prof. Dr. Rainer Koschke, eine eigene Implementation einer CodeCity mit einer Virtual Reality (VR) Umgebung um. Das Projekt Software Engineering Experience (SEE) liest ein beliebiges Programm in Form eines Graphen ein und erstellt dazu eine Visualisierung. Die Graphen werden von dem Werkzeug „Axivion Bauhaus Suite" erzeugt [4].

Versionskontrollsysteme haben zunehmende Bedeutsamkeit bei der Entwicklung von Software. Entwicklungsabläufe werden technisch unterstützt und die Zusammenarbeit mehrerer Entwickler erleichtert [5]. Diese Bachelorarbeit soll einen Ansatz implementieren, der die Informationen eines Versionskontrollsystems in SEE visualisiert.

1.1 Zielsetzung

Ziel dieser Bachelorarbeit ist es, eine Versionshistorie mithilfe einer CodeCity zu visualisieren. Dazu wird ein Werkzeug programmiert, welches eine Versionshistorie durchläuft, analysiert und in einem auf Graphen ausgelegtes Dateiformat ausgibt. Diese Dateien werden in ein vorhandenes Programm eingelesen, welches daraus CodeCities erstellt und anzeigt. Dieses Programm soll zusätzlich um weitere sinnvolle Methoden zur Visualisierung erweitert werden. Die Versionshistorie soll schlussendlich wie ein Video abgespielt werden, sodass die Entwicklung eines Programms, mit den zusätzlichen Informationen aus dem Version Control System (VCS) animiert dargestellt wird. Anschließend soll über eine Evaluation ermittelt werden, wie das Programm herkömmlichen Werkzeugen gegenüber abschneidet und ob es Vorteile bietet.

1.2 Aufbau der Bachelorarbeit

Im folgenden Kapitel 2 werden Grundlagen zum Verständnis dieser Bachelorarbeit geschaffen. Die Funktionsweise eines VCS, ein Dateiformat zum Austausch von Graphen und die Umgebung, in der sich dieses Projekt befindet, werden vorgestellt. Darauf folgt das Kapitel 3, in welchem die Zielsetzung definiert wird, um daraus einen Programmentwurf zu entwickeln. Architektur- und Designentscheidungen werden hier auf Basis der Ausgangslage getroffen. In Kapitel 4 wird das Programm, wie im vorherigen Entwurf geplant, implementiert. Zusätzlich wird die Implementierung und eventuell entstandene Problematiken erläutert. Das Ergebnis der fertigen Implementierung wird anschließend in Kapitel 5 evaluiert. Schlussendlich wird in Kapitel 6 ein Fazit gezogen und ein Ausblick darauf gegeben, wie diese Bachelorarbeit fortgesetzt werden kann.

2 | Grundlagen

Im folgenden Kapitel werden die Grundlagen zum Verständnis dieser Bachelorarbeit geschaffen. Begonnen wird mit einem allgemeinen Verständnis darüber, was ein Versionskontrollsystem ist. Anschließend wird die Bibliothek vorgestellt, welche genutzt wird, um die Versionskontrollsysteme einzulesen. Daraufhin wird Bauhaus vorgestellt, ein Werkzeug, mit dem eine Software in Graphen umgewandelt werden kann. Um diese Graphen zu persistieren und zu transportieren wird das Dateiformat Graph eXchange Language (GXL) verwendet, welches in dem darauf folgenden Kapitel erläutert wird. Zuletzt wird der Ansatz einer Softwarevisualisierung als Stadtmetapher erläutert und genauer auf die vorhandene Implementierung der AG Softwaretechnik der Universität Bremen eingegangen.

2.1 Versionskontrollsysteme

Ein Versionskontrollsystem verfolgt Änderungen am Quelltext und speichert diese in einer speziellen Datenbank, dem Repository. Unterläuft Entwicklern ein Fehler, können sie einen Schritt zurückgehen und vorherige Änderungen am Quelltext nachvollziehen, um das Problem zu beheben. Arbeitsstände werden dabei regelmäßig über Commit's an das Repository übermittelt. Diese Übermittlung nennt sich Push. Aus einer Menge an Commit's in chronologischer Reihenfolge ergibt sich eine Versionshistorie. Commit's sind einzelne Elemente in der Historie, zu denen später zurückgesprungen werden kann. Ein Commit wird zusammen mit einer kurzen Beschreibung der Änderungen, dem Autor und einem Zeitstempel gepushed, sodass Entwickler mit einem Blick auf die Historie einen Commit identifizieren können, ohne sich die einzelnen Änderungen in den Dateien anzuschauen. Der Sprung auf einen anderen Commit in der Versionshistorie heißt Checkout.

Ein Projekt mit Versionskontrolle ist wie ein Baumstamm aufgebaut. Die Entwickler laden einen Klon des Stammes (oder des Hauptzweigs) aus dem Repository herunter und können dann einen Zweig erstellen, auf welchem sie experimentieren und arbeiten können, ohne den Hauptzweig zu beeinflussen. Die Zweige werden bei VCS Branches genannt.

Wenn mehrere Entwickler auf demselben Branch arbeiten kommt es zu Konflikten zwischen den beiden Änderungen innerhalb eines Branches. Um diese Konflikte zu vermeiden sollten mehrere Branches verwendet werden. Der Haupt-Branch wird master genannt. Wenn die Funktion erfolgreich implementiert wurde und der Branch wieder stabil ist, wird dieser in den master gemerged (Abbildung 2.D). Entwickler können jederzeit Änderungen, welche auf dem master entwickelt werden, auf ihren Branch mergen (Abbildung 2.C), um z. B. einen aktuelleren Stamm in ihrer Entwicklung zu erhalten.

Bei der Zusammenarbeit an einem Projekt werden dieselben Dateien, von verschiedenen Benutzern, in unterschiedlichen Zweigen geändert. Wenn diese Zweige zusammengeführt werden, können Konflikte auftreten, die verhindern, dass die Zusammenführung abgeschlossen wird. Die Zusammenführung selber wird Merge genannt. Beim mergen von Zweigen mehrerer Benutzer gibt das VCS eine Fehlermeldung aus und markiert den Abschnitt der Datei, der den Konflikt enthält. Dieser Fall nennt sich Merge-Conflict. Dieser Konflikt kann von einem Entwickler behoben werden, indem die Änderungen manuell zusammengeführt und anschließend commited und gepushed werden.

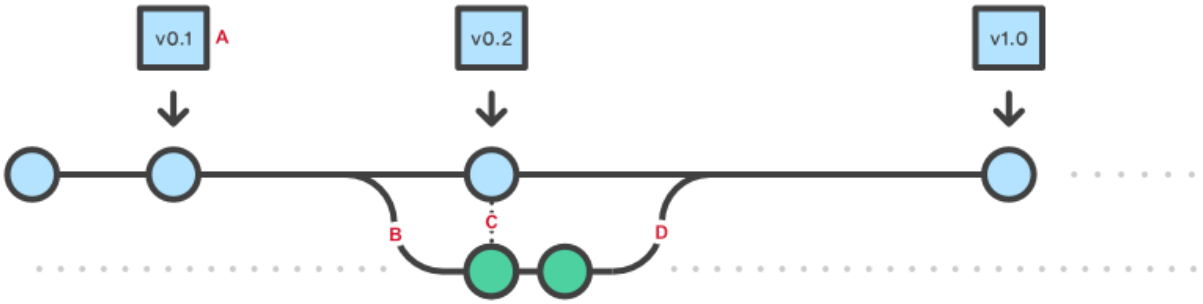


Abbildung 2: Visualisierung der Historie eines Versionskontrollsystems

Um Besonderheiten in der Zeitachse, wie z. B. die Veröffentlichung einer Version, hervorzuheben, können Tags (Abbildung 2.A) verwendet werden. Mit diesen ist es möglich bestimmte Commit's zu markieren.

Es gibt verschiedene Implementierungen von Versionskontrollsystemen, wie z. B. Git, Mercurial oder Subversion (häufig abgekürzt mit SVN), welche sich im oben vorgestellten Grundkonzept von Versionskontrollsystemen nicht unterscheiden. Bei den Implementierungen wird zwischen Centralized Version Control Systems (CVCS) und Distributed Version Control Systems (DVCS) unterschieden. CVCS speichern das Repository ausschließlich auf einem zentralen Server, während bei DVCS jeder Entwickler lokal eine Repository hat, welche z. B. zur Zusammenarbeit optional mit einer zentralen Einheit synchronisiert werden kann. Dieser Aufbau gibt den Entwicklern die Möglichkeit auch ohne Zugang zum Internet zu arbeiten [6].

OpenHub (https://www.openhub.net/) ist eine Datenbank für Open-Source Projekte, welcher z. B. entnommen werden kann, welches VCS in Projekten eingesetzt wird. Die Daten sind exemplarisch für die Vielfalt an Projekten in der gesamten Branche. Die Abbildung 3 zeigt, dass Git und Subversion am weitesten verbreitet sind. Git vertritt die DVCS als am weitesten verbreitete Implementierung von VCS in der IT-Branche.

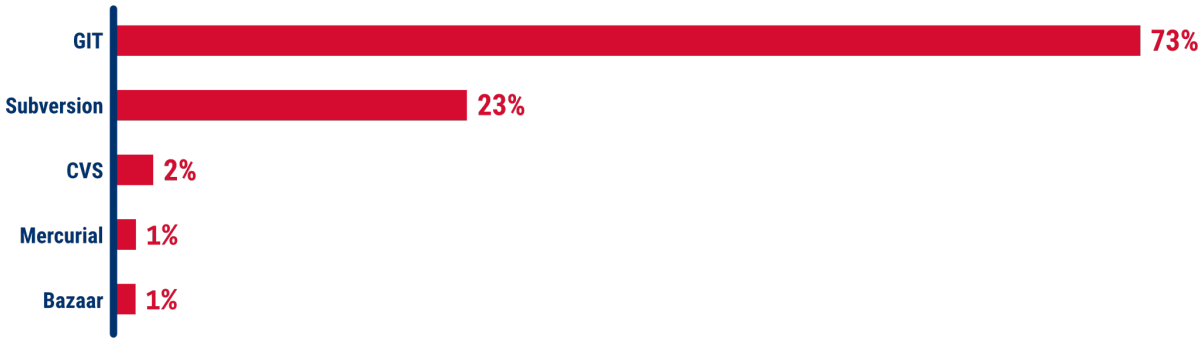


Abbildung 3: Marktanteil von Versionskontrollsoftware bei OpenSource Projekten

2.2 LibVCS4j

Um die Informationen, welche in einem VCS enthalten sind in einem Programm zu verarbeiten, wird die Java-Bibliothek LibVCS4j von Marcel Steinbeck verwendet. Die Bibliothek unterstützt bisher die Versionskontrollsysteme Git, Mercurial und Subversion [7] und ist in dieser Hinsicht der Engpass für die Unterstützung von Versionskontrollsystemen in dieser Bachelorarbeit. Wie Abbildung 3 zu entnehmen, machen diese drei VCS 97% des Marktanteils aus. Die verbleibenden 3% sind zu vernachlässigen.

Das Ablaufdiagramm in Abbildung 4 veröffentlicht Marcel Steinbeck zusammen mit seinem Paper „Mining version control systems and issue trackers with LibVCS4j“ [5]. Es zeigt die Verwendung von LibVCS4j in einem Programm. Die Knoten, welche die `ITEngine` behandeln sind dabei für diese Bachelorarbeit zu vernachlässigen, da keine Issues aus den VCS extrahiert werden. Begonnen wird mit der Initialisierung der `VCSEngine`. Die Initialiserung benötigt einen Pfad zur Repository. Je nach verwendetem VCS wird der Pfad mit einer anderen Methode gesetzt. Im folgenden Beispiel wird die Methode `ofGit()` für ein Git VCS verwendet. Zusätzlich wird der Branch abgegeben, der verarbeitet werden soll. In dieser Bachelorarbeit wird nur der `master`-Branch betrachet, deshalb wird dieser explizit angegeben. Der Quelltext zur Initialisierung sieht dann wie folgt aus:

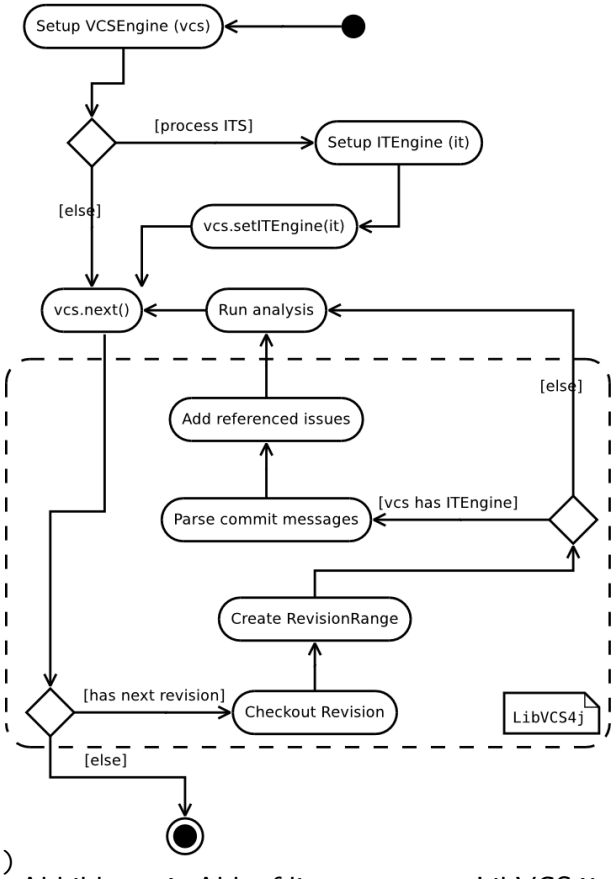


Abbildung 4: Ablaufdiagramm von LibVCS4j

```
VCSEngine vcs = VCSEngineBuilder
    .ofGit("file://<REPOSITORY_PATH>")
    .withBranch("master")
    .build();
```

Die `VCSEngine` erbt von einem Iterator, kann mit der Methode `next()` iteriert werden. Bei dem Aufruf der ersten Iteration legt die Bibliothek eine Kopie von der Repository in einem temporären Verzeichnis an und führt ein checkout auf den ersten Commit aus. Der Pfad zum temporären Verzeichnis lässt sich mit der Methode `VCSEngine.getOutput()` zurückgeben.

Iteratoren können in Java in einer `for`-Schleife durchlaufen werden. Das Objekt, welches als Rückgabe aus der Iteration folgt, ist eine `RevisionRange`. Diese Klasse repräsentiert einen Zustandsübergang von einer Revision zu einer anderen. Damit kann die `RevisionRange` mehrere Commit's enthalten. Die Implementierung von LibVCS4j bietet damit die Möglichkeit, mehrere Commit's für z. B. einen Merge oder eine monatliche Zusammenfassung zusammenzuführen.

Mit jedem Aufruf der `next()` Methode des Iterators wird auf die Kopie ein Checkout ausgeführt und der nächste Commit geladen. Der Quelltext im temporären Verzeichnis ist somit immer auf dem Stand des aktuellen Commit's. Dadurch kann z. B. auf das Verzeichnis eine Analyse durch ein externes Programm ausgeführt werden.

Es ergibt sich hier eine Hierarchie. Die `VCSEngine` enthält mehrere `RevisionRange`'s, diese

wiederum enthalten mehrere `Commit`'s und diese enthalten mehrere `FileChange`'s.

Ein `FileChange` repräsentiert Änderungen an einer Datei. Es gibt verschiedene Typen: `ADD` für eine hinzugefügte Datei, `REMOVE` für eine gelöschte Datei, `RELOCATE`, wenn eine Datei verschoben wurde und `MODIFY` für eine bearbeite Datei. Dazu bringt die Klasse noch zwei Methoden mit sich, um die zugehörige Datei zurückzugeben. `getOldFile()` und `getNewFile()`. Beide Methoden geben ein `Optional<VCSFile>` zurück. Bei einem `RELOCATE` lassen sich somit die aktuelle und die alte Datei ermitteln. Eine mögliche Implementierung, um jeweils die neue oder die alte Datei zu erhalten, sehe wie folgt aus:

```
VCSFile file = fileChange.getNewFile()
    .orElse(fileChange.getOldFile().orElse(null));
```

Ein weiteres Element in der Hierarchie sind die `LineChange`'s. Von dieser Klasse wird die Änderung auf einer einzelnen Zeile repräsentiert. Mit der Methode `fileChange.computeDiff()` lassen sich die `LineChange`'s berechnen.

```
List<LineChange> lineChanges = fileChange.computeDiff();
```

Ein `LineChange` hat die Typen `INSERT` oder `DELETE`. Eine veränderte Zeile wird mit zwei `LineChange`'s repräsentiert, indem die Zeile erst entfernt und dann als neue hinzugefügt wird.

2.3 Bauhaus

Bauhaus wird 1996 als Forschungsprojekt von Erhard Ploedereder und Rainer Koschke an der Universität Stuttgart ins Leben gerufen. Kurze Zeit später entsteht durch den Wechsel von Rainer Koschke an die Universität Bremen eine Kooperation zwischen beiden Universitäten. Ziel ist seitdem das Verstehen von Software und Reverse Engineering, um die Überholung und Wartung von Softwaresystemen zu unterstützen. [8]

Um der wachsenden Resonanz der Industrie gerecht zu werden, wurde 2006 ein kommerzieller Zweig des Forschungsprojekts gegründet, welcher heute als das Unternehmen Axivion bekannt ist. Axivion unterhält seit seiner Gründung enge Forschungskooperationen mit den Universitäten Stuttgart und Bremen für die Weiterentwicklung seiner Konzepte und Werkzeuge. Vertrieben wird das Produkt inzwischen unter dem Namen „Axivion Suite“. [9]

Die Axivion Suite unterstützt sämtliche Funktionen wie fortgeschrittene Code- und Datenflussanalysen, Zeigeranalysen, Seiteneffektanalysen, Programm-Slicing, Klon-Erkennung, Quellcode-Metriken, statisches Tracing, Abfragetechniken, Quellcode-Navigation und -Visualisierung, Objektwiederherstellung, Remodularisierung und Architekturwiederherstellungstechniken. [10]

2.3.1 Resource Flow Graph

Ein Resource Flow Graph (RFG) ist ein hierarchischer Graph, der aus typisierten Knoten und Kanten besteht. Knoten repräsentieren Bestandteile eines Programmes, z. B. Pakete, Klassen und Methoden. Die Beziehungen zwischen diesen Elementen werden durch Kanten abgebildet. [10] Komponenten der Axivion Suite generieren aus ihren Analysen `.rfg` Dateien. Das Dateiformat ist binär und für die interne Verwendung von der Axivion Suite vorgesehen. Zum Austausch der Graphen mit anderen Programmen, verfügt die Axivion Suite über ein Werkzeug zur Umwandlung in GXL Dateien.

