



# **DOKUMENTATION PRAKTIKUM MENSCH-MASCHINE-SYSTEMTECHNIK**

**ENTWURF EINER VISUALISIERUNG FÜR DAS SEMANTISCHE  
REVISIONSVERWALTUNGSSYSTEM R43PLES**

Gruppe 2.3: Lukas Buntkiel, Alexander Lehmann, Miao Zhang, Sven  
Schönfeld, Falk-Jonatan Strube

5. Februar 2015

# **INHALTSVERZEICHNIS**

<b>1</b>	<b>Einleitung</b>	<b>3</b>
1.1	Einordnung . . . . .	4
1.2	Theoriebildung . . . . .	4
1.3	Literatur . . . . .	5
<b>2</b>	<b>Analyse und Entwurf</b>	<b>7</b>
2.1	Analyse . . . . .	8
2.2	Entwurf . . . . .	9
<b>3</b>	<b>Pflichtenheft</b>	<b>12</b>
3.1	Produktfunktionen . . . . .	13
3.1.1	Muss . . . . .	13
3.1.2	Kann . . . . .	13
3.1.3	Soll . . . . .	13
3.2	Qualitätsanforderungen . . . . .	13
3.3	Einschränkungen und Randbedingungen . . . . .	13
3.4	Annahmen und Abhängigkeiten . . . . .	13
<b>4</b>	<b>Gestaltungsentwurf / Implementation</b>	<b>14</b>
4.1	Implementation . . . . .	15
4.2	Ausblick . . . . .	16

# **1 EINLEITUNG**

## 1.1 EINORDNUNG

Hauptbestandteil der Aufgabenstellung ist das Entwerfen einer interaktiven Darstellung der Revisions-Struktur des Versionsverwaltungssystems (Version Control System, VCS) R43ples.

R43ples kann zur Versionsverwaltung von Named Graphs genutzt werden, dem Schlüssel-Bestandteil des Semantic Web [?]. Es verwendet dabei zur Verwaltung der Revisionen wiederum Named Graphs, in denen auch sämtliche, zur Darstellung der Struktur notwendigen, Informationen in Form von Linked Data enthalten sind [?]. R43ples verwendet dabei ein ähnliches Konzept wie klassische Versionsverwaltungssysteme (wie z.B. git [?]) indem es Verzweigungen von Revisionen in Form von Branches sowie das Kennzeichnen spezieller Revisionen mit Tags unterstützt [?].

Der Hauptunterschied zu klassischen VCS liegt also weniger im Konzept der Versionsverwaltung selbst, als in der Anwendung dieses Konzeptes auf einen neuen Typ von Ressource (Named-Graphs). Es kann daher angenommen werden, dass durch andere VCS bereits Lösungen für die graphische Darstellung von Revisionen vorhanden sind, die im Verlauf dieser Arbeit analysiert werden können, um günstige Merkmale herauszuarbeiten.

## 1.2 THEORIEBILDUNG

Die durch den Lehrstuhl gegebene Aufgabenstellung lautet wie folgt:

Ziel dieser Aufgabe ist es, eine gebrauchstaugliche Visualisierung für das semantische Revisionsverwaltungssystem R43ples) zu implementieren, die Revisionsgraphen mit Branches, Merges und Tags darstellen kann. Das Werkzeug soll sich nahtlos in die HTML-Oberfläche des Gesamtsystems einfügen und dem Nutzer Überblicksinformationen über den Revisionsverlauf und Detailinformationen zu den einzelnen Revisionen geben.

Dem entsprechend wurden bei der Theoriebildung folgende Kernanforderungen an das Projekt herausgearbeitet.

**Schnittstelle zu bestehender Software** Zuerst muss eine Schnittstelle zur bestehenden Software hergestellt werden. Das wäre möglich, indem man beispielsweise die turtle Datensätze ausliest die der laufende Server ausgibt. Dadurch hätte man ein separates System das unabhängig vom Revisions-Server turtle Datensätze auslesen und darstellen könnte.