2.3.2 Klon-Erkennung

Bei der Entwicklung von Programmen ist es gängig, Quelltext zu kopieren und an anderen Stellen des Programms einzufügen. Der Quelltext wird dabei teilweise verändert. Wenn ein Fehler in einer der Kopien gefunden und verbessert wird, sollten alle anderen Kopien ebenfalls korrigiert werden. Im Quelltext besteht allerdings keine Referenz zu den anderen Kopien. Infolgedessen wird derselbe Fehler erneut aufgedeckt, neu analysiert und auf unterschiedliche Weise in jedem Klon behoben.

Die Klon-Erkennung ist ein Verfahren, mit welchem die Kopien aufgedeckt werden. Die Entwickler haben so die Möglichkeit, die Klone von einem zu verbessernden Quelltext aufzuspüren und auf die gleiche Weise zu beheben. Dieses Vorgehen steigert die Qualität des Quelltextes und verhindert Fehler in nicht gefundenen Klonen. [10]

2.4 Graph eXchange Language

Die GXL ist ein Standardformat zum Austausch von Graphen, welches auf der eXtensible Markup Language (XML) basiert. Das Ziel von GXL ist es, ein flexible Datenmodell zur Interoperabilität zwischen Software und Werkzeugen, welche mit Graphen arbeiten, zur Verfügung zu stellen. GXL ist ein allgemeines Graphenaustauschformat und kann damit auch für den Austausch beliebiger graphenbasierter Daten verwendet werden. Formal stellt GXL typisierte, attributierte, gerichtete, geordnete Graphen dar, die zur Darstellung von Hypergraphen und hierarchischen Graphen erweitert werden. Das Format kann auch um andere Arten von Graphen erweitert werden. [11]

2.4.1 Beispiel

```
<!DOCTYPE gxl SYSTEM "http://www.gupro.de/GXL/gxl-1.0.dtd">
<gxl xmlns:xlink="http://www.w3.org/1999/xlink">
    <graph id="example" edgeids="true">
        <node id="N1">
            <type xlink:href="File"/>
            <attr name="Linkage.Name">
                <string>example\folder\Example-A.txt</string>
            </attr>
        </node>
        <node id="N2">
            <type xlink:href="File"/>
            <attr name="Linkage.Name">
                <string>example\folder\Example-B.txt</string>
            </attr>
        </node>
        <node id="N3">
            <type xlink:href="File"/>
            <attr name="Linkage.Name">
                <string>example\folder\Example-B.txt</string>
            </attr>
        </node>
        <edge id="E1" from="N1" to="N2">
            <type xlink:href="Enclosing"/>
        </edge>
        <edge id="E2" from="N1" to="N3">
            <type xlink:href="Clone"/>
        </edge>
    </graph>
</gxl>
```

Abbildung 5: Visualisierung des Beispiels

2.4.2 Aufbau

Die Struktur eines XML Dokuments wird über eine XML Document Type Definition (DTD) festgelegt. Der Pfad zur DTD wird zu Beginn eines Dokuments angegeben. Die DTD für GXL Dateien ist, wie in dem Beispiel zu erkennen, online unter `http://www.gupro.de/GXL/gxl-1.0.dtd` verfügbar.

Der Aufbau eines XML Dokuments ist hierarchisch. Die hierarchische Ebene ist äquivalent zu der Einrückungstiefe im Dokument. Auf der obersten Ebene gibt es nur einen Knoten. Dieser Knoten ist in GXL Dokumenten der `<gxl>` Tag. Die DTD von GXL sieht es vor, dass die Knoten innerhalb eines `<gxl>` Tags Graphen, sprich `<graph>` Tags sind. Jeder Graph wird mit einer ID versehen, welche direkt als Attribut im Tag mitbestimmt wird.

Graphen können verschiedene Tags enthalten: Attribute (`<attr>`), Knoten (`<node>`) und Kanten (`<edge>`). Knoten und Kanten haben Besonderheit, dass sie zusätzlich einen frei definierbaren Typen benötigen. Kantenelemente können einen Hinweis zur Hierarchie liefern, wie im Beispiel und Abbildung 5 zu sehen.

2.4.3 Java Bibliothek

Auf der offiziellen Website von GXL wird unter den Werkzeugen die GXL-Java-API zum Herunterladen bereitgestellt. Die GXL-Java-API ist eine Java Bibliothek zum Lesen und Bearbeiten von Graphen im GXL-Format. Die Klassenhierarchie ist an die Struktur eines GXL-Dokuments angelehnt, sodass die Verwendung der Bibliothek keine großen Zusatzkenntnisse erfordert. Die Klassennamen entsprechen dabei den Namen der Tags aus dem GXL-Dokument mit einem „GXL“ davor (Beispiel: `GXLNode`). [12]

2.5 Softwarevisualisierung

Die Größe und Komplexität von Programmen steigt mit den Anforderungen der Digitalisierung. Die Entwicklung der Programme erfordert ein gewisses Verständnis über die Funktionsweise und den Aufbau. Schon Aristoteles sagt 350 v. Chr., dass „Denken ohne Bilder unmöglich sei“. Programme bestehen aus Text und logischen Zusammenhängen, die für einen Menschen, ohne visuelle Unterstützung und entsprechende Vorstellungen, schwierig zu verstehen sind.

Durch eine Visualisierung sollen Entwickler Programme schneller und besser verstehen. Das Ziel soll dabei nicht sein, eine schöne Grafik zu erzeugen, sondern mentale Bilder bei den Entwicklern auslösen und so das Verstehen von Software erleichtern. [13]

2.5.1 Code Cities

In der Informatik ist es üblich in Metaphern zu sprechen und zu denken. Beispiele dafür sind die Begriffe Automaten, Knoten, Kanten, Ordner oder Archive. Eine „Code City“ ist ein Visualisierungsansatz in Form einer Stadtmetapher. Die Idee dieser Visualsisierung stammt von Richard Wettel und Michele Lanza.

In der Implementierung von Wettel repräsentiert ein Gebäude eine Klasse. Die Höhe ist proportional zur Anzahl der Methoden. Länge und Breite der Gebäude sind proportional zur Anzahl der Attribute. Klassen mit besonders vielen Attributen, sogenannte Datenklassen, sind somit flache, große Gebäude, welcher in der Metapher als Lagerhallen zu verstehen sind. Hohe, weniger flächige Gebäude, sind Klassen in welchen viele, unterschiedliche Verarbeitungen stattfinden. In der Metapher sind diese Gebäude als Bürogebäude zu verstehen. [14]

In Abbildung 6 ist eine CodeCity von Wettel zu sehen.

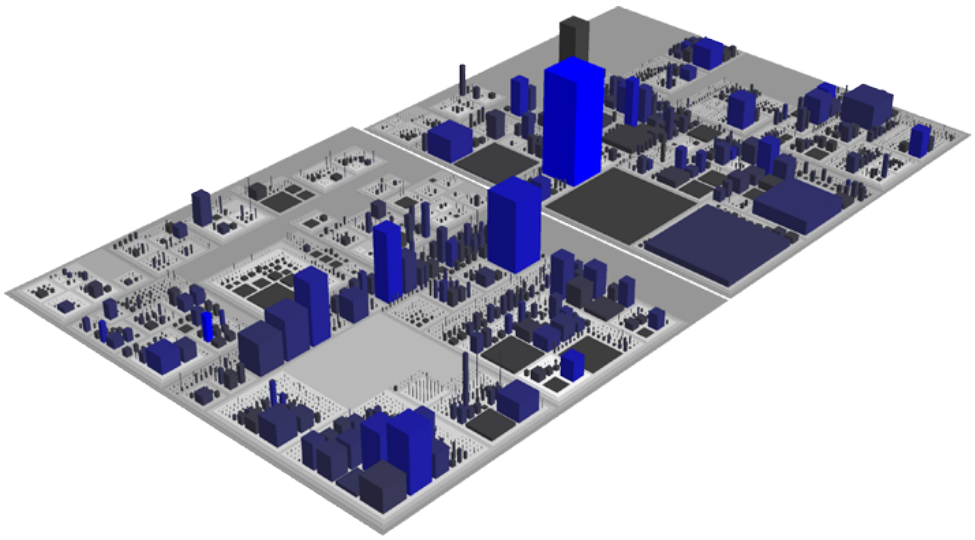


Abbildung 6: CodeCity von Wettel

Die Anordnung der Gebäude erfolgt mit Hilfe einer TreeMap auf einer Fläche, deren Höhe die Verschachtelungsebene der Pakete anzeigt. Es entstehen Stadtteile.

Eine Stadt mit ihren Stadtteilen, Funktionen und Aufbau ist den meisten Menschen ein vertrautes Bild und bietet damit ein klares Orientierungskonzept. [14]

2.6 Software Engineering Experience

Software Engineering Experience (SEE) ist eine Implementation einer CodeCity als Forschungsprojekt von Rainer Koschke an der Universität Bremen. Dem Projekt liegt Bauhaus, das in Abschnitt 2.3 vorgestellte Werkzeug zur Analyse von Programmen, zugrunde. Umgesetzt wurde es in der populären Spiele-Engine Unity, auf welche in Abschnitt 2.6.1 genauer eingegangen wird. Durch die Verwendung von Unity konnten Funktionen, welche in Computerspielen üblich sind, vereinfacht implementiert werden.

Somit bringt SEE z. B. einen „Mehrspieler-Modus“ mit, der mehreren Entwicklern eine Verbindung per Fernzugriff ermöglicht, um von jedem Ort der Welt aus miteinander zu kommunizieren.

Durch die Verwendung von Bauhaus gibt es diverse Daten aus Analysen, welche in SEE visualisiert werden können. Auch das Debuggen von Programmen und prüfen von Architekturen kann mit SEE bewältigt werden.

Während der Ausführung verfügt das Programm noch nicht über ein vollumfängliches User Interface (UI). Es besteht die Möglichkeit diverse Einstellungen vor der Ausführung in Unity zu treffen. Ein Beispiel eines solchen Menüs ist in Abbildung 15 zu sehen.

Da SEE diverse Visualisierungen bietet, gibt es verschiedene Stadttypen, welche zur Visualisierung verwendet werden können. Für diese Bachelorarbeit wird der Typ „SEECity Evolution“ verwendet und genauer betrachtet. Mehr dazu in Abschnitt 2.6.2.

2.6.1 Unity Spiel-Engine

Generell ist eine Spiel-Engine eine Sammlung von Bibliotheken, Komponenten und Werkzeugen, welche speziell auf die Erstellung von Computerspielen ausgelegt sind. Jede Spiel-Engine bringt ihre eigenen Stärken mit. Eine Spiel-Engine soll die Entwicklung eines Computerspiels stark vereinfachen, indem sie vorgefertigte Funktionen für schwer oder aufwendig zu implementierende Funktionen mitbringt. Folgende drei Komponenten sind typisch für eine Spiel-Engine:

1. **Physik-Engine**: Bietet Funktionen zur Simulation von Physik. So lassen sich in Spielen Gravitation, Bewegungsabläufe und Kollisionen zwischen Objekten auf Grundlage physikalischer Modelle darstellen.
2. **Grafik-Engine**: Verwaltet das laden und anzeigen von Grafiken. Oft stellt die Engine vorgefertigte Komponenten für z. B. Menüs zur Verfügung. Rendering-Prozesse können ebenfalls optimiert werden.
3. **Netzwerk-Engine**: Bringt vorgefertigte Funktionen für eine Netzwerkverbindung zu einem Server oder zwischen Spielern.

In dem Projekt SEE wird die Spiel-Engine Unity verwendet. Unity bringt keine eigene Entwicklungsumgebung mit sich, aber eine nahtlose Einbindung für den Quelltext-Editor „Visual Studio“. Mit Unity können Spiele für alle gängigen Betriebssysteme, sowie für Plattformen wie WebGL, Android, iOS oder Playstation gebaut werden.

Die grafische Oberfläche von Unity wird in Abbildung 7 gezeigt.

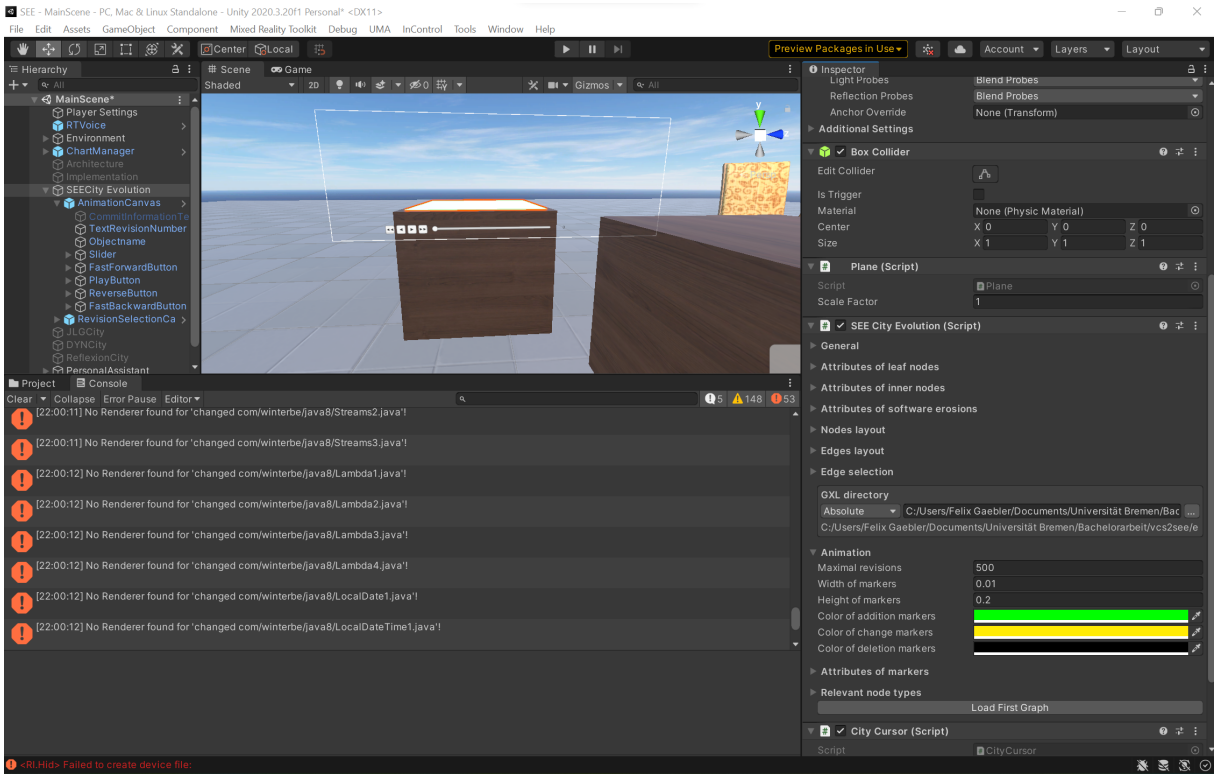


Abbildung 7: Screenshot von Unity

Die Umgebung, in der sich das Spiel befindet, nennt sich Szene und kann in Unity direkt modelliert werden. Das Modellieren wird über das Fenster in der Mitte vorgenommen. Links an der Seite sind die verschiedenen Objekte der Szene zu finden. Die Stadttypen von SEE sind einzelne Objekte, die aktiviert oder deaktiviert werden können. Einem Objekt kann mit Skripten, welche in C# programmiert werden, eine Logik verliehen werden. Dies geschieht außerhalb von der Unity Oberfläche, in Visual Studio. An der rechten Seite des Screenshots ist der Inspector als Menü zur Konfiguration von SEE zu finden.

2.6.2 SEECity Evolution

Die `SEECity Evolution` ist ein Stadttyp von SEE. Im Gegensatz zu den anderen Stadttypen lädt die `SEECity Evolution` nicht nur einen Graphen, sondern einen ganzen Ordner. Die Graphen werden wie ein Video hintereinander abgespielt. Die Übergänge von einem Graphen in den nächsten werden ebenfalls von diesem Stadttypen animiert. In Abbildung 8 ist eine `SEECity Evolution` vor der Implementierung dieser Bachelorarbeit zu sehen.

Eine weitere Besonderheit des Stadttypen ist die Verwendung der Marker. Marker sind Strahlen, welche sich auf Knoten befinden können. Die Farbe der Strahlen kann über ein Menü im Unity Inspector angepasst werden. Es gibt verschiedene Marker und dementsprechend Farben für hinzugefügte, bearbeitete und gelöschte Dateien.

3 Entwurf

Im Rahmen dieser Bachelorarbeit wird ein Programm und einige Komponenten zur Visualisierung in SEE entwickelt. Das Programm wird automatisiert und chronologisch die Revisionen eines VCS auschecken und mit Bauhaus analysieren. Das Ergebnis ist ein Verzeichnis mit GXL Dateien - das Eingabeformat für eine `SEECity Evolution`. Nach der Erstellung wird jede einzelne GXL Datei zusätzlich mit Informationen aus dem VCS erweitert.

Wie in Abschnitt 2.6.2 beschrieben, verfügt SEE bereits über einen Stadttypen, welcher eine Serie an Graphen wie ein Video abspielen kann. Änderungen werden von SEE erkannt und die Knoten werden dementsprechend mit einem farbigen Marker versehen. Die Farbe des Markers wird darüber bestimmt, ob der Knoten hinzugefügt, verändert oder gelöscht wird. Unveränderte Knoten haben keinen Marker. Auf Abbildung 8 ist die erste Revision einer Versionshistorie zu sehen, deshalb hat jeder Knoten einen „hinzugefügt“-Marker.

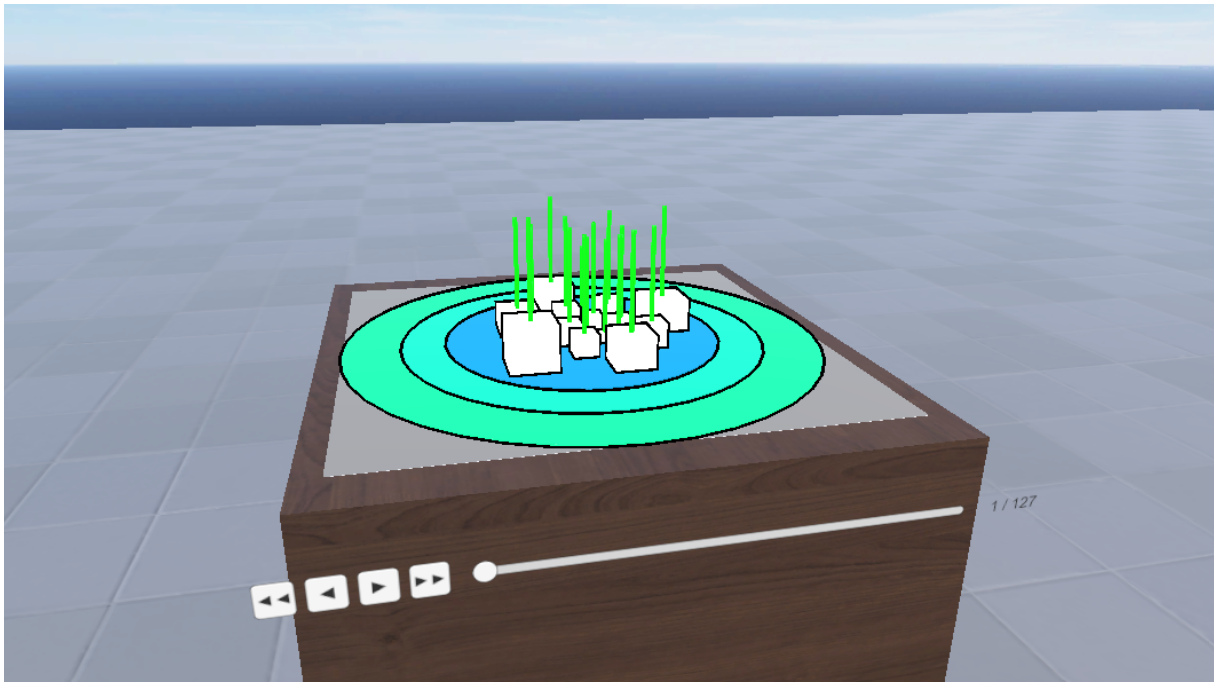


Abbildung 8: Zustand von SEE vor der Bachelorarbeit

Bisher existiert keine automatisierte Erstellung von GXL Dateien für eine `SEECity Evolution`. Um Beispiele zu generieren wurden vereinzelt Skripte erstellt oder Revision für Revision manuell analysiert.

3.1 Anforderungen

Zukünftig soll es möglich sein, ein beliebiges VCS mit LibVCS4j einzulesen und die Revisionen chronologisch zu einem GXL-Verzeichnis zu verarbeiten. Informationen zu dem VCS sollen dabei gesammelt und aggregiert werden, damit diese Informationen zu den GXL Dateien hinzugefügt werden können. Dieser Teil wird aufgrund der exklusiven Verfügbarkeit der Bibliothek für Java in einem Java-Programm und nicht direkt in SEE mit C# umgesetzt.

Das Verzeichnis soll sich in SEE einlesen lassen und in einer SEECity Evolution visualisiert werden. Es sollen weitere sinnvolle Komponenten zur Visualisierung der neuen Informationen hinzugefügt werden. Dieser Teil wird in C# zu dem vorhandenen Quelltext von SEE hinzugefügt.

3.1.1 Vcs2See (Java)

Um sich ein Überblick über den Funktionsumfang zu verschaffen, wurde ein Ablaufdiagramm (Abbildung 9) erstellt.

Das Programm beginnt damit die Konfigurationsdatei auszulesen. Die Struktur Datei wird wie folgt aussehen:

```
environment.bauhaus=#Pfad zu den Bauhaus Binaries
environment.cpfcsv2rfg=#Pfad zu der cpfcsv2rfg Binary

repository.name=#Name der Repository
repository.path=#Pfad zur Repository
repository.type=#Typ der Repository (GIT, HG, SVN)
repository.language=#Programmiersprache (C, CPP, CS, JAVA, ADA)

project.base=#Base-Path (z.B. maven src/main/java)

analyser.before.command=#Befehl zur Vorbereitung
analyser.before.directory=#Ort an dem der Befehl ausgefhrt werden soll

analyser.0.command=#Erster Befehl der in jedem Zyklus ausgefhrt wird
analyser.0.directory=#Pfad an dem der Befehl ausgefhrt werden soll
analyser.1.command=#Zweiter Befehl der in jedem Zyklus ausgefhrt wird
analyser.1.directory=#Pfad an dem der Befehl ausgefhrt werden soll
#... beliebig viele weitere Befehle

analyser.after.command=#Befehl zur Nachbereitung
analyser.after.directory=#Ort an dem der Befehl ausgefhrt werden soll

modifier.path=#Pfad zur GXL Datei nach der Analyse
# z.B. %repository.temp%/%repository.name%/%filename%.gxl
```

Anschließend wird geprüft, ob der Startparameter -Dci=true gesetzt ist.

Dieser kann verwendet werden, um die Einrichtung zu überspringen. Wenn der Parameter nicht gesetzt ist, wird die *Einrichtung durchlaufen*. Hierbei wird jeder Wert der Konfigurationsdatei angezeigt und die Möglichkeit geboten, diesen durch eine Eingabe zu ändern.

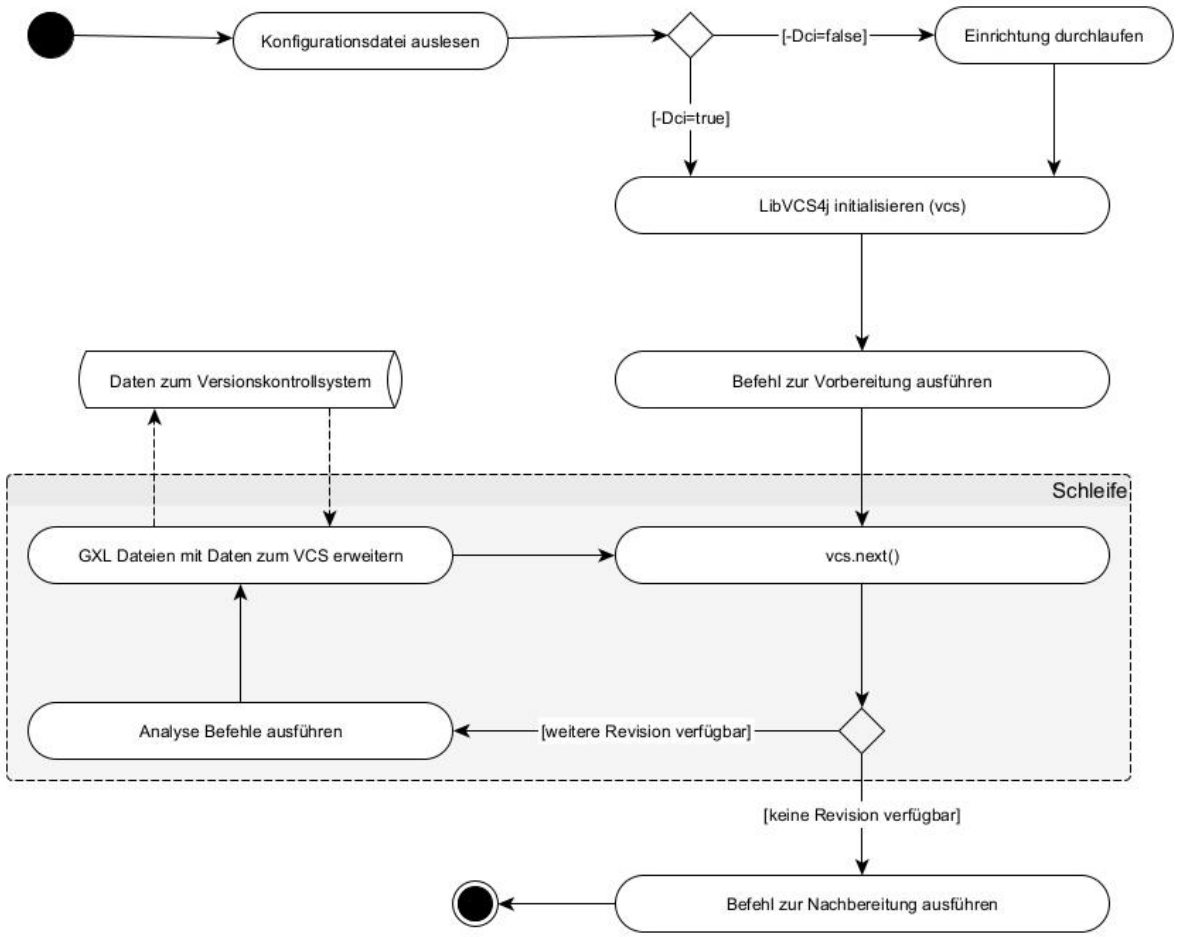


Abbildung 9: Ablaufdiagramm VCS2SEE

Wenn die Einrichtung abgeschlossen oder übersprungen ist, wird *LibVCS4j initialisiert*. Für die Initialisierung wird der Typ des VCS und der Pfad zum Repository benötigt. Diese Werte sind nach dem vorherigen Schritt der Konfigurationsdatei zu entnehmen.

Bevor die Analyse beginnt, wird ein *Befehl zur Vorbereitung ausgeführt*, welcher in der Konfigurationsdatei festgelegt werden kann (`analyser.before.command`). Hierbei können z. B. Verzeichnisse angelegt und bereits vorhandene Dateien entfernt werden, um eine saubere Umgebung zu schaffen.

Die Analyse findet nun zyklisch statt. Zuerst wird geprüft, ob eine weitere Revision in dem VCS vorhanden ist. Wenn das der Fall ist, werden alle *Befehle zur Analyse ausgeführt*. Die Befehle sollen eine GXL Datei erstellen, welche unter einem vorkonfigurierten Pfad abgelegt wird (`modifier.path`).

Diese GXL Datei wird geladen und mit den Informationen aus dem VCS erweitert. Die Daten aus dem VCS werden in eine Datenstruktur eingepflegt und Iteration für Iteration erweitert, sodass die Daten der nachfolgenden Revision von den vorherigen abhängig sind.

Sollte im nächsten Zyklus keine Revision verfügbar sein, wird ein *Befehl zur Nachbereitung* ausgeführt und anschließend das Programm beendet. In der Nachbereitung werden z. B. überflüssige Dateien gelöscht und die generierten GXL Dateien aus dem temporären Verzeichnis in das Verzeichnis des Programms kopiert.

Zur Umsetzung in Java werden die einzelnen Aufgaben in Klassen gegliedert. Die Hauptklasse ist `Vcs2See`. Die Klassen `PropertiesManager` und `ConsoleManager` sind Hilfsklassen und die restlichen Klassen implementieren die Funktionen, welche von der Hautklasse aufgerufen werden.

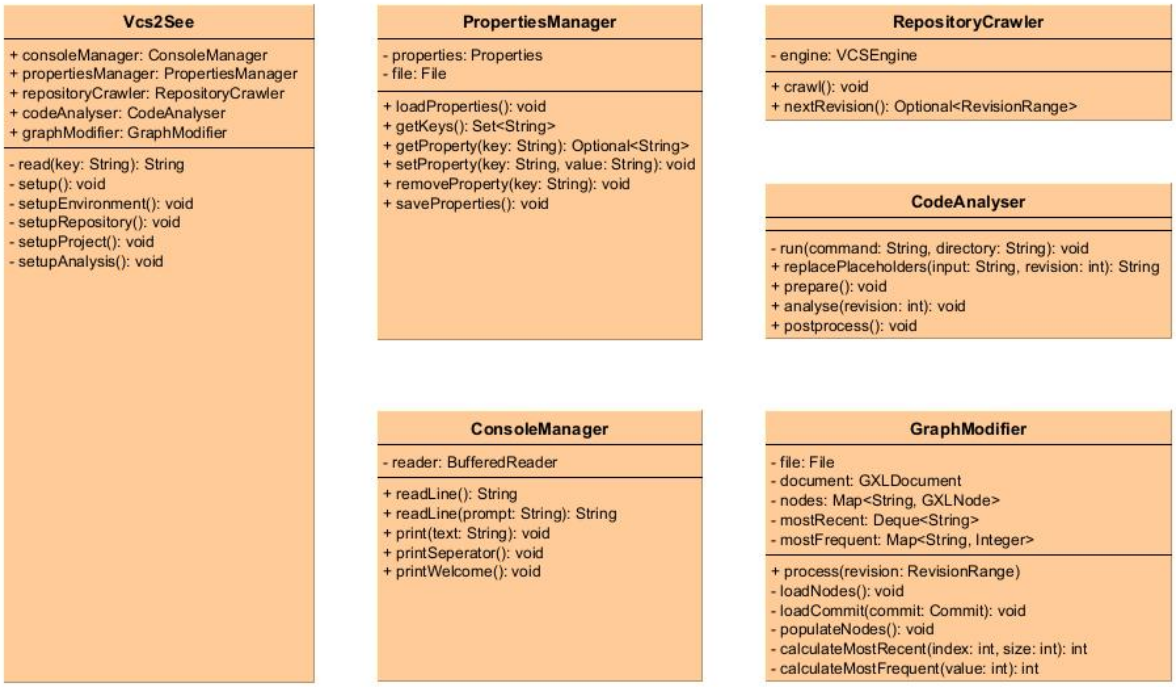


Abbildung 10: Klassendiagramm VCS2SEE

Bei der Planung des Programms müssen einige Entscheidungen zur Implementierung abgewogen werden, welche im Folgenden als „Probleme“ bezeichnet werden.

Problem 1: Branches in dem VCS

Ein VCS wird häufig mit mehreren Branches betrieben. Um alle Änderungen zu visualisieren müsste man einen synthetischen Branch erstellen, welcher alle Commit's enthält wie in Abbildung 11 zu sehen.

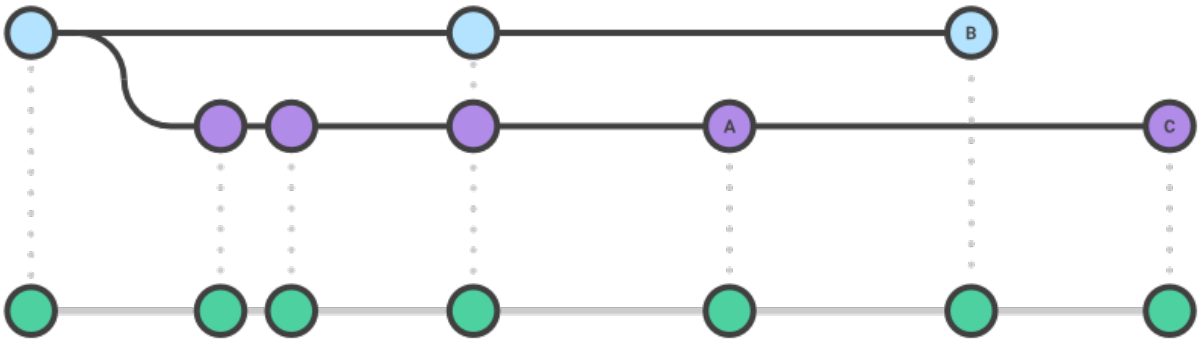


Abbildung 11: Visualisierung synthetischer Branch

Das Problem mit dieser Lösung ergibt sich z. B. in den Commit's `A`, `B` und `C`. Wenn in `A` eine Datei hinzugefügt wird, dann ist sie in `B` nicht vorhanden, aber wieder in `C`. In der Visualisierung führt das zu Verwirrungen.

Diese Komplikation wird durch eine Einschränkung behoben. In dieser Bachelorarbeit wird immer nur ein Branch betrachtet.

Problem 2: Daten oder Indizes?

Die „SEECity Evolution“ hat bereits einen Fortschrittsbalken, welcher sich nach der Anzahl der Dateien im angegebenen Verzeichnis richtet. Es gibt zwei Möglichkeiten diesen Balken zu nutzen:

a) Mit **Daten**. Zunächst wird das älteste und das neuste Datum ermittelt und dann die Tage dazwischen berechnet. Jeder Schritt auf dem Balken entspricht einen Tag. So ist es möglich mehrere Commit's in einem Schritt oder mehrere Schritte hintereinander keine Änderungen zu haben.

b) Mit **Indizes**. Jeder Schritt auf dem Balken entspricht hier einem Commit.

Die Wahl viel hier auf `b)`, da es teilweise große Zeiträume zwischen Commit's gibt und die Implementierung von LibVCS4j zwei Iterationen benötigen würde, da sich der letzte Commit nur mit vollständigen Durchlaufen des Iterators finden lässt.

3.1.2 Bauhaus

Die Java Komponente wird mit einem funktionierenden Beispiel ausgeliefert. Dazu werden im folgenden Abschnitt die nötigen Bauhaus Befehle erläutert und umgeformt, sodass sie mit Vcs2See zu verwenden sind. Ziel ist es, auf ein Verzeichnis mit Quelltext eine Klon-Erkennung auszuführen, welche in Abschnitt 2.3.2 genauer erläutert ist. Die Befehle zum Ausführen einer Klon-Erkennung stammen aus dem nicht öffentlichen Wiki des Repository von SEE. [4]

Zur Klon-Erkennung wird der Befehl `cpf` (Copy Paste Finder) genutzt.

```
cpf -m 100 -c clones.cpf -s clones.csv -a -i "*.java".
```

Die verwendeten Argumente haben folgende Bedeutungen:

- `-m` minimale Länge eines Klons angeben, die gemeldet werden soll (in der Anzahl der Token).
- `-c` Ausgabedatei, in der die Klonpaare gespeichert werden (im cpf-Format).
- `-s` Ausgabedatei, in der die Klonstatistiken gespeichert werden (im CSV-Format).
- `-a` fordert cpf auf, absolute Pfade zu verwenden; dies ermöglicht SEE, den Quellcode zu finden, wenn ein Quelltextbetrachter geöffnet wird.
- `-i` gibt ein Dateiglobbing für die zu berücksichtigenden Dateien an (mehrere Optionen dieser Art sind möglich).