Eine andere Möglichkeit stellt die direkte Integration in das R43ples-Projekt auf github dar, in der Visualisierung direkt im System integriert wäre. Diese Herangehensweise hätte den Vorteil, dass man nicht erst eine turtle Datei parsen müsste, sondern direkt auf die Datensysteme des Servers zugreifen könnte. Durch einen fork des repositorys könnte man an einer Lösung der Aufgabe arbeiten und hätte bei einer guten Lösung einen einfachen Weg die Ergebnisse schnell im Hauptprojekt zu nutzen.

**Graphische Darstellung realisieren** Nachdem die Daten für das System über die Schnittstelle beschafft wurden, ist eine Hauptaufgabe diese als graphische Darstellung zu realisieren.

Eine erste Realisierung liegt bereits als einfache Implementierung mit Viz.js vor. Diese ist allerdings lediglich als Platzhalter zu bewerten.

Der Graph wird anscheinend nach keinen gestalterischen Grundsätzen generiert. Bis auf die chronologische Auflistung der Revisionen von oben nach unten und die Pfeile als optische Hilfestellung wird keine weitere Hilfestellung geboten, die die Erfassung des Graphen erleichtern würde.

Des weiteren ist die Auswahl der Informationen die von den Revisionen angezeigt werden nicht optimal. Sie bietet dem Nutzer nur zwei Informationen, wobei eine davon für den Nutzer uninteressant ist. Es fehlen aber andere, wichtige Informationen wie beispielsweise der Autor oder das Datum.

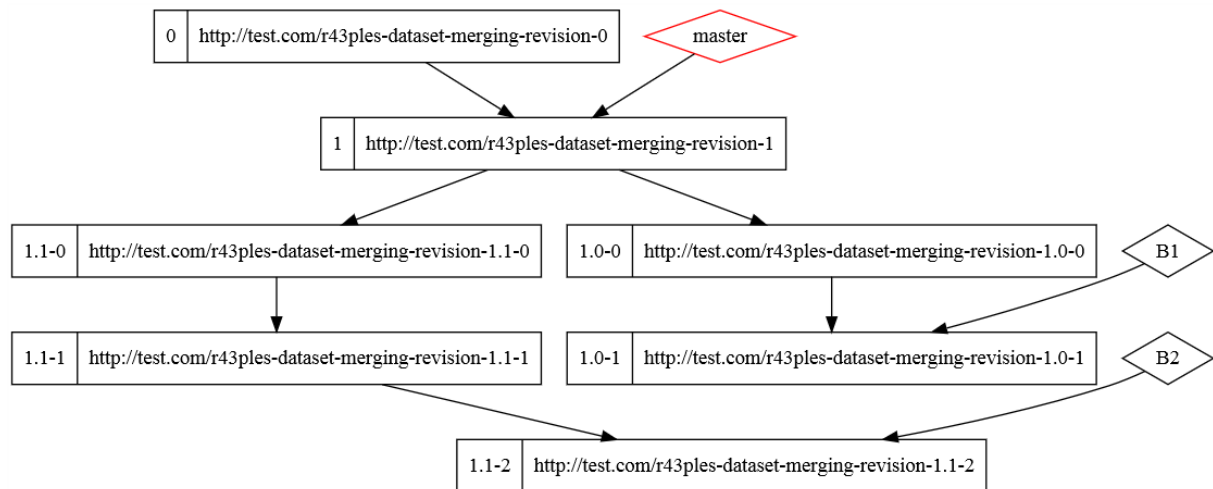


Abbildung 1.1: Vorgegebene Implementierung mit Viz.js

**Überblicks- und Detailsicht** Wenn die Idee der vorhandenen Implementierung beibehalten werden soll, die die Informationen der Revisionen direkt in den Knoten des Graphen darstellt, muss überlegt werden, wie bei großen Revisionsbäumen eine übersichtliche Darstellung erreicht werden kann.