Die entstandenen `.csv` und `.cpf` Dateien werden anschließend in eine `.rfg` Datei umgewandelt. Das `rfg`-Format wird in Abschnitt 2.3.1 genauer erläutert. Die Umwandlung wird mit folgendem Befehl vorgenommen:

```
cpfcsv2rfg clones.cpf clones.csv clones.rfg
```

`.rfg` ist ein binäres Dateiformat zur internen Verwendung von Bauhaus. Damit die Daten in SEE verwendet werden können, muss dieses Format noch in eine, für den Austausch von Graphen vorgesehende, GXL Datei umgewandelt werden. Dazu wird folgender Befehl ausgeführt:

```
rfgexport -o Clones -f GXL clones.rfg clones.gxl
```

- `-o` Ausgabename, auch Name des Graphen, welcher in einer GXL Datei enthalten ist.
- `-f` Ausgabeformat, in welches die RFG Datei umgewandelt werden soll.

Die drei Befehle würden bei Ausführung in der Kommandozeile funktionieren. Damit sie auch in Vcs2See funktionieren, muss z. B. der Pfad zu ausführbaren Programmen absolut sein und die Dateinamen müssen mit dem Index der Revision versehen werden. Ansonsten würden diese überschrieben.

Verwendung in Vcs2See

In Vcs2See wird zuerst der Befehl `analyser.before.command` im temporären Verzeichnis ausgeführt. Befehle wie `mkdir` oder `del` gehören zu der `cmd` von Windows und können deshalb auch nur innerhalb dieser Anwendung ausgeführt werden. Der folgende Befehl führt `mkdir` mit `cmd` aus:

```
cmd /c mkdir "%repository.name%"
```

Der Befehl legt ein Ordner mit dem Namen des Repository im temporären Verzeichnis an. Danach wird auf jede Revision die Analyse ausgeführt. Die Analyse besteht im Beispiel aus den drei Befehlen, welche oben erläutert sind. Zuerst wird der Pfad zu `cpf` mit einem Platzhalter als absoluter Pfad angegeben. Hiermit wird das Programm gefunden. Alternativ ließe sich der Befehl auch in der `cmd` ausführen, da die Pfade hier automatisch aufgelöst werden. Hiervon ist abzuraten, da sich dann eine komplizierte Schachtelung der Befehle ergibt. Der erste Befehl der Analyse ist fast derselbe wie oben angegeben. Die Dateinamen sind hier nicht festgelegt, sondern mit Platzhaltern versehen. So wird die Datei bei dem Repository-Namen `example` und der Revisionsnummer `1` unter `example/example-1.cpf` abgelegt.

```
"%environment.bauhaus%/cpf"
-B %repository.temp%
-m 100
%extensions%
-c %repository.name%/%filename%.cpf
-s %repository.name%/%filename%.csv
-t %repository.name%/%filename%
.
```

Zu jedem Befehl kann auch ein Verzeichnis konfiguriert werden, in welchem der Befehl ausgeführt werden soll. Bei der Angabe der Verzeichnisse lassen sich ebenfalls Platzhalter verwenden.

Der nächste Befehl wird in `%repository.temp%/%repository.name%` ausgeführt. Dies ist das Verzeichnis, welches im Vorbereitungsbefehl angelegt wird und in welches der erste Analyse-Befehl die Dateien ablegt.

In dem Wiki wird darauf hingewiesen, dass `cpfcsv2rfg` nicht mit der Axivion Suite ausgeliefert wird [4]. Das Programm muss separat installiert werden. Das Pyhton Skript `cpfcsv2rfg` ist in einem Verzeichnis abgelegt und der Pfad dahin in der Variable `environment.cpfcsv2rfg` angegeben. Das Skript wird in dem Verzeichnis ausgeführt und wandelt die `.cpf` und `.csv` Dateien zu einer `.rfg` Datei um. Da der Befehl bereits im Unterverzeichnis ausgeführt wird, muss dieses bei den Dateinamen nicht erneut angegeben werden. `cpfcsv2rfg` wird nicht direkt mit Python, sondern über ein Werkezug von Bauhaus (`rfgscript`) wie folgt ausgeführt:

```
"%environment.bauhaus%/rfgscript"
"%environment.cpfcsv2rfg%/cpfcsv2rfg.py"
%filename%.cpf %filename%.csv %filename%.rfg
```

Der letzte Befehl der Analyse wird ebenfalls im Unterverzeichnis ausgeführt. Der Pfad zu den Programmen ist in Anführungszeichen angegeben, da es unter Windows vorkommt, dass ein Ordnername Leerzeichen enthält. In diesem Fall wird alles nach dem Leerzeichen als Parameter interpretiert und es kommt zu Fehlern. Dieser Befehl wandelt die `.rfg` Datei zu einer `.gxl` Datei um:

```
"%environment.bauhaus%/rfgexport"
-o Clones
-f GXL
%filename%.rfg %filename%.gxl
```

Vor der Ausführung des Nachbereitungs-Befehls, wird die Analyse für jede Revision wiederholt. Am Ende liegt ein Verzeichnis mit dem Namen des Repository im temporären Verzeichnis, in welchem diverse Dateien mit einem Namen in dem Format `repositoryName-revisionIndex`

liegen, welche zur Verarbeitung benötigt werden. Diese Dateien werden in der Nachbereitung entfernt. Anschließend wird das Verzeichnis zu dem Ort der Ausführung von Vcs2See kopiert. Sollte das Verzeichnis dort schon vorhanden sein, wird es vorher gelöscht. Dazu werden vier Befehle verwendet, welche mit dem Operator && verknüpft werden.

```
cmd /S /c (del *.tok & del *.csv & del *.files & del *.cpf) &&
rmdir /s /q "%here%\\output\\%repository.name%" &&
mkdir "%here%\\output\\%repository.name%" &&
xcopy . "%here%\\output\\%repository.name%" /q
```

3.1.3 SEE (C#)

In SEE besteht der größte Teil darin, sich in das Programm einzulesen. Als Ausgangspunkt für die Recherche wurde das Skript der `SEE City Evolution` verwendet. Das Skript ist in der Datei `Assets/SEE/Game/City/SEECityEvolution.cs` zu finden.

Commit-Information im Graphen
Beim Laden eines Skriptes wird zuerst die Methode `Awake()` aufgerufen. Der Inhalt dieser Methode in der Klasse `SEECityEvolution` enthält einen Hinweis zum Laden der Graphen:
`List<Graph> graphs = LoadData();`
Von dieser Methode aus lassen sich die Aufrufe zu der Klasse verfolgen, in welcher das Einlesen der GXL Dateien implementiert ist. Der Pfad ist in Abbildung 12 dargestellt.

Abbildung 12: Aufrufe von der `SEECityEvolution` zum `GXLParser`

In der `GXLParser` Klasse ist ein `.NET` XML-Parser implementiert. Dieser durchläuft in einem `while` Konstrukt alle Elemente des Dokuments. In einem `switch-case` wird zwischen den Typen eines Elementes (siehe Abbildung 13) unterschieden. Bei Beginn und Ende eines Elementes wird jeweils eine Methode aufgerufen. Bei dem Typen `Text` wird der Wert in einer Variable zwischengespeichert. Der `GXLParser` selbst implementiert diese Methoden als `virtual`, sodass sie von einer abgeleiteten Klasse außer Kraft gesetzt werden können. Der `GraphReader` überschreibt manche der leeren Methoden mit dem Schlüsselwort `override`. Die Methode `StartNode` setzt das Attribut `current` vom Typen `GraphElement` auf eine `Node`, sodass die GXL Attribute innerhalb des Tags auf das Element gesetzt werden können. `EndNode` setzt current anschließend wieder auf `null` zur Bereinigung der Variable.

Abbildung 13: XML

`StartAttr()` prüft zu beginn der Methode, ob `current == null` ist und wirft die Fehlermeldung „Attributdeklaration außerhalb einer Knoten-/Eckendeklaration gefunden“, sollte das zutreffen. Um Attribute in Graphen zu lesen ist hier eine Änderung erforderlich.

Dynamische Marker

Die existierenden Marker in der SEECity Evolution sollen verwendet werden, um weitere Metriken zu visualisieren. Die Länge des Markers lässt sich als Visualisierung für eine weitere Metrik verwenden. Zusätzlich ließe sich der Marker wie in Abbildung 14 in einzelne Segmente unterteilen, die einen Teil einer Gesamtmenge als gestapeltes Balkendiagram darstellen können.

Abbildung 14: Beispiel Marker Segmente

Die Implementierung der Marker ist nicht direkt in der SEECityEvolution zu finden, jedoch in der Klasse EvolutionRenderer, welchen ebenfalls im Awake() initialisiert wird. Der EvolutionRenderer ruft je nach existierenden Markertypen marker.MarkBorn(), marker.MarkChanged() oder marker.MarkDead() auf. Die Methoden haben als gemeinsamen Nenner den Aufruf der Methode MarkByBeam(). In dieser Methode muss eine Fallunterscheidung für dynamische und statische Marker hinzugefügt und dementsprechend die Erstellung des Objektes angepasst werden.

Menü zur Konfiguration dynamischer Marker

Um die dynamischen Marker zu konfigurieren muss ein weiteres Menü zur SEECity Evolution hinzugefügt werden. Folgende Felder sollen in dem Menü enthalten sein:

- Typ des Markers: Dynamisch oder Statisch. Wenn der Typ Statisch ausgewählt ist, ändert sich nichts an der Anwendung. Die Marker haben eine einheitliche Länge, die Farbe wird von der Änderung an der Datei bestimmt. Die dynamischen Marker sind wie in 3.1.3 beschrieben.
- Minimale Länge des Markers: Eine untere Begrenzung der Länge des Markers. Da die Länge durch Metriken bestimmt werden kann ist es sinnvoll, eine minimale Länge anzugeben, damit der Marker immer zu sehen ist.
- Maximale Länge des Markers: Dieselbe Begründung wie bei der minimalen Länge. Wenn die Marker durch die Metriken zu lang werden, ist das Ende nicht mehr zu sehen. Dennoch ist es eine optische Konfiguration.
- Segmente: Für die einzelnen Segmente auf einem dynamischen Marker muss eine sortierte Liste konfiguriert werden. Die Listenelemente enthalten jeweils einen Metriknamen und eine Farbe. Der Reihenfolge nach sollen die Segmente später visualisiert werden.

Die vorhanden Menüs sind in der Klasse SEECityEvolutionEditor zu finden.

4 Implementierung

Im folgenden Kapitel wird die Implementierung der beiden Komponenten beschrieben und auf Probleme eingegangen.

4.1 Vcs2See (Java)

Das Java Projekt wird als neues `Maven` Projekt mit `IntelliJ IDEA` aufgesetzt. Zuerst wird die Hauptklasse `Vcs2See` mit der Methode `main(String[] args)` als Ausgangspunkt der Anwendung angelegt. Alle weiteren Klassen können wie in dem Klassendiagramm aus Abbildung 10 inklusive der Methoden, Attribute und Konstruktoren angelegt werden. Die Körper der Methoden bleiben vorerst leer.

Da die GXL-Java-API aus Abschnitt 2.4.3 als `.jar` und nicht als Maven Packet zur Verfügung gestellt wird, wird auf der obersten Ebene des Projektes ein `lib/` Ordner angelegt, in welchen die Bibliothek abgelegt wird. Die Bibliothek wird dann wie folgt zu den `Maven Dependencies` hinzugefügt:

```
<dependency>
    <groupId>net.sourceforge.gxl</groupId>
    <artifactId>gxl</artifactId>
    <version>1.0</version>
    <scope>system</scope>
    <systemPath>${basedir}/lib/gxl.jar</systemPath>
</dependency>
```

Als weitere Bibliotheken werden LibVCS4j für das Extrahieren der Informationen aus den VCS, Lombok zum Einsparen von redundanten Quelltext-Abschnitten und Log4j um die Protokollierung von LibVCS4j zu steuern, hinzugefügt.

Als nächstes werden der `PropertiesManager` und der `ConsoleManager` implementiert, da es sich um zwei Hilfsklassen handelt, welche von dem Rest des Programms benötigt werden. Beide Klassen enthalten ausschließlich Methoden zur Einsparung von Redundanzen.

Im nächsten Schritt wird der `RepositoryCrawler` implementiert, da dieser die nächste Ebene der Verarbeitung darstellt. Die Reihenfolge der Implementierung folgt dem Ablaufdiagramm aus Abbildung 9.

`RepositoryCrawler.java`
Der `RepositoryCrawler` hat eine Methode `crawl()` zur Initialisierung von LibVCS4j. Bei der Initialisierung muss wird der Typ des VCS aus der Konfiguration gelesen. Je nach Typ wird in einem `switch-case` die zugehörige Methode aufgerufen.

```
Type type = Type.valueOf(propertiesManager.getProperty("repository.type"));

switch (type) {
    case GIT:
        engine = VCSEngineBuilder.ofGit(path).build();
        break;
```

Nach der Initialisierung kann das temporäre Verzeichnis, in welchem LibVCS4j seine Checkout's ausführt, ermittelt werden. Das Verzeichnis wird zur Laufzeit in die Konfiguration unter dem Schlüssel `repository.temp` bereitgestellt. Aus Kompabilitätsgründen müssen Backslashes (\) mit einem Doppel-Backslash (\\) ersetzt werden. Diese dienen ansonsten als Escape-Charakter für andere Zeichen und der Pfad ist ungültig. Dieses Vorgehen wird im Programm mehrfach genutzt, jedoch dort nicht erneut erläutert. Die verbleibende Methode `nextRevision()` aus dem Klassendiagramm ist ein Wrapper für `engine.next()`.

`CodeAnalyser.java`
Der `CodeAnalyser` besitzt keine Methode zur Initialisierung, allerdings die Methoden `prepare()` und `postprocess()`, welche jeweils vor und nach allen Analysen ausgeführt werden sollten.

```
public void prepare() throws IOException {
    PropertiesManager pm = Vcs2See.getPropertiesManager();
    String cmd = pm.getProperty("analyser.before.command").orElseThrow();
    String dir = pm.getProperty("analyser.before.directory").orElseThrow();
    run(replacePlaceholders(cmd, 0), replacePlaceholders(dir, 0));
}
```

Die Besonderheit, die diesem Quelltext zu entnehmen ist, ist die Methode `replacePlaceholders()`, welche jeweils auf den Befehl und das Verzeichnis aufgerufen wird.

Die Methode `replacePlaceholders()` sucht mit einer Regex-Expression alle Platzhalter aus einem Text heraus und ersetzt diese mit dem Wert aus der Konfigurationsdatei. Wenn in der Konfiguration kein Wert zu dem Schlüssel vorhanden ist, wird der Platzhalter entfernt. Zusätzlich gibt es weitere Platzhalter:

- `%here%` wird mit dem Verzeichnis, in dem das Programm ausgeführt wird, ersetzt.
- `%filename%` wird mit dem Namen des Repository, konkateniert mit einem - und der Revisionsnummer, ersetzt.
- `%extensions%` wird mit allen möglichen Dateiendungen der konfigurierten Programmiersprache, als `-i` Parameter, ersetzt. (Beispiel: `-i "*.c" -i "*.h"`)

Die eigentliche Analyse wird von der Methode `analyse()` ausgeführt. Die Methode prüft ob der Befehl mit dem Schlüssel `analyser.n.command` existiert, wobei `n` mit jeder Iteration erhöht wird. Wenn der Schlüssel in der Konfiguration nicht vorhanden ist, gibt es keine weiteren Befehle und die Analyse ist abgeschlossen.

`GraphModifier.java`
Die letzte Komponente ist der `GraphModifier`, welcher mit jeder Revision das `Commit`-Objekt und die Nummer übergeben bekommt. Zuerst wird die GXL Datei, dessen Pfad über den Schlüssel `modifier.path` konfiguriert werden kann, geladen und anschließend die Methoden der Klasse in folgender Reihenfolge ausgeführt:

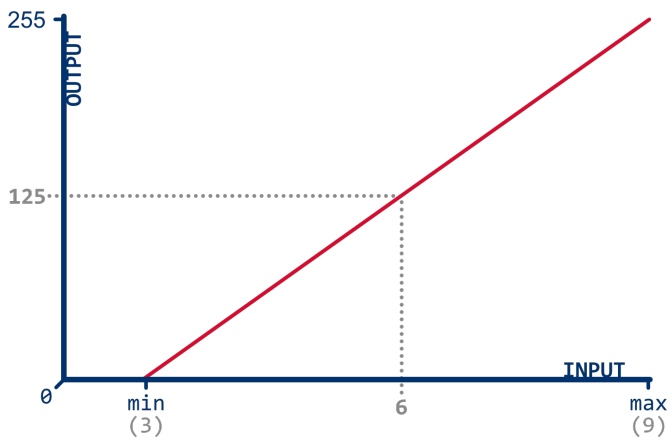
1. `loadNodes()` - Laden aller Knoten aus der `gxl` Datei. Die Knoten werden in eine `Map<String, GXLNode>` hinzugefügt. Der `String` soll der Pfad zu der Datei sein, welche der Knoten repräsentiert und die `GXLNode` ist das Objekt aus der Java-GLX-API, welches später verändert werden soll.
2. `loadCommit()` - Versucht die veränderten Dateien aus dem `Commit` von LibVCS4j den `GLXNode`'s über den Pfad zuzuordnen. Bei einem Erfolg werden Attribute, die nicht

aggregiert werden müssen direkt zur `GXLNode` hinzugefügt. Die weiteren Daten werden ebenfalls in Datenstrukturen zur späteren Berechnung hinzugefügt. Dieser Prozess wird im Anschluss detailierter beschrieben.

3. `populateNodes()` - Berechnet mit den Methoden `calculateMostRecent()` und `calculateMostFrequent()` die Werte für die am häufigsten und zuletzt veränderten Knoten und fügt diese zu dem Knoten hinzu.
4. `addCommitGraph()` - Fügt die Informationen über den Commit wie den Autor, das Datum, die Kurznachricht und die ID zu dem Graphen hinzu.
5. Speichern der GXL Datei.