Um einen guten Kompromiss aus Informationsfülle und Übersichtlichkeit zu finden würde sich also vor allem bei vielen Daten ein Trennung des Graphen in eine Überblicks- und eine Detailansicht anbieten. So könnte man in einem Überblicksgraphen die wichtigsten Informationen anzeigen und durch das Navigieren auf eine spezielle Revision eine Detailansicht erreichen, die sämtliche relevanten Informationen darstellt.

## WIE VISUALISIERT MAN EIN SEMANTISCHES VERSIONSVERWALTUNGSSYSTEM?

Wie bereits in der Einordnung erwähnt, kann als Konzept ein ähnliches verwendet werden wie ein klassischen Versionsverwaltungssystemen. Folgende Richtlinien wurden als Gestaltungsgrundsatz festgelegt:

- Übersichtlichkeit – alle wichtigen Informationen sollen mit einem Blick gesehen werden
- Verständlichkeit – alle gesehenen Informationen sollen möglichst schnell im Zusammenhang verstanden werden
- Angemessener Informationsgehalt – Je nach Ansicht sollen angemessen viele Informationen dargestellt werden nach dem Motto: „so wenig wie möglich, so viel wie nötig“

## 1.3 LITERATUR

Nachdem in der Einordnung die grundlegende Literatur für das Thema dargelegt wurde ([?], [?], [?]), finden sich auch noch mehr Informationen in verschiedensten wissenschaftlichen Publikationen:

Der Artikel „Special Graph Representations And Visualization Of Semantic Networks“ [?] steigt genau bei der Fragestellung nach der Visualisierung von semantischen Netzwerken ein und bietet einen grundlegenden Überblick für die Herangehensweise.

Die Masterarbeit „Ein empirischer Vergleich von semistrukturierter und unstrukturierter Konfliktbehandlung in Versionsverwaltungssystemen“ [?] gibt dabei einen tieferen Einblick zu einem Aspekt der Versionsverwaltung, ist bei der Visualisierung aber nur bedingt hilfreich.

In „Approaches to Visualising Linked Data: A Survey“ [?] wird zwar die Visualisierung von Linked Data vorgestellt, allerdings weniger im Bereich der Graphenbildung. Dadurch ist der Artikel für dieses Projekt in dem die Daten durch einen Graphen geordnet darstellen wollen weniger bedeutsam.

Ganz ähnlich bietet „A General Introduction To Graph Visualization Techniques“ [?] einen Überblick über die Visualisierungs-Optionen, wobei hier vermehrt Lösungen durch Graphen vorgestellt werden, was den Artikel etwas interessanter für das Projekt macht.

Nach der Theoriebildung bietet sich an nach Literatur zu den gefundenen Problem- und Fragestellungen zu suchen:

Der Artikel „Eye tracking for visualization evaluation: Reading values on linear versus radial graphs“ [?] führt auf, was die Unterschiede sind, ob man Graphen linear oder radial darstellt. Dabei wird der Schluss präsentiert, dass linear dargestellte Graphen deutlich einfacher auf den ersten Blick zu erfassen sind, weswegen in dem Projekt eine solche Darstellung verwendet werden soll.

Die im Internet zu findende Seite „A successful Git branching model“ [?] zeigt schon auf eine gute Weise, wie man ein Versionsprotokoll darstellen kann und worauf zu achten ist.

Die DIN „Software-Ergonomie für Multimedia-Benutzungsschnittstellen“ DIN EN ISO 14915 [?] [?] [?] gibt entscheidende Richtlinien für die Gestaltung des Projektes im Rahmen einer Benutzerschnittstelle, was vor allem dann von Bedeutung ist, falls die verschiedenen Ansichten in ihren Zoom-Stufen umgesetzt werden soll.

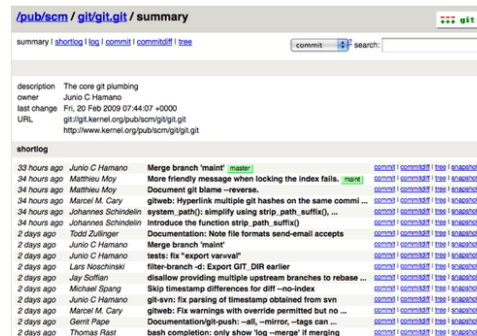
## **2 ANALYSE UND ENTWURF**



Nachdem das Thema eingeordnet, eine Entwurfstheorie entwickelt und dazu Literatur herausgesucht wurde, soll auf dieser Grundlage nun eine Analyse geschehen.

## 2.1 ANALYSE

**Darstellungen anderer Subversion Systeme** Um das Rad nicht neu zu erfinden lohnt es sich die verschiedenen Visualisierungen für die bekannten Versionsverwaltungssysteme GIT und SVN zu betrachten. Dabei werden die Programme GitWeb [?], Gource [?] und Gitready [?] unter die Lupe genommen. Im Folgenden sollen diese kurz vorgestellt werden.



Time	Author	Commit Message	Options
33 hours ago	Junio C Hamano	Merge branch 'maint'	[options]
34 hours ago	Mathieu Moy	More friendly message when locking the index fails.	[options]
34 hours ago	Mathieu Moy	Document git blame -reres.	[options]
34 hours ago	Marcel M. Cary	gitweb: Hyperlink multiple git hashes on the same commit ...	[options]
34 hours ago	Johannes Schindelin	system_path(): simplify using strip_path_suffix(), ...	[options]
34 hours ago	Johannes Schindelin	Introduce the function strip_path_suffix()	[options]
2 days ago	Todd Zullinger	Documentation: Note file formats send-email accepts	[options]
2 days ago	Junio C Hamano	Merge branch 'maint'	[options]
2 days ago	Junio C Hamano	tests: fix "export varvaf"	[options]
2 days ago	Lars Noschinski	filter-branch -d: Export GIT_DIR earlier	[options]
2 days ago	Jay Soffian	Disallow providing multiple upstream branches to rebase ...	[options]
2 days ago	Michael Spang	Skip timestamp differences for diff --no-index	[options]
2 days ago	Junio C Hamano	git-svn: fix parsing of timestamp obtained from svn	[options]
2 days ago	Marcel M. Cary	gitweb: Fix warnings with override permitted but no ...	[options]
2 days ago	Gerrit Paep	Documentation/git-push: --all, --mirror, --tags can ...	[options]
2 days ago	Thomas Rast	bash completion: only show 'log -m' if merging	[options]

Abbildung 2.1: gitweb Visualisierung

GitWeb ist eine simple, webbasierte Visualisierung für die Versionsverwaltung GIT. Sie ist im Standardinstallationsumfang von GIT enthalten und stellt die grundlegendste Visualisierungsmöglichkeit für GIT dar. Die Informationen werden dabei tabellarisch in absteigender Reihenfolge ihrer Aktualität dargestellt. In der ersten Spalte wird die Zeit seit dem zugehörigen Commit angegeben. Die zweite Spalte gibt den Namen des Nutzers an, der den jeweiligen Commit durchgeführt hat. In der dritten Spalte wird die Commit-Message wiedergegeben und in der vierten Spalte stehen einige Optionen zur Verfügung (Abb. 2.1). Die Visualisierung mittels GitWeb ist klar, kompakt und verständlich, allerdings ist die Zugehörigkeit zu einzelnen Branches nicht klar erkennbar, da lediglich Commits zum Master-Branch als solche markiert werden.