Um die letzten und häufigsten Änderungen an Dateien festzustellen, werden zwei Datenstrukturen bei dem Durchlaufen der Versionshistorie erweitert. Die Werte werden jeweils von einer Funktion auf den Zahlenraum 0-255 interpoliert, da der Farbraum so viele Farbtöne zwischen zwei Farben unterstützt.

Die Interpolation funktioniert wie in Abbildung 4.1 gezeigt: Zuerst werden Minimal- und Maximalwert des Eingabe-Zahlenraums (`INPUT`) bestimmt. In dem Beispiel sind das 3 und 9. Anschließend wird ein Wert aus diesem Zahlenraum in die lineare Funktion eingegeben und es ergibt sich ein Wert auf dem Ziel-Zahlenraum (`OUTPUT`). 6 ist die Mitte zwischen 3 und 9, so ergibt sich 125 zwischen 0 und 255 (gerundet).



Die mathematische Funktion einer linearen Interpolation auf 0 bis 255 ergibt sich wie folgt:
`output = (255 / (max - min)) * (input - min)`

Die Daten müssen vor der Interpolation erfasst werden. Hierfür ist bei beiden Daten ein anderes Vorgehen nötig:

a) **Letzte Änderungen**:
Datenstruktur:

```
private final Map<String, Integer> mostFrequent;
```

Aufruf bei Änderung:

```
if(mostFrequent.computeIfPresent(path, (k, v) -> v + 1) == null) {
    mostFrequent.put(path, 1);
}
```

Wenn der Schlüssel bereits vorhanden ist, wird der Wert um 1 erhöht, ansonsten wird der Schlüssel mit dem Wert 1 hinzugefügt. Der Minimal- und Maximalwert ergibt sich aus den Werten in der `Map`.

b) **Häufigste Änderungen**:
Datenstruktur:

```
private final Deque<String> mostRecent;
```

Aufruf bei Änderung:

```
mostRecent.remove(path);
mostRecent.addFirst(path);
```

Wenn die Datei bereits in der `Deque` vorhanden ist, wird sie entfernt. Anschließend wird diese an den Anfang der `Deque` angefügt. Nach jedem Aufruf sind die Elemente entsprechend der Aufrufzeit chronologisch sortiert.

`Vcs2See.java`

Im Folgenden müssen die implementierten Komponenten in der Hauptklasse `Vcs2See` zusammengefügt werden. Der Ausgangspunkt des Programmes liegt in dieser Klasse als `main()` Methode. Begonnen wird mit dem Laden der Konfigurationsdatei. Unmittelbar darauf folgt die Einrichtung, welche durch das setzen eines Startparameters übersprungen werden kann.

```
if(!Boolean.parseBoolean(System.getProperty("ci", "false"))) {
    vcs2See.setup();
} else {
    consoleManager.print("SETUP\nProgram was started in CI mode...");
    consoleManager.printSeparator();
}
```

Der Startparameter wird wie folgt gesetzt:
`java -jar vcs2see.jar -Dci=true`

Das Überspringen der `setup()` Methode bewirkt, dass keine manuellen Eingaben auf der Kommandozeile erwartet werden. Andernfalls gibt das Programm jeden Schlüssel aus der Konfigurationsdatei mit einem Wert aus und bietet die Möglichkeit einen neuen Wert zu setzen.

```
private void read(String key) throws IOException {
    Optional<String> value = propertiesManager.getProperty(key);
    consoleManager.print("Current value: " + value.orElse("<empty>"));
    String newValue = consoleManager.readLine(key + "=");
    if(!newValue.isBlank()) {
        propertiesManager.setProperty(key, newValue);
    }
}
```

Wenn der, durch den Nutzer eingegebene Wert keine Zeichen enthält (durch Bestätigung mit `<Enter>` ohne Eingabe), wird der bisherige Wert beibehalten. Die Methode `read()` ist eine Hilfsmethode, welche für jeden zu konfigurierenden Schlüssel aus der Datei aufgerufen wird.

Anschließend wird der `RepositoryCrawler` initialisiert, die erste Revision mit `repositoryCrawler.nextRevision()` geladen und die `prepare()` Methode aufgerufen. Grund für diese Reihenfolge ist die Beobachtung, dass das temporäre Verzeichnis bis zum Laden der ersten Revision nicht erstellt wird und die `prepare()` Methode deshalb keine Befehle auf das Verzeichnis ausführen kann. Um die erste Revision in der Berechnung zu beachten wird ein `do-while` statt einem `while` Konstrukt genutzt.

```
Optional<RevisionRange> optional = repositoryCrawler.nextRevision();
```

```
int index = 1;
do {
    for(Commit commit : optional.orElseThrow().getCommits()) {
        codeAnalyser.analyse(index);
        graphModifier.process(commit, index);
        index++;
    }
} while ((optional = repositoryCrawler.nextRevision()).isPresent());
```

In dem `do-while` wird jeder `Commit` aus der Revision in einer `for`-Schleife durchlaufen und darauf der `CodeAnalyser` und der `GraphModifier` ausgeführt.

Nach der Analyse aller Commit's wird die `postprocess()` des `RepositoryCrawler` zur Ausführung der Nachbereitung aufgerufen. Damit ist das Programm beendet.

4.2 SEE (C#)

4.2.1 Commit-Informationen im Graphen

Im Entwurf wird festgestellt, dass die Methode `StartAttr()` im `GraphReader` angepasst werden muss. Attribute werden nach der Anpassung außerhalb von Kanten und Knoten ohne Ausgabe von Fehlern akzeptiert. Hierfür wird eine umschließend `if`-Abfrage entfernt.

`Assets/SEE/DataModel/DG/IO/GraphReader.cs`

```
protected override void StartAttr()
{
- . if (ReferenceEquals(current, null))
- . {
- ..... LogError("Found attribute declaration outside of a node/edge");
- . }
- . else
- . {
        if (reader.HasAttributes)
        {
            while (reader.MoveToNextAttribute())
            {
                if (reader.Name == "name")
                {
                    // save for later when we know the attribute type
                    currentAttributeName = reader.Value;
                    break;
                }
            }
        }
        else
        {
            LogError("Attribute declaration without name.");
        }
- . }
}
```

Bevor die Werte im Graphen gesetzt werden können, müssen zu diesem die entsprechenden Attribute mit `gettern` und `settern` hinzugefügt werden.

`Assets/SEE/DataModel/DG/Graph.cs`

```
+ public string CommitId { get; set; }

+ public string CommitMessage { get; set; }

+ public string CommitAuthor { get; set; }

+ public string CommitTimestamp { get; set; }
```

Da die Variable `current` vom Typen `GraphElement` ist und nicht auf den Graphen gesetzt werden kann, muss die `EndString()` Methode angepasst werden. Die Methode enthält die

gleiche `if`-Abfrage wie in `StartAttr()`, daher würde hier ein Fehler ausgegeben werden. Statt den Fehler auszugeben, wird eine Fallunterscheidung für den Attributnamen hinzugefügt und das dementsprechende Attribut im Graphen gesetzt.

`Assets/SEE/DataModel/DG/IO/GraphReader.cs`

```
protected override void EndString(string value)
{
    if (ReferenceEquals(current, null))
    {
+ ..... if(currentAttributeName == "CommitId")
+ ..... {
+ ......... graph.CommitId = value;
+ ..... }
+ ..... else if (currentAttributeName == "CommitMessage")
+ ..... {
+ ......... graph.CommitMessage = value;
+ ..... }
+ ..... else if (currentAttributeName == "CommitAuthor")
+ ..... {
+ ......... graph.CommitAuthor = value;
+ ..... }
+ ..... else if (currentAttributeName == "CommitTimestamp")
+ ..... {
+ ......... graph.CommitTimestamp = value;
+ ..... }

        // LogError("Found string attribute outside of a node/edge declaration."); - y
    }
    else if (currentAttributeName == "")
    {
        LogError("There is not attribute name for this string.");
    }
    else
    {
        current.SetString(currentAttributeName, value);
    }
}
```

Die Informationen werden zu dem `Graph` hinzugefügt. Im nächsten Schritt werden die Informationen grafisch dargestellt. Das Objekt `C` hat dasselbe Verhalten wie das zu implementierende Objekt. Es zeigt Text, welcher mit jeder Revision verändert wird.

Da `AnimationCanvas` ein Prefab ist, muss dieses erst geöffnet werden, bevor eine Kopie von der `TextRevisionNumber` angelegt werden kann. Die Kopie wird `CommitInformationText` genannt (siehe `A`).

Nachdem die Kopie (`B`) angelegt ist, bekommt diese einen Platzhalter-Text in dem Feld `D`. Anschließend kann das Objekt positioniert und skaliert werden.

Das `AnimationCanvas` enthält das Skript `AnimationDataModel`. Dieses Skript muss ebenfalls um den `CommitInformationText` erweitert werden. Der Wert vom `RevisionNumberText` wird in der Klasse `AnimationInteraction` und der Methode `OnShownGraphHasChanged()` verändert. Der Text für den `CommitInformationText` muss verändert werden.

`Assets/SEE/Game/Evolution/AnimationInteraction.cs`

```
private void OnShownGraphHasChanged()
{
    animationDataModel.RevisionNumberText.text = (evolutionRenderer.CurrentGraphIndex
+ . animationDataModel.CommitInformationText.text =
+ ..... "Commit #" + evolutionRenderer.GraphCurrent.CommitId +
+ ..... "\nAuthor: " + evolutionRenderer.GraphCurrent.CommitAuthor +
+ ..... "\nTimestamp: " + evolutionRenderer.GraphCurrent.CommitTimestamp +
+ ..... "\nMessage:\n" + evolutionRenderer.GraphCurrent.CommitMessage;
    animationDataModel.Slider.value = evolutionRenderer.CurrentGraphIndex;
}
```

4.2.2 Dynamische Marker Menü

Zur Umsetzung des Menüs wird ein Teil des vorhandenen Quelltextes kopiert und angepasst. Mit folgendem Quelltext kann das Menü aus Abbildung 15 umgesetzt werden:

```
private void MarkerAttributes()
{
    showMarkerAttributes = EditorGUILayout.Foldout(showMarkerAttributes,
        "Attributes of markers", true, EditorStyles.foldoutHeader);
    if (showMarkerAttributes)
    {
        MarkerAttributes settings = city.MarkerSettings;

        settings.Kind = (MarkerKinds)EditorGUILayout
            .EnumPopup("Type", settings.Kind);
        settings.LengthMetric = EditorGUILayout
            .TextField("Length", settings.LengthMetric);
        settings.MinimalMarkerLength = Mathf.Max(0, EditorGUILayout
            .FloatField("Minimal lengths", settings.MinimalMarkerLength));
        settings.MaximalMarkerLength = EditorGUILayout
            .FloatField("Maximal lengths", settings.MaximalMarkerLength);

        EditorGUI.indentLevel++;

        SerializedProperty sections = serializedObject
            .FindProperty("MarkerSettings.MarkerSections");
        EditorGUILayout.PropertyField(sections,
            new GUIContent("Marker sections"), true);

        EditorGUI.indentLevel--;

    }
}
```

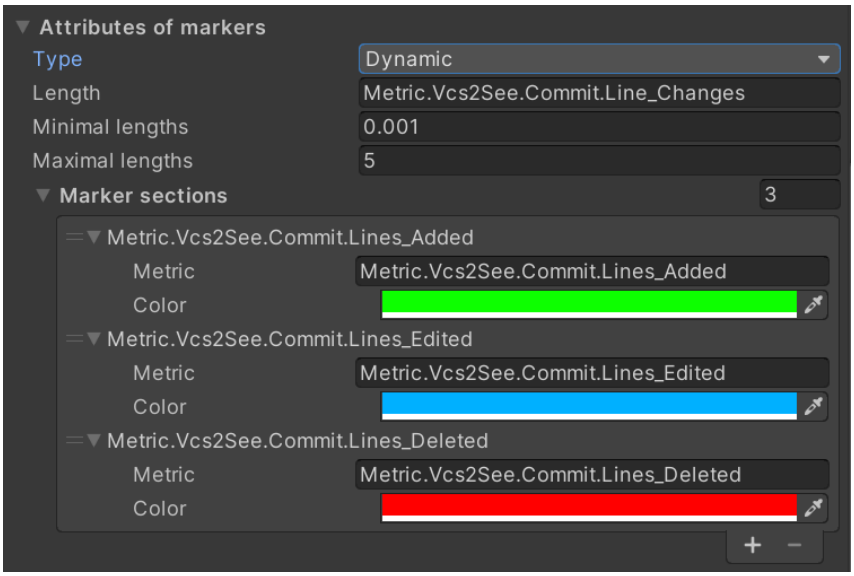


Abbildung 15: Menü zur Konfiguration der Marker

4.2.3 Dynamische Marker

Zur Implementierung der dynamischen Marker muss die `MarkByBeam()` Methode in der Klasse `Assets/SEE/Game/Evolution/Marker.cs` angepasst werden. Der Quelltext der Methode wird von einer Fallunterscheidung umschlossen und aufgerufen, wenn der Typ der Marker auf statisch gesetzt ist. Wenn die Marker auf dynamisch gesetzt sind, wird der Quelltext in abgewandelter Form aufgerufen. Wie in Abbildung 16 gezeigt wird jeder Wert aus der konfigurierbaren Liste nacheinander als Marker hinzugefügt. Die Position des ersten Markers liegt direkt auf dem Knoten. Die Position des zweiten Markers ist um die Länge des ersten nach oben verschoben, damit die Marker sich nicht überschneiden. Das gleiche Prinzip wird auf alle weiteren Marker angewendet. Die Länge der Marker wird aus der kofnigurierten Metrik entnommen.

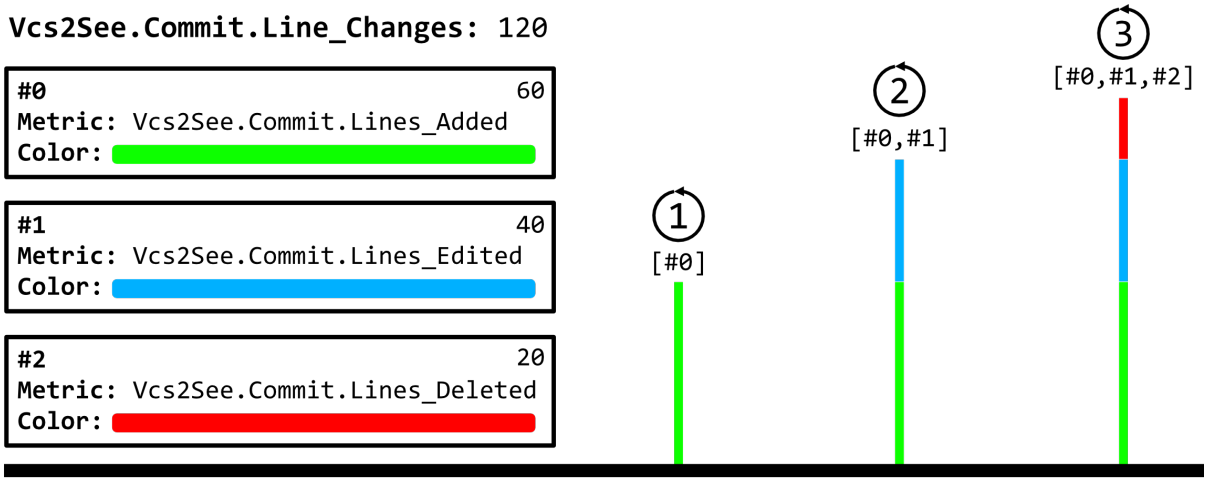


Abbildung 16: Funktionsweise `MarkByBeam()` bei dynamischen Markern

Vorab wird ein neues `GameObject` erzeugt und alle Marker bekommen dieses als `parent` gesetzt. Bei den statischen Markern kann der einzelne Marker von der Funktion zurückgegeben werden, sodass das Objekt später weiterverarbeitet werden kann. Bei den dynamischen Markern wird das neue `GameObject` zurückgegeben, damit alle Marker bei z. B. einer Entfernung beachtet werden.

5 Evaluation

Im folgenden Kapitel wird die Implementierung dieser Bachelorarbeit gegen ein anderes Werkzeug zur Analyse eines Versionskontrollsystems evaluiert.

5.1 System Usability Scale

Die System Usability Scale (SUS) ist einer der bekanntesten Fragebögen zur Evaluation der Nutzbarkeit von Programmen. Veröffentlicht wird er 1996 von John Brooke als „schnelle und schmutzige“ Methode zur Bewertung der Benutzerfreundlichkeit von Programmen. Der Ansatz bietet einen kostengünstigen, allgemeinen Maßstab, zum Vergleich der Gebrauchstauglichkeit. [15]

Im Original ist der Fragebogen auf Englisch verfasst. 2013 veröffentlicht SAP eine Übersetzung auf Deutsch. [16]

Der Fragebogen besteht aus 10 Aussagen. Zu bewerten sind diese Aussagen auf einer Likert-Skala mit den Antworten „trifft nicht zu“ bis „trifft zu“. Auf die Likert-Skala wird genauer in Abschnitt 5.2 eingegangen. Die Aussagen sind abwechselnd positiv und negativ. Um mehr Konsistenz in den Antworten des Probanden zu erhalten, können Paare von widersprüchlichen Aussagen verwendet werden. [15]

5.2 Likert-Skala

Die Likert-Skala ist eine Skala um die Einstellung einer befragten Person zu einem Thema zu erfassen. Dazu wird ein Likert-Test mit Aussagen zur Zustimmung oder Ablehnung erstellt. Die Antwort wird auf einer Skala gegeben, welche üblicherweise aus 5, 7 oder 11 Feldern besteht. Die Entscheidung fällt zwischen den beiden Extrema „trifft zu“ und „trifft nicht zu“. Bei ungeraden Skalen hat die befragte Person die Option sich nicht für eine Seite zu entscheiden. [17] [18]

5.3 Einschränkungen

Zur Zeit der Durchführung hält die COVID-19 Pandemie bereits 2 Jahre an. Physische Treffen sind möglich, werden zum Schutz der Teilnehmer jedoch vermieden. Die Evaluation wird Online durchgeführt. Die Benutzer bekommen ein Programm zur Ausführung. Durch Inkompatibilitäten kann es dazu kommen, dass Freiwillige an die Studie nicht teilnehmen können. Daten wie die Dauer der Bearbeitung werden nicht von einem Versuchsleiter aufgenommen, sondern in Selbstverantwortung von den Probanden. Es kann zu Ungenauigkeiten kommen.