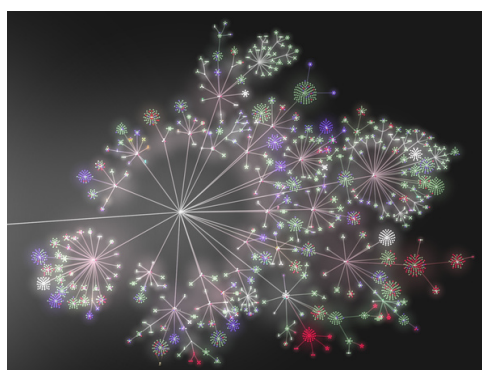


Abbildung 2.2: Visualisierung durch gource

Anders als GitWeb kann das Programm Gource nicht nur mit GIT, sondern auch mit SVN, CVS und anderen verwendet werden. Gource repräsentiert das Dateiverzeichnis als animierte Baumstruktur, bei der das Wurzelverzeichnis im Zentrum liegt (Abb. 2.2). Die Blätter des Baumes repräsentieren Dateien. Avatare, die die Entwickler repräsentieren, arbeiten dabei in der selben Reihenfolge wie in der Realität an den einzelnen Dateien und Verzeichnissen, fügen neue hinzu oder entfernen überflüssige. Die Animation kann dabei jederzeit unterbrochen

werden. Es werden weder Commit-Messages noch Änderungsdaten angezeigt, der Informationsgehalt der Animation hält sich also in Grenzen. Hierdurch ist Gource eher als Spielerei zur Unterhaltung der Entwickler oder zu Präsentationszwecken zu verstehen, denn als Werkzeug zur Überwachung der einzelnen Versionen.

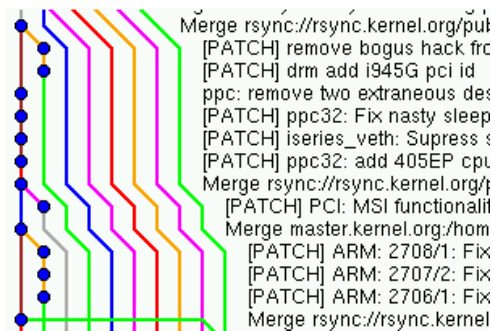


Abbildung 2.3: gitready Visualisierung

Gitready ähnelt eher GitWeb als Gource. Hier werden die einzelnen Branches als verschiedenfarbige Linien, die parallel nebeneinander auf der linken Seite des Bildschirms verlaufen, dargestellt (Abb. 2.3). Kreise auf den Linien repräsentieren einzelne Commits. Da jeder Commit eine eigene Zeile zugewiesen bekommt ist die Reihenfolge klar ersichtlich. Rechts neben den einzelnen Linien werden die jeweiligen Commit-Messages angezeigt. Auf Grund der Abschnittsweise großen Distanz zwischen Knoten und Message ist das Gesetz der Nähe nicht immer gegeben. Dadurch kann es zu fehlerhaften Zuweisungen zwischen Commit und Message kommen. Allerdings bietet die Darstellung mittels Gitready einen guten Kompromiss zwischen Übersichtlichkeit und Informationsgehalt der betrachteten Visualisierungsprogramme.

## 2.2 ENTWURF

Bei dem Entwurf stellt sich zunächst die Frage, ob den in der Analyse betrachteten Systemen lediglich nachgeahmt werden soll oder ob neue Ideen einfließen dürfen. Obgleich die Analyse ergeben hat, dass die Visualisierung wie sie von gitready genutzt wird unsere geforderten Qualitäten von „Übersichtlichkeit“, „Verständlichkeit“ und „angemessenem Informationsgehalt“ ausreichend erfüllt, wurde zunächst ein neuer Weg eingeschlagen. Dieser sollte angelehnt an moderne graphische Oberflächen optisch ansprechend sein und dadurch die geforderte Übersichtlich- und Verständlichkeit erfüllen.

**Erster Entwurf** Bei dieser Überlegung ist ein erster Entwurf heraus gekommen, der sich auf folgende Stärken konzentriert (Abb. 2.4):

Vor allem die Übersichtlichkeit und Verständlichkeit soll durch breit gestaltete Führungslinien zwischen Commits (bzw. Revisionen bei r43ples) sichergestellt werden. Durch die Einordnung in „Spalten“ sollen die Branches eindeutig erkannt werden. Tags sollen zur genauen Zuordnung als wichtige Information direkt am Commit dran stehen.

Als zweite Ebene sollte entsprechend der Idee in der Theoriebildung (Paragraph „Überblicks- und Detailansicht“ in Abschnitt 1.2) eine Detailansicht angezeigt werden. Diese soll sich nach einem Klick auf den entsprechenden Commit öffnet, den Hintergrund abdunkeln und alle wichtigen Details darstellen (Abb. 2.5).

**Probleme beim Entwurf** Durch den Fokus auf die Benutzerführung durch große Commit-Blätter und breite Zweige ergibt sich allerdings das Problem des benötigten Platzes. Vor allem

bei größeren Graphen, wenn nicht mehr der gesamte Graph auf eine Seite passt (oder das Browserfenster verkleinert wurde), würde sich die Übersichtlichkeit ins Gegenteil umkehren.

Zudem wurde aus Gründen des Platzes auch entschieden, dass nicht jedes Commit eine eigene „Zeile“ erhalten kann, wodurch allerdings die chronologische Abfolge nicht gegeben ist und diese Information nicht auf dem ersten Blick zu erfassen ist.

Aufgrund dieser Nachteile wurde sich in der Implementation des Gestaltungsentwurfs dafür entschieden die in der Analyse gelobte Darstellung von GitReady (Abb. 2.3) nachzuahmen. In dieser Darstellung begegnet man den genannten Problemen nicht, hat aber eine vergleichbare Übersicht.