Als Vergleichstool wird eine Website gewählt, da andere Alternativen eine Systeminstallation des Probanden erfordern. Diese Begebenheit senkt die Bereitschaft zur Teilnahme enorm und wird über die Nutzung eines Online-Vergleichstools umgangen.

5.4 Google Formulare

Für die Durchführung der Studie wird „Formulare“ von Google verwendet. Das Umfragewerkzeug ermöglicht die Erstellung von mehreren Abschnitten. Die Abschnitte können mit verschiedenen Typen von Eingabefeldern oder Infotexten gefüllt und nacheinander aufgerufen werden. Die Ergebnisse der Umfrage werden in „Tabellen“ von Google gespeichert und können nach Microsoft Excel exportiert werden.

5.5 Fragebogen

Der Fragebogen befindet sich im vollen Umfang im Anhang der Bachelorarbeit.

Um an der Evaluation teilzunehmen muss der Proband drei Voraussetzungen erfüllen. Diese Grundvoraussetzungen werden im ersten Abschnitt mit Kästchen abgefragt. Der Proband muss alle Kästchen ankreuzen, um zu dem nächsten Abschnitt zu gelangen. Dadurch soll unaufmerksamen Lesern vorgebeugt werden.

Im nächsten Abschnitt wird der Kenntnisstand des Probanden abgefragt. Erfasst werden das Alter, Erfahrungen mit Versionskontrollsystemen und der höchste Bildungsabschluss. Die Erfahrungen mit den Versionskontrollsystemen dienen einem späteren Vergleich mit der Statistik aus Abbildung 3.

Der dritte Abschnitt ist der erste praktische Teil der Evaluation. Hier sollen die Probanden jeweils drei Aufgaben für zwei Projekte mit GitHub bearbeiten. Der zweite praktische Teil ist äquivalent, mit Ausnahme der Bearbeitung mit SEE.

Zu Beginn der praktischen Abschnitte wird die Bedienung erklärt. Am Ende der Erklärung wird eine Frage gestellt, welche zur Beantwortung das Verständnis über die meisten Bedienelemente erfordert. Die Antwort wird validiert und der Proband kann die Evaluation nur mit einer korrekten Antwort fortsetzen. Dadurch soll sichergestellt werden, dass der Teilnehmer sich mit dem Werkzeug vorher beschäftigt und die Bedienelemente versteht. Der Lernprozess während der Bearbeitung der Aufgaben, ist in der Auswertung nicht zu beachten.

Die drei Aufgaben sind in beiden praktischen Abschnitten dieselben. Jede Aufgabe wird doppelt, jedoch mit einem anderen Projekt, gestellt. Daraus Ergeben sich in dem Fragebogen z. B. die Aufgaben `1a` und `1b`.

- **Aufgabe 1**: Erkennung iterativer Softwareentwicklung
- **Aufgabe 2**: Finden der am häufigsten veränderten Datei
- **Aufgabe 3**: Finden der Datei mit den meisten Änderungen

Jede Aufgabe wird mit einer kurzen Erklärung eingeleitet, um einen möglichst geringen Kenntnisstand bei den Probanden vorauszusetzen. Darauf folgt die Frage mit zusätzlichen Hinweisen zur Beantwortung dieser. Am Ende der Frage wird der Proband z. B. aufgefordert die Stoppuhr zu starten. Der letzte Bestandteil einer Frage ist das Zeitfeld. Hier wird der Proband, nach dem Ausfüllen der Antwort, aufgefordert die Stoppuhr zu stoppen und die gemessene Zeit einzutragen.

Zwischen den beiden praktischen Abschnitten wird der Teilnehmer zu einer Pause aufgefordert. Da die drei Antworten sich für die beiden zu vergleichenden Werkzeuge nicht unterscheiden,

wird der Teilnehmer abgelenkt. So sollen die Ergebnisse nicht aus der Erinnerung dokumentiert werden.

Am Ende eines praktischen Abschnittes wird der Proband aufgefordert eine System Usability Scale (Abschnitt 5.1) zur Bewertung des Werkzeugs auszufüllen.

Zum Abschluss der Umfrage wird abgefragt, ob es technische Schwierigkeiten während der Umfrage oder sonstige Anmerkungen gibt. Bei Ausreißern oder unschlüssigen Antworten, hat der Proband hier die Möglichkeit eine Erklärung dafür abzugeben. Die Anmerkungen können verwendet werden, um Verbesserungsvorschläge für den Fragebogen aufzunehmen.

5.6 Veröffentlichung

Der Fragebogen wird auf verschiedenen Kanälen z. B. an der Universität Bremen veröffentlicht. Die meisten Personen werden innerhalb von 24 Stunden durch Push-Benachrichtigungen auf ihrem Smartphone aufmerksam und beginnen bei Interesse die Bearbeitung zeitnah. Die Anzahl der Teilnehmer pro Tag würde fortlaufend abklingen. Das Problem eines solchen Online-Fragebogens liegt in der zeitversetzten und parallelen Bearbeitung. Es ist sehr wahrscheinlich, dass erste Fehler auftreten, wenn bereits Probanden teilgenommen haben. Die Daten dieser Teilnehmer sind gegebenenfalls nicht aussagekräftig oder müssen angepasst werden.

Bei einer Präsensveranstaltung können nach jedem Teilnehmer Anpassungen zur Verständniserleichterung vorgenommen werden. Zudem besteht ein direkter Kontakt zu den Teilnehmern. Ähnlich wie bei iterativer Softwareentwicklung würde der Fragebogen Evaluation für Evaluation verbessert werden.

Um diesen Vorteil auch im Online-Format zu gewinnen, wird der Fragebogen nach und nach an kleinere Gruppen ausgerollt. Nach drei Iterationen wird weiträumig veröffentlicht, da die grundlegenden Fehler verbessert sind und es keine Breaking-Changes mehr gibt.

5.7 Ergebnisse

Im folgenden Abschnitt erfolgt die Auswertung der Evaluation. Dazu wird die Berechnung einer SUS erläutert und anschließend durchgeführt. Weitere Daten, wie die Fehlerquote, Bearbeitungszeit und Vorerfahrungen, werden ebenfalls betrachtet und Rückschlüsse daraus gezogen.

An der Evaluation haben 13 Probanden teilgenommen. Das Durchschnittsalter der Probanden liegt bei 26 Jahren. Der Großteil der Probanden hat mindestens eine Berufsausbildung oder einen Bachelor.

5.7.1 Auswertung System Usability Scale

Die SUS wird mit der, von John Brooke entwickelten, Formel zur Bewertung einer SUS (siehe Abbildung 18, links), ausgewertet [15]. Den Antworten der Probanden werden nummerische Werte zugeordnet. Da die Fragen im Fragebogen abwechselnd positiv und negativ gestellt werden, wird jeder zweite Wert invertiert. Zur Invertierung wird die Anzahl der Antwortmöglichkeiten benötigt (als **x** angegeben).

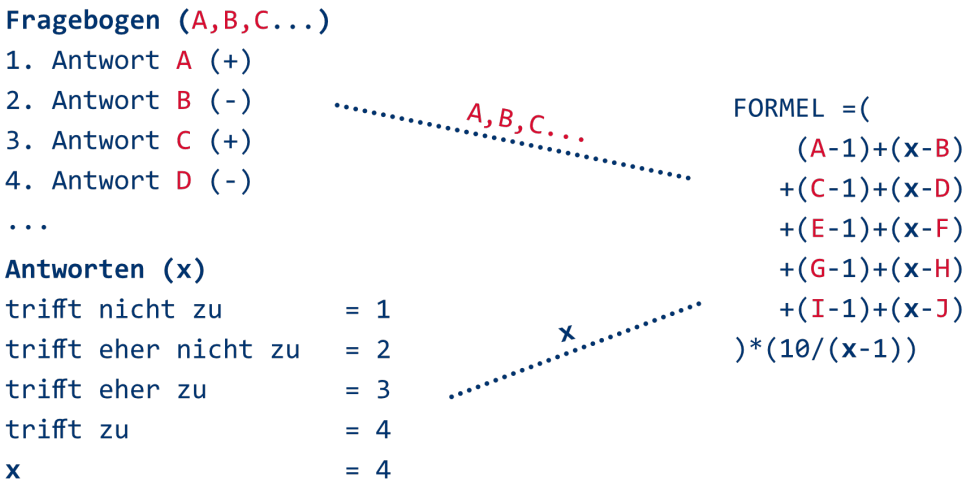


Abbildung 17: Auswertung der SUS

A steht jeweils für den nummerischen Wert der Antwort des Probanden. Die Zuweisung von Antworten zu Werten ist der Abbildung 18 (unten links) zu entnehmen.

Die folgende Abbildung enthält die Ergebnisse der Evaluation. Die Ergebnisse wurde bereits zu den nummerischen Werten konvertiert. Die Spalte S enthält den SUS-Score, welcher mit der Formel errechnet wurde. Der SUS-Score liegt auf einer Skala von 0 bis 100.

1	2	3	4	5	6	7	8	9	10	S
4	1	4	1	3	2	1	2	2	1	73,33
2	2	3	1	4	1	4	4	2	2	66,66
4	2	3	1	3	1	4	1	4	1	90
3	2	4	3	3	3	4	1	3	1	73,33
2	2	3	3	3	2	3	2	2	3	53,33
3	1	4	1	3	1	4	1	3	1	90
4	1	4	1	4	2	3	1	4	1	93,33
2	3	2	3	2	4	2	3	2	4	26,66
4	2	3	1	3	1	2	2	3	2	73,33
2	3	2	1	2	1	2	3	1	2	46,66
1	2	3	4	3	2	1	2	1	4	33,33
2	3	2	3	2	3	1	3	1	3	26,66
1	1	3	1	3	1	4	1	2	1	76,66

GitHub Durchschnitt: 63,33

1	2	3	4	5	6	7	8	9	10	S
1	4	2	3	2	1	1	3	2	3	30
3	1	4	1	2	3	4	1	3	1	80
3	1	4	1	3	1	4	1	4	1	93,33
3	2	4	1	3	2	4	1	4	1	86,66
3	2	3	3	4	2	3	2	3	3	63,33
4	1	4	1	4	1	4	1	4	2	96,66
2	2	2	1	3	3	2	2	2	1	56,66
3	2	4	2	4	2	3	2	3	2	73,33
1	2	3	1	3	2	2	1	3	2	63,33
3	2	4	3	4	2	4	1	4	2	80
3	2	1	2	3	1	3	2	3	2	63,33
4	1	3	2	3	1	3	2	4	2	80
2	3	3	1	2	1	4	1	1	2	63,33

SEE Durchschnitt: 71,53

Abbildung 18: Ergebnisse der SUS

SEE ist im Durchschnitt, mit einer Differenz im SUS-Score von ~8 Punkten, besser abgeschnitten. Mit $63,33$ liegt der SUS-Score von GitHub unter dem Durchschnitt von 68 Punkten. SEE liegt mit $71,53$ Punkten über dem Durchschnitt. [19] Für die Probanden war die Aufgabenbearbeitung mit SEE leichter durchzuführen, als mit GitHub.

5.7.2 Auswertung Fehler und Bearbeitungszeit

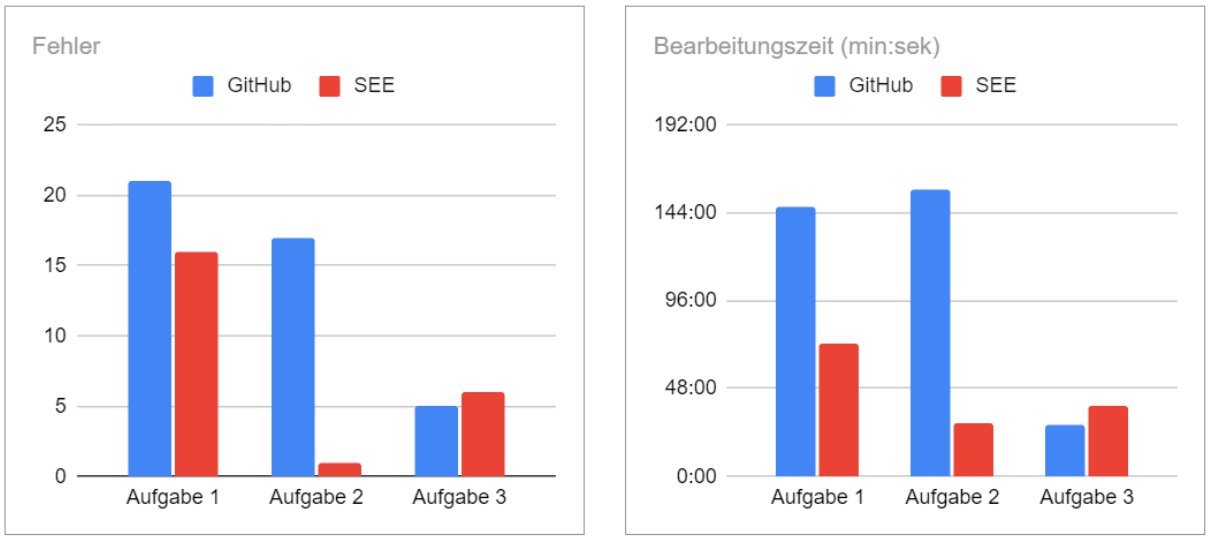


Abbildung 19: Weitere Ergebnisse der Evaluation

Zur Auswertung der Fehler, wird von der Anzahl der gegebenen Antworten, die Anzahl der richtigen Antworten abgezogen. Die Fehlerquote der Probanden bei der Bearbeitung der Aufgaben 1 und 2 fiel bei der Nutzung von GitHub erheblich höher aus. Bei der Bearbeitung der dritten Aufgabe wurde von den Probanden in SEE eine fehlerhafte Antwort mehr gegeben, als mit GitHub. Die Fehler für Aufgabenteil a und b werden für die Auswertung jeweils aufaddiert.

Die Bearbeitungszeiten der Aufgaben werden ebenfalls aufsummiert. Auch hier wurden die Aufgabenteile zusammengerechnet. Wie auch bei der Auswertung der Fehler, sind in Aufgabe 1 und 2 die Werte erheblich höher für GitHub, als für SEE und in Aufgabe 3 anders herum.

Aus der Fehlerquote sowie der benötigten Bearbeitungszeit lässt sich schließen, dass die Lösung der dritten Aufgabe mit GitHub leichter durchzuführen ist.

5.7.3 Abgleich Erfahrung mit Versionskontrollsystemen

Die Erfahrung der Probanden mit den Versionskontrollsystemen entspricht der Darstellung aus Abbildung 3. Die Probanden können ihren Kentnissstand ebenfalls mit Hilfe einer Likert-Skala angeben. Für „Viel Erfahrung“ gibt es dementsprechend 3 Punkte und für „Keine Erfahrung“ 0 Punkte. Die Skala hat 4 Stufen zur Auswahl. Die Werte auf der x-Achse entsprechen den summierten Likert-Punkten aller Probanden.

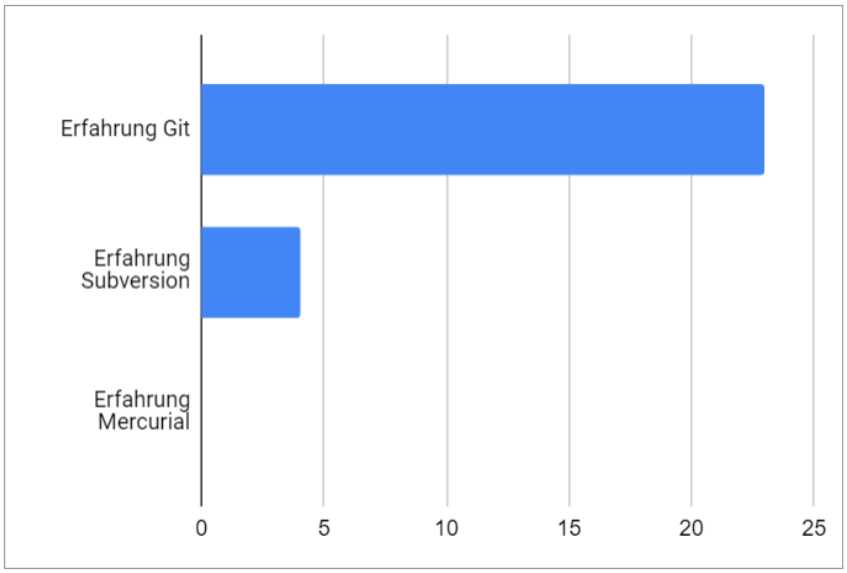


Abbildung 20: Erfahrung der Probanden mit Versionskontrollsystemen

6 Fazit und Ausblick

6.1 Fazit

Im Rahmen dieser Bachelorarbeit wurde ein Programm entwickelt, welches den Quelltext aus einem VCS einließt und jede Revision zu einer GXL Datei verarbeitet. Die Informationen aus dem VCS werden parallel verarbeitet und ebenfalls in die GXL Dateien eingepflegt. Das entstandene Verzeichnis wird von SEE eingelesen und wie ein Video abgespielt. Die zusätzlichen Informationen werden unter anderem mit den neuen dynamischen Markern in SEE visualisiert. Die neuen Marker können verwendet werden, um eine Metrik in der Länge des Markers darzustellen oder mehrere Metriken mit den farbigen Segmenten, ähnlich wie bei einem Kreisdiagramm, zu vergleichen.

Die Evaluation dieser Bachelorarbeit hat gezeigt, dass Probanden mit einem erheblich niedrigeren Zeitaufwand Informationen über das Versionskontrollsystem erhalten können, es als Bedienungsfreundlicher empfunden haben und weniger Fehler mit dem System gemacht haben.

Bei der dritten Aufgabe der Evaluation ist SEE schlechter abgeschnitten als GitHub. Die Aufgabe bestand darin, eine Metrik zu vergleichen und den größten Wert ausfindig zu machen. Die Handhabung einer Website Oberfläche ist schneller als die Bedienung einer komplexen, 3-dimensionalen Umgebung. Außerdem ist der Vergleich verschiedener Längen unpräziser, als der Vergleich verschiedener Zahlen. SEE ließe sich hier insofern verbessern, dass man die Daten vorverarbeitet und die Größenunterschiede eindeutiger darstellt. Alternativ wäre das Anzeigen der nummerischen Werte hilfreich, um Entscheidungen zwischen zwei ähnlich langen Markern zuverlässiger treffen zu können.

6.2 Ausblick

In diesem Kapitel wird ein Ausblick darüber gegeben, wie die Ergebnisse dieser Bachelorarbeit verwendet werden können, um SEE zu erweitern oder diese Bachelorarbeit fortzusetzen.

6.2.1 Visualisierung der Autorenschaft

Die erstellten GXL Dateien enthalten unter Anderem den Autor einer Revision. Die Autoren können als Karten mit deren Profilbildern und Namen in einem Ring um die Stadt visualisiert werden. Von den Karten aus gehen dann Kanten zu den bearbeiteten Knoten und bleiben für zehn Revisionen bestehen. Wenn eine Kante erneut bearbeitet wurde, wird dieser Zähler zurückgesetzt und die Kante bleibt erneut für die nächsten zehn Revisionen bestehen. Als Inspiration für diese Funktion ist das Projekt „Gource“ in Betracht zu ziehen (https://gource.io/).

6.2.2 Berücksichtigung mehrerer Zweige

In dieser Bachelorarbeit wurden nur die Änderungen auf dem Hauptzweig betrachtet. In einer weiteren Ausarbeitung könnte, statt einem Zeitstrahl (weißer Balken in Abbildung 8), eine interaktive Visualisierung einer Versionshistorie (wie in Abbildung 2) verwendet werden, um durch die Revisionen zu navigieren. Der Vorteil wäre, dass der Nutzer weitere Branches betrachten kann. Dazu müssten die GXL Dateien in einer anderen Struktur abgelegt werden,

um zwischen den einzelnen Zweigen zu unterscheiden. Eine Möglichkeit wäre, die Hierarchie der Dateisysteme zu benutzen und jeweils einen Unterordner für einen Zweig zu erstellen.

6.2.3 Aggregation weiterer Statistiken

Neben den am häufigsten veränderten und zuletzt veränderten Dateien können auch weitere Daten aus dem VCS aggregiert werden. Die neuen Statistiken müssen als neue Metrik zur Verfügung gestellt werden, indem sie zur GXL Datei hinzugefügt werden.

III | Literaturverzeichnis

[1] S. Bhatia und J. Malhotra, "A survey on impact of lines of code on software complexity," in *2014 International Conference on Advances in Engineering Technology Research (ICAETR - 2014)*, 2014, S. 1–4. DOI: 10.1109/ICAETR.2014.7012875.

[2] J.-J. Tedro. (2018). "kernelstats," Adresse: https://github.com/udoprog/kernelstats (besucht am 29. 12. 2021).

[3] R. Wettel und M. Lanza, "Visualizing Software Systems as Cities," in *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, Los Alamitos, CA, USA: IEEE Computer Society, Juni 2007, S. 92–99. DOI: 10.1109/VISSOF.2007.4290706. Adresse: https://doi.ieeecomputersociety.org/10.1109/VISSOF.2007.4290706.

[4] U. o. B. Software Engineering Group. (2019). "SEE," Adresse: https://github.com/uni-bremen-agst/SEE/wiki (besucht am 29. 12. 2021).

[5] M. Steinbeck, "Mining version control systems and issue trackers with LibVCS4j," *SANER '20 / edited by Kostas Kontogiannis, Foutse Khomh, Alexander Chatzigeorgiou, Marios-Eleftherios Fokaefs, and Minghui Zhou ; sponsored by: IEEE Computer Society; Western University, Canada ; supported by: TCSE*, S. 647–651, 2020. Adresse: http://dx.doi.org/10.1109/SANER48275.2020.9054841.

[6] S. Otte, "Version Control Systems," Jan. 2009. Adresse: https://www.mi.fu-berlin.de/inf/groups/ag-tech/teaching/2008-09_WS/S_19565_Proseminar_Technische_Informatik/otte09version.pdf.

[7] M. Steinbeck. (2021). "LibVCS4j readme," Adresse: https://github.com/uni-bremen-agst/libvcs4j/blob/master/README.md (besucht am 17. 12. 2021).

[8] Wikipedia, *Bauhaus Project (computing)*, http://en.wikipedia.org/w/index.php?title=Bauhaus%20Project%20(computing)&oldid=988310284, 2021. (besucht am 29. 12. 2021).

[9] T.-T.-I. GmbH. (2021). "AXIVION," Adresse: https://www.tti-stuttgart.de/axivion/ (besucht am 29. 12. 2021).

[10] A. Raza, G. Vogel und E. Plödereder, "Bauhaus – A Tool Suite for Program Analysis and Reverse Engineering," in *In Reliable Software Technologies, Ada Europe 2006 (LNCS 4006*, 2006, S. 71.

[11] R. Holt, A. Schürr, S. E. Sim und A. Winter. (2021). "Graph eXchange Language," Adresse: https://userpages.uni-koblenz.de/~ist/GXL/index.php (besucht am 29. 12. 2021).

[12] E. Larsson. (2021). "GXL-Java-API," Adresse: http://gxl.sourceforge.net/docs/gxl/api/index.html (besucht am 29. 12. 2021).

[13] S. Diehl, "Softwarevisualisierung," *Informatik-Spektrum*, Jg. 26, Nr. 4, S. 257–260, Aug. 2003, ISSN: 1432-122X. DOI: 10.1007/s00287-003-0314-4. Adresse: https://doi.org/10.1007/s00287-003-0314-4.

[14] R. Wettel, M. Lanza und R. Robbes, "Software Systems as Cities: A Controlled Experiment," in *Proceedings of the 33rd International Conference on Software Engineering*, Ser. ICSE '11, Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, S. 551–560, ISBN: 9781450304450. DOI: 10.1145/1985793.1985868. Adresse: https://doi.org/10.1145/1985793.1985868.

[15] J. Brooke, "SUS: A quick and dirty usability scale," *Usability Eval. Ind.*, Jg. 189, Nov. 1995.

[16] B. Rummel, "System Usability Scale (Translated into German)," Apr. 2013.

[17] "5-Point Likert Scale," in *Handbook of Disease Burdens and Quality of Life Measures*, V. R. Preedy und R. R. Watson, Hrsg. New York, NY: Springer New York, 2010, S. 4288–4288, ISBN: 978-0-387-78665-0. DOI: 10.1007/978-0-387-78665-0_6363. Adresse: https://doi.org/10.1007/978-0-387-78665-0_6363.

[18] "Likert Scale," in *Handbook of Disease Burdens and Quality of Life Measures*, V. R. Preedy und R. R. Watson, Hrsg. New York, NY: Springer New York, 2010, S. 4248–4248, ISBN: 978-0-387-78665-0. DOI: 10.1007/978-0-387-78665-0_6017. Adresse: https://doi.org/10.1007/978-0-387-78665-0_6017.

[19] A. S. f. P. Affairs, *System Usability Scale (SUS)*, Sep. 2013. Adresse: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

IV | Anhang

(Auf der letzten Seite der Bindung befindet sich eine CD mit dem Quelltext von Vcs2See)

Evaluation

Bachelorarbeit: Extraktion von Daten aus Versionskontrollsystemen zur anschließenden Visualisierung in einer CodeCity

* Erforderlich

1. E-Mail-Adresse *

Guten Tag!

Vielen Dank für Ihre Teilnahme an der Evaluation zu meiner Bachelorarbeit.

Im Folgenden werden Sie zwei Werkzeuge zur Visualisierung von Versionshistorien vergleichen, indem Sie die Werkzeuge ausprobieren und anschließend ein paar Fragen dazu beantworten. Die Evaluation wird maximal 40 Minuten in Anspruch nehmen und benötigt Windows oder Linux mit Wine und eine Stoppuhr zur Teilnahme.

2. Wenn Sie bereit für die Evaluation sind und die Voraussetzungen erfüllen, dann bestätigen Sie bitte unten. *

Wählen Sie alle zutreffenden Antworten aus.

- [ ] Stoppuhr
- [ ] Windows / Linux mit Wine
- [ ] 40 Minuten Zeit

Kenntnisse

In diesem Abschnitt möchte ich Ihren Kenntnisstand abfragen.

3. Wie alt sind Sie? *

4. Wie viel Erfahrung haben Sie bereits mit folgenden Versionskontrollsystemen gesammelt? *

Markieren Sie nur ein Oval pro Zeile.

	Keine Erfahrung	Wenig Erfahrung	Mehr Erfahrung	Viel Erfahrung
Git	◯	◯	◯	◯
Subversion	◯	◯	◯	◯
Mercurial	◯	◯	◯	◯

5. Was ist Ihr höchster Bildungsabschluss? *

Markieren Sie nur ein Oval.

- ◯ Ohne Abschluss
- ◯ Abitur / (Fach-)Hochschulreife
- ◯ Berufsausbildung
- ◯ Bachelor
- ◯ Master
- ◯ Diplom
- ◯ Promotion

GitHub

VORBEREITUNG

6. Wie viele Erfahrungen haben Sie bereits mit GitHub gesammelt? *

Markieren Sie nur ein Oval.

	1	2	3	4	
Keine	◯	◯	◯	◯	Sehr viel

Öffnen

Bitte öffnen Sie die folgenden zwei Links in einem Browser Ihrer Wahl. Die Links müssen vor jeder Frage neu geöffnet werden, damit Sie überall den gleichen Ausgangspunkt haben. Die Links brauchen Sie nicht speichern, da Sie vor jeder Frage noch einmal zur Verfügung gestellt werden.

(JavaExample)
https://github.com/winterbe/java8-tutorial/commits/master?after=faf9793524cd6956428fa5d79e7fdce3a3184b56+139

(Vcs2See)
https://github.com/FelixGaebler/vcs2see/commits/master?after=c9eca23146645d51efc7530035484ba86e1f8fef+34

Bedienung

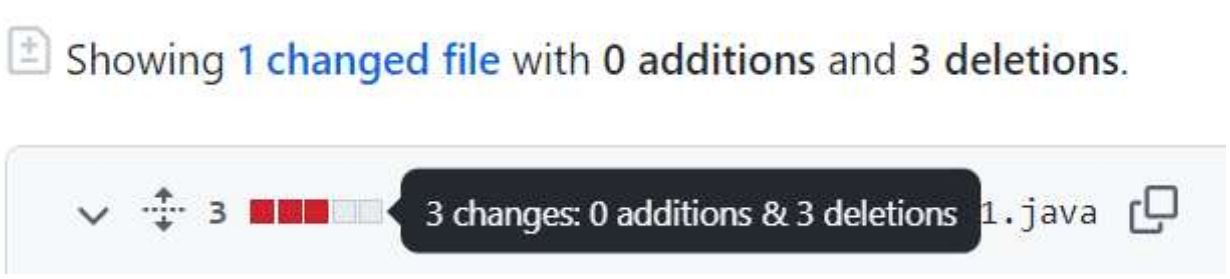
- Auf der Seite sehen Sie eine Versionshistorie.
- Die Revisionen sind absteigend nach Datum sortiert.
- Auf der rechten Seite ist ein Feld mit den ersten sieben Stellen der Commit-ID.
- Der Knopf mit den beiden sich überlappenden Quadraten kopiert die gesamte Commit-ID in die Zwischenablage.
- Wenn man auf die Nachricht der Revision klickt, wird man auf die Detail-Seite der Revision weitergeleitet.

Detail-Seite:
- Oben rechts finden Sie die Commit-ID hinter "commit".
- Die grünen/grauen/roten Quadrate vor jedem Dateipfad auf der linken Seite zeigen die Anzahl der Änderungen.
- Wenn man über diesen Balken mit der Maus schwebt, werden mehr Informationen angezeigt.
- (Siehe Bild unten)

- Die Seite zeigt die ersten 35 Revisionen. Sie sind durch den Link bereits auf der letzten Seite.
- Mit den Knöpfen "Newer" und "Older" am unteren Ende der Seite, können Sie zwischen den Seiten wechseln.

WICHTIGER HINWEIS
Bearbeiten Sie in den Aufgaben die Revisionen von hinten nach vorne. Beginnen Sie also ganz unten auf der Seite und wenn Sie ganz oben angekommen sind, benutzen Sie den "Newer" Knopf, um auf die nächste Seite zu gelangen, wo Sie wieder von unten Anfangen.

(Balken auf Detail-Seite)



Ablauf der Umfrage

Sie werden gleich abwechselnd die gleiche Frage für beide Links gestellt bekommen.
Bitte lesen Sie trotzdem den gesamten Informationstext erneut, da es teilweise Abwandlungen in der Aufgabenstellung gibt.

Die Stoppuhr starten Sie jedes Mal, wenn Sie die Fragestellung fertig gelesen haben und stoppen sie anschließend wieder, wenn Sie die Antwort fertig eingegeben haben. Sie werden im Fragebogen dazu auch explizit aufgefordert.

Nach der Beantwortung von drei Fragen pro Link, werden Sie aufgefordert die Plattform kurz zu evaluieren. Daraufhin geht es zum zweiten Teil der Befragung mit einem anderen Werkzeug.

7. Bedienung verstanden? *

Öffnen Sie den JavaExample Link und machen Sie sich mit der Bedienung vertraut. Was ist die Commit-ID der ersten Revision?

GitHub - Aufgabe 1a

AUFGABE 1 (JavaExample)

Iterative Softwareentwicklung

Bei iterativer Softwareentwicklung wird ein Bestandteil oder ein Projekt als Ganzes verworfen und durch einen Nachfolger ersetzt. In einem Versionskontrollsystem lässt sich ein solches Vorgehen identifizieren, wenn man Revisionen findet, in welchen es kaum Änderungen am Quelltext gab, sondern eine Menge Dateien gelöscht und parallel dazu, eine Menge Dateien hinzugefügt wurden.

TIPP
Achte auf die Balken vor den Dateipfaden auf den Detail-Seiten. Wenn fast alle Balken einfarbig (grün oder rot) sind, ist das ein starkes Indiz.

Frage: Gibt es eine Revision mit dieser Eigenschaft?

1. Geben Sie eine Commit-ID an.
2. Die Commit-ID finden Sie in dem weißen Informationstext hinter der Raute.
3. Antwort sollte ungefähr so aussehen: "faf9793524cd6956428fa5d79e7fdce3a3184b56"
4. Geben Sie das Ergebnis ohne Raute an.
5. Wenn Sie keine Revision mit diesen Eigenschaften identifizieren können, antworten Sie bitte mit "-".
6. Wenn Sie mehrere Revisionen mit diesen Eigenschaften identifizieren können, trennen Sie die Antworten mit ", ".

Bitte starten Sie nun die Stoppuhr und öffnen anschließend den Link "JavaExample".

Link

https://github.com/winterbe/java8-tutorial/commits/master?after=faf9793524cd6956428fa5d79e7fdce3a3184b56+69

8. Antwort *

9. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

AUFGABE 1 (Vcs2See)

GitHub - Aufgabe 1b

Iterative Softwareentwicklung

Bei iterativer Softwareentwicklung wird ein Bestandteil oder ein Projekt als Ganzes verworfen und durch einen Nachfolger ersetzt. In einem Versionskontrollsystem lässt sich ein solches Vorgehen identifizieren, wenn man Revisionen findet, in welchen es kaum Änderungen am Quelltext gab, sondern eine Menge Dateien gelöscht und parallel dazu, eine Menge Dateien hinzugefügt wurden.

TIPP
Achte auf die Balken vor den Dateipfaden auf den Detail-Seiten. Wenn fast alle Balken einfarbig (grün oder rot) sind, ist das ein starkes Indiz.

Frage: Gibt es eine Revision mit dieser Eigenschaft?

1. Geben Sie eine Commit-ID an.
2. Die Commit-ID finden Sie in dem weißen Informationstext hinter der Raute.
3. Antwort sollte ungefähr so aussehen: "faf9793524cd6956428fa5d79e7fdce3a3184b56"
4. Geben Sie das Ergebnis ohne Raute an.
5. Wenn Sie keine Revision mit diesen Eigenschaften identifizieren können, antworten Sie bitte mit "-".
6. Wenn Sie mehrere Revisionen mit diesen Eigenschaften identifizieren können, trennen Sie die Antworten mit ", ".

Bitte starten Sie nun die Stoppuhr und öffnen anschließend den Link "Vcs2See".

Link

https://github.com/FelixGaebler/vcs2see/commits/master?after=c9eca23146645d51efc7530035484ba86e1f8fef+34

10. Antwort *

11. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

GitHub - Aufgabe 2a

AUFGABE 2 (JavaExample)

Frage: Welche Datei wurde bis zur 15ten Revision am häufigsten Verändert?

1. Die Nachricht der Revision lautet: "playing around with nashorn javascript extensions"
2. Antwort sollte ungefähr so aussehen: "ExampleAnswer.java"
3. Wenn mehrere Dateien in Frage kommen entscheiden Sie sich für eine.
4. Sie dürfen jedes beliebige Hilfsmittel zur Auswertung benutzen (Stift, Papier, Editor, Programm...).
5. Zählen Sie nur .java Dateien, der Rest ist kein Quellcode.

(EMPFEHLUNG)
Führen Sie auf einem Blatt Papier eine Strichliste mit den Dateinamen und gehen Sie Revision für Revision durch.
Die Datei mit den meisten Streichen (die in den meisten Revisionen vorkam) wurde am häufigsten Verändert.

Bitte starten Sie nun die Stoppuhr und öffnen anschließend den Link "JavaExample".

Link

https://github.com/winterbe/java8-tutorial/commits/master?after=faf9793524cd6956428fa5d79e7fdce3a3184b56+139

12. Antwort *

13. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

GitHub - Aufgabe 2b

AUFGABE 2 (Vcs2See)

Frage: Welche Datei wurde bis zur 15ten Revision am häufigsten Verändert?