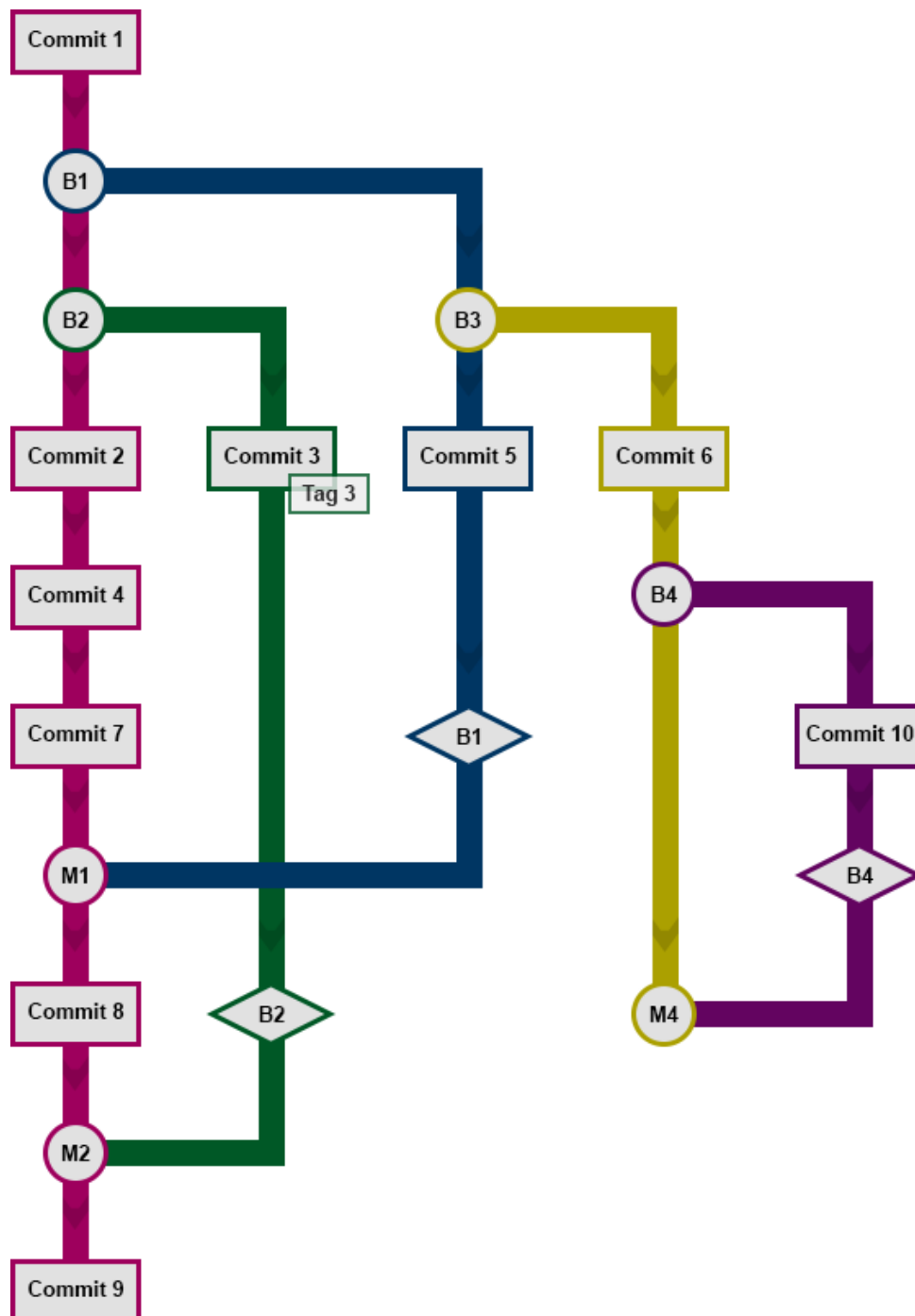


Abbildung 2.4: Skizze für ersten Entwurf



Abbildung 2.5: Skizze für die Detailansicht des ersten Entwurfs

## **3 PFLICHTENHEFT**

Im Rahmen des Praktikums erfolgte eine Erstellung eines Pflichtenhefts welches hier nachzulesen ist.

Die Erstellung des Pflichtenhefts erfolgt entsprechend der Empfehlungen des IEEE Guide for Software Requirements Specifications (SRS) [?]. Die dort empfohlene Struktur wird in deutscher Übersetzung nach Balzert [?] stichpunktartig knapp übernommen.

## **3.1 PRODUKTFUNKTIONEN**

### **3.1.1 MUSS**

- Daten (bspw. aus turtle-Datensatz) auslesen (zum weiterverarbeiten)
- Daten werden graphisch abgebildet

### **3.1.2 KANN**

- Daten werden in in unterschiedlichen Darstellungsebenen dargestellt (Gesamt- und Detailübersicht)
- Einzelansicht für die Details von Revisions
- Daten werden tabellarisch-strukturiert abgebildet

### **3.1.3 SOLL**

- Zweckmäßige Übergangsanimationen zwischen Darstellungsansichten

## **3.2 QUALITÄTSANFORDERUNGEN**

- Daten werden unverfälscht abgebildet
- Durch Informationsreduzierung (auf das nötigste) wird ein höhere Übersichtlichkeit erreicht (Minimierung der Darstellung von merges und revisions durch Unterteilung in Detailansichten)

## **3.3 EINSCHRÄNKUNGEN UND RANDBEDINGUNGEN**

- Läuft auf neuerem Firefox und Chrome

## **3.4 ANNAHMEN UND ABHÄNGIGKEITEN**

- Nutzer ist mit r43ples und dem semantic web vertraut

## **4 GESTALTUNGSENTWURF / IMPLEMENTATION**

In der Analyse nach dem ersten Entwurf wurde die Entscheidung für eine Darstellung ähnlich der von GitReady (Abb. 2.3) getroffen.

Der erste Schritt zur Implementation war das github-Projekt zu forken und alle benötigten Dienste (Virtuoso) zu installieren um es in der eigenen Entwicklungsumgebung (eclipse) lauffähig zu machen.

Im Folgenden sollen einige wichtige (Design-) Entscheidungen aufgeführt werden, die bei der Implementation entstanden sind.