1. Die Nachricht der Revision lautet: "Auto stash before merge of 'master' and 'origin/master'"
2. Antwort sollte ungefähr so aussehen: "ExampleAnswer.java"
3. Wenn mehrere Dateien in Frage kommen entscheiden Sie sich für eine.
4. Sie dürfen jedes beliebige Werkzeug zur Auswertung benutzen (Stift, Papier, Editor...).
5. Zählen Sie nur .java Dateien, der Rest ist kein Quellcode.

(EMPFEHLUNG)
Führen Sie auf einem Blatt Papier eine Strichliste mit den Dateinamen und gehen Sie Revision für Revision durch.
Die Datei mit den meisten Streichen (die in den meisten Revisionen vorkam) wurde am häufigsten Verändert.

Bitte starten Sie nun die Stoppuhr und öffnen anschließend den Link "Vcs2See".

Link

https://github.com/FelixGaebler/vcs2see/commits/master?after=c9eca23146645d51efc7530035484ba86e1f8fef+34

14. Antwort *

15. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

    Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

GitHub - Aufgabe 3a

AUFGABE 3 (JavaExample)

Frage: In Revision #8b96094 wurden in welcher Datei die meisten Zeilen verändert?

1. Die Nachricht der Revision lautet: "Add more stream examples"
2. Antwort sollte ungefähr so aussehen: "ExampleAnswer.java"
3. Wenn mehrere Dateien in Frage kommen entscheiden Sie sich für eine.

Bitte starten Sie nun die Stoppuhr und öffnen anschließend den Link "JavaExample".

Link

https://github.com/winterbe/java8-tutorial/commits/master?after=faf9793524cd6956428fa5d79e7fdce3a3184b56+104

16. Antwort *

17. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

GitHub - Aufgabe 3b

AUFGABE 3 (Vcs2See)

Frage: In Revision #d672cb5 wurden in welcher Datei die meisten Zeilen verändert?

1. Die Nachricht der Revision lautet: "Changed data models"
2. Antwort sollte ungefähr so aussehen: "ExampleAnswer.java"
3. Wenn mehrere Dateien in Frage kommen entscheiden Sie sich für eine.

Bitte starten Sie nun die Stoppuhr und öffnen anschließend den Link "Vcs2See".

Link

https://github.com/FelixGaebler/vcs2see/commits/master?before=c9eca23146645d51efc7530035484ba86e1f8fef+35

18. Antwort *

19. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

GitHub

BEWERTUNG

20. Bitte beantworten Sie folgende Fragen: *

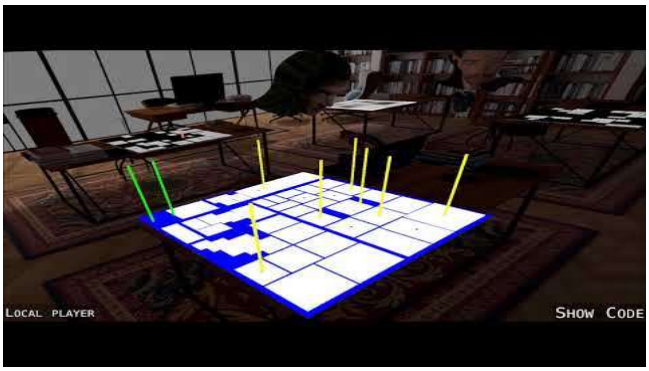
Markieren Sie nur ein Oval pro Zeile.

	trifft nicht zu	trifft eher nicht zu	trifft eher zu	trifft zu
Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.	◯	◯	◯	◯
Ich empfinde das System als unnötig komplex.	◯	◯	◯	◯
Ich empfinde das System als einfach zu nutzen.	◯	◯	◯	◯
Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.	◯	◯	◯	◯
Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.	◯	◯	◯	◯
Ich finde, dass es im System zu viele Inkonsistenzen gibt.	◯	◯	◯	◯
Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	◯	◯	◯	◯
Ich empfinde die Bedienung als sehr umständlich.	◯	◯	◯	◯
Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.	◯	◯	◯	◯
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.	◯	◯	◯	◯

Kurze Pause

Wenn Sie eine Pause benötigen, können Sie diese jetzt einlegen.

Hier wäre zum Beispiel ein kurzer Trailer zu dem folgenden Werkzeug:



http://youtube.com/watch?v=V2WpRtKF5wM

SEE

VORBEREITUNG

21. Wie viel Erfahrung haben Sie bereits mit SEE gesammelt? *

Markieren Sie nur ein Oval.

	1	2	3	4	
Keine	◯	◯	◯	◯	Sehr viel

Herunterladen

Bitte laden Sie die folgenden zwei zip-Dateien herunter und entpacken Sie die Archive auf Ihrem Computer. Achten Sie darauf, die Ordner wie in den Klammern vor den URLs zu benennen, damit Sie die Beispiele später identifizieren können.

(JavaExample)
https://drive.google.com/file/d/1XCpUcRARP3qfNe7CIwrNR6NE1pTBDMX9/view?usp=sharing

(Vcs2See)
https://drive.google.com/file/d/11yqEpdw6uMEbgQ-Kte9_HQc93N-6Q1zV/view?usp=sharing

Bedienung

- Benutzen Sie W, A, S, D als Steuerkreuz zum Navigieren durch die Welt.
- Benutzen Sie L, um die Tasten A und D zu aktivieren.
- Benutzen Sie die Pfeiltasten links und rechts um eine einzelne Revision vor- oder zurückzuspringen.
- Mit der Tastenkombination Alt+F4 können Sie die Anwendung schließen.
- Halten Sie die rechte Maustaste gedrückt und bewegen Sie den Mauszeiger, um sich umzusehen.
- Wenn Sie mit dem Mauszeiger über ein Gebäude schweben, sehen Sie den Namen der Datei.
- Auf der rechten Seite des Podests finden Sie die Steuerung für die Animation.
- Zum Abspielen der Animation können Sie die Steuerung an dem Podest benutzen.
- Bitte benutzen Sie nur die beiden einfachen Pfeile zum Abspielen der Animation.
- Bevor Sie die Animation in die andere Richtung abspielen, stoppen Sie die Animation in die aktuelle Richtung.
- Unterhalb der Steuerung ist ein Informationstext. Die Zeichen hinter der Route sind die Commit-ID der Revision.
- Wenn etwas nicht mehr Funktioniert wie erwartet, starten Sie das Programm neu.

22. Bedienung verstanden? *

Starten Sie die SEE.exe aus dem JavaExample Ordner und machen Sie sich mit der Bedienung vertraut. Was sind die ersten 5 Zeichen der Commit-ID ohne Raute?

SEE - Aufgabe 1a

AUFGABE 1 (JavaExample)

Iterative Softwareentwicklung

Bei iterativer Softwareentwicklung wird ein Bestandteil oder ein Projekt als Ganzes verworfen und durch einen Nachfolger ersetzt. In einem Versionskontrollsystem lässt sich ein solches Vorgehen identifizieren, wenn man Revisionen findet, in welchen es kaum Änderungen am Quelltext gab, sondern eine Menge Dateien gelöscht und parallel dazu, eine Menge Dateien hinzugefügt wurden.

Frage: Gibt es eine Revision mit dieser Eigenschaft?

1. Geben Sie die ersten 6 Zeichen einer Commit-ID an.
2. Die Commit-ID finden Sie in dem weißen Informationstext hinter der Raute.
3. Antwort sollte ungefähr so aussehen: "6afcb0"
4. Geben Sie das Ergebnis ohne Raute an.
5. Wenn Sie keine Revision mit diesen Eigenschaften identifizieren können, antworten Sie bitte mit "-".
6. Wenn Sie mehrere Revisionen mit diesen Eigenschaften identifizieren können, trennen Sie die Antworten mit ", ".

Bitte starten Sie nun die Stoppuhr und anschließend die SEE.exe in dem Ordner "JavaExample".

23. Antwort *

24. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

AUFGABE 1 (Vcs2See)

SEE - Aufgabe 1b

Iterative Softwareentwicklung

Bei iterativer Softwareentwicklung wird ein Bestandteil oder ein Projekt als Ganzes verworfen und durch einen Nachfolger ersetzt. In einem Versionskontrollsystem lässt sich ein solches Vorgehen identifizieren, wenn man Revisionen findet, in welchen es kaum Änderungen am Quelltext gab, sondern eine Menge Dateien gelöscht und parallel dazu, eine Menge Dateien hinzugefügt worden.

Frage: Gibt es eine Revision mit dieser Eigenschaft?

1. Geben Sie die ersten 6 Zeichen einer Commit-ID an.
2. Die Commit-ID finden Sie in dem weißen Informationstext hinter der Raute.
3. Antwort sollte ungefähr so aussehen: "6afcb0"
4. Geben Sie das Ergebnis ohne Raute an.
5. Wenn Sie keine Revision mit diesen Eigenschaften identifizieren können, antworten Sie bitte mit "-".
6. Wenn Sie mehrere Revisionen mit diesen Eigenschaften identifizieren können, trennen Sie die Antworten mit ", ".

Bitte starten Sie nun die Stoppuhr und anschließend die SEE.exe in dem Ordner "Vcs2See".

25. Antwort *

26. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

AUFGABE 2 (JavaExample)

SEE - Aufgabe 2a

Farbskala der Gebäude in SEE

Die Farbe der Gebäude in SEE ist einer Farbskala zugeordnet. Sie gibt die Häufigkeit an, wie oft eine Datei verändert wurde.
Die am wenigsten bearbeitete Datei ist weiß und die am häufigsten bearbeitete Datei rot.

Keine Änderungen Viele Änderungen

Frage: Welche Datei wurde bis zur 15ten Revision am häufigsten Verändert?

1. Die Nachricht der Revision lautet: "playing around with nashorn javascript extensions"
2. Antwort sollte ungefähr so aussehen: "ExampleAnswer.java"
3. Wenn mehrere Gebäude in Frage kommen entscheiden Sie sich für eines.

Bitte starten Sie nun die Stoppuhr und anschließend die SEE.exe in dem Ordner "JavaExample".

27. Antwort *

28. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

SEE - Aufgabe 2b

AUFGABE 2 (Vcs2See)

Farbskala der Gebäude in SEE

Die Farbe der Gebäude in SEE ist einer Farbskala zugeordnet. Sie gibt die Häufigkeit an, wie oft eine Datei verändert wurde.
Die am wenigsten bearbeitete Datei ist weiß und die am häufigsten bearbeitete Datei rot.

Keine Änderungen Viele Änderungen

Frage: Welche Datei wurde bis zur 15ten Revision am häufigsten Verändert?

1. Die Nachricht der Revision lautet: "Auto stash before merge of 'master' and 'origin/master'"
2. Antwort sollte ungefähr so aussehen: "ExampleAnswer.java"
3. Wenn mehrere Gebäude in Frage kommen entscheiden Sie sich für eines.

Bitte starten Sie nun die Stoppuhr und anschließend die SEE.exe in dem Ordner "Vcs2See".

29. Antwort *

30. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

AUFGABE 3 (JavaExample)

SEE - Aufgabe 3a

Marker in SEE

Über den Gebäuden in SEE werden teilweise bunte Strahlen angezeigt. Diese Strahlen nennen sich Marker und visualisieren Änderungen in Dateien.

Die Höhe der Marker ist relational zu der Anzahl an Änderungen in der Datei. Die Art der Änderungen verteilt sich prozentual über die drei verschiedenen Farbsegmente des Markers:
- grün: hinzugefügte Zeilen
- blau: bearbeitete Zeilen
- rot: gelöschte Zeilen

Frage: In Revision #8b96094 wurden in welcher Datei die meisten Zeilen verändert?

1. Die Nachricht der Revision lautet: "Add more stream examples"
2. Antwort sollte ungefähr so aussehen: "ExampleAnswer.java"
3. Wenn mehrere Dateien in Frage kommen entscheiden Sie sich für eine.

Bitte starten Sie nun die Stoppuhr und anschließend die SEE.exe in dem Ordner "JavaExample".

31. Antwort *

32. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

AUFGABE 3 (Vcs2See)

SEE - Aufgabe 3b

Marker in SEE

Über den Gebäuden in SEE werden teilweise bunte Strahlen angezeigt. Diese Strahlen nennen sich Marker und visualisieren Änderungen in Dateien.

Die Höhe der Marker ist relational zu der Anzahl an Änderungen in der Datei. Die Art der Änderungen verteilt sich prozentual über die drei verschiedenen Farbsegmente des Markers:
- grün: hinzugefügte Zeilen
- blau: bearbeitete Zeilen
- rot: gelöschte Zeilen

Frage: In Revision #d672cb wurden in welcher Datei die meisten Zeilen verändert?

1. Die Nachricht der Revision lautet: "Changed data models"
2. Antwort sollte ungefähr so aussehen: "ExampleAnswer.java"
3. Wenn mehrere Dateien in Frage kommen entscheiden Sie sich für eine.

Bitte starten Sie nun die Stoppuhr und anschließend die SEE.exe in dem Ordner "Vcs2See".

33. Antwort *

34. Stoppen Sie nun die Stoppuhr. Wie lange hat Ihre Bearbeitung der Aufgabe gedauert? *

Bitte geben Sie Ihre Antwort in dem Format <min>:<sec> an.

35. Bitte beantworten Sie folgende Fragen: *

Markieren Sie nur ein Oval pro Zeile.

	trifft nicht zu	trifft eher nicht zu	trifft eher zu	trifft zu
Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.	◯	◯	◯	◯
Ich empfinde das System als unnötig komplex.	◯	◯	◯	◯
Ich empfinde das System als einfach zu nutzen.	◯	◯	◯	◯
Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.	◯	◯	◯	◯
Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.	◯	◯	◯	◯
Ich finde, dass es im System zu viele Inkonsistenzen gibt.	◯	◯	◯	◯
Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	◯	◯	◯	◯
Ich empfinde die Bedienung als sehr umständlich.	◯	◯	◯	◯
Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.	◯	◯	◯	◯
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.	◯	◯	◯	◯

Feedback

36. Gab es technische Schwierigkeiten während der Umfrage? *

Markieren Sie nur ein Oval.

- Ja
- Nein

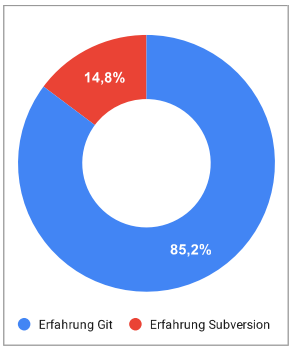
37. Wenn es technische Schwierigkeiten gab: Bitte erläutern Sie das Problem.

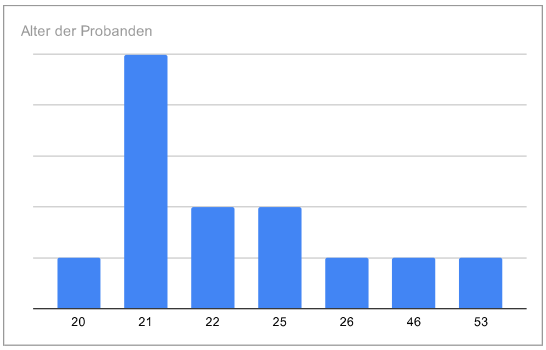
38. Sonstige Anmerkungen

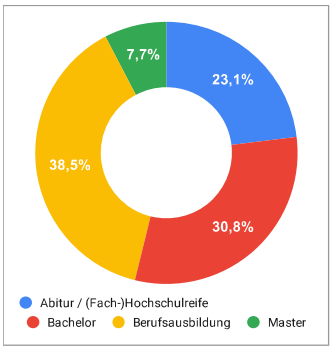
Vielen Dank für die Teilnahme!

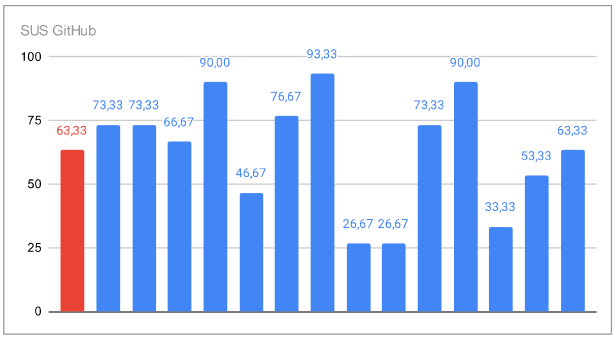
Als Belohnung gibt es einen Keks.

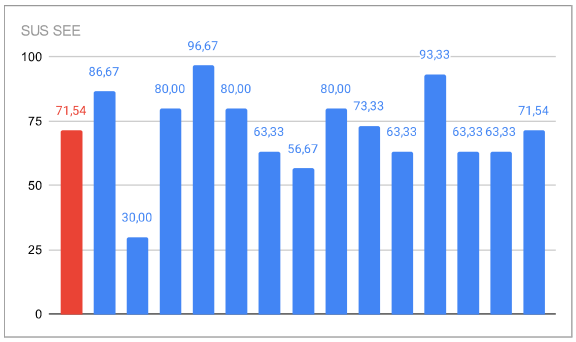
Teilnehmer: 13

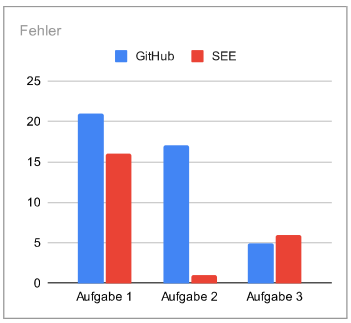


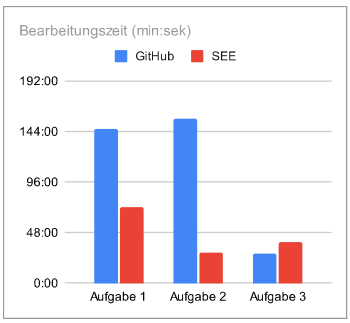


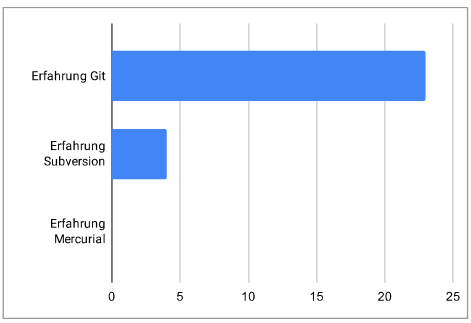












Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Kiel, 30. Dezember 2021

Felix Gaebler