## 4.1 IMPLEMENTATION

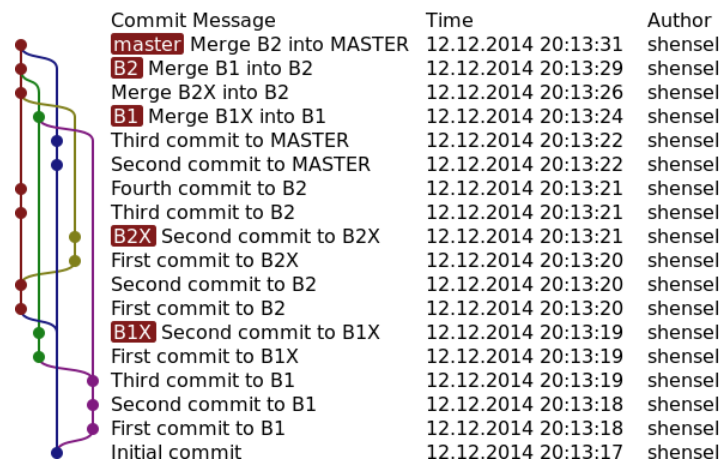


Abbildung 4.1: Das erste Ergebnis der Implementation

**Framework** Zuerst wurde entschieden welches Framework verwendet werden soll, um die Visualisierung darzustellen. Die Ausgabe sollte als svg-Datei erfolgen, da dieses eine flexible Einbindung in die Weboberfläche bietet.

Dem entsprechend wurde das Apache Batik Framework genutzt, da dieses die Java Graphics2D API implementiert und somit Code zum Zeichnen (beispielsweise auf einem AWT Canvas) wiederverwendet werden kann.

**Commitgraph** Der Commitgraph soll durch eine neue Datenstruktur des „StructuredTree“s dargestellt werden, um die Arbeit mit Commits zu vereinfachen. Dieser beinhaltet einen doppelt verknüpften Graphen der Commits, eine Liste der Branches und eine Liste der Tags. Beide Listen haben dabei jeweils Referenzen auf die Commits.

Hierdurch wird die Logik zum Generieren der Datenstruktur aus dem SPARQL Result abgekapselt.

**Informationsdarstellung** Entsprechend der Darstellung von GitReady sollen der Commit-Graph und die Informationen zu den Commits nebeneinander dargestellt werden. Dafür ist es wichtig, dass pro Zeile im Commit-Graphen nur ein Commit dargestellt wird, um die entsprechenden Informationen daneben anzeigen zu können.

Mit dieser „ein Commit pro Zeile“-Regel erzeugt man eine bessere Übersichtlichkeit für den Anwender, da die Beschriftungen nicht die Ausmaße des Graphen vergrößern. Somit bleibt der Graph kompakt und kann mit wenigen Blicken erfasst werden.



Des weiteren ist nur eine geradlinige horizontale Augenbewegung nötig um von Commit-„Punkt“ im Graphen zur Beschriftung zu gelangen. Somit hat man trotz der Informations-Trennung eine hohe Lesegeschwindigkeit (siehe Literatur [?]).

Der Commit-Graph verläuft in Spalten, wobei pro Spalte ein Branch dargestellt wird. Diese eindeutige Zuordnung der Spalten zum Branch ist ebenfalls ein Gewinn für die Übersichtlichkeit, die durch die Einfärbung der Branches (=Spalten) weiter erhöht wird. Bei der Auswahl der Farben für die Branches wurde außerdem darauf geachtet keine aggressiven oder grellen Farben zu verwenden, um einer Ermüdung des Auges vorzubeugen.

**Überarbeitung für weniger Redundanz** Nach einer ersten Implementationsphase wurde das Ergebnis (Abb. 4.1) bewertet. Dabei wurde festgestellt, dass es bei der Informationsdarstellung, vor allem bei der Datumsdarstellung viel Redundanz gibt. Für eine besser Übersicht wurde deshalb entschieden die Zeit-Angabe nur noch Tag-genau darzustellen (da die genau Zeit nicht so wichtig ist und bei direkt aufeinanderfolgenden Commits mehr das „welcher vor welchem“ und weniger das „welcher genau wann vor welchem“ zählt).

Herausgekommen ist das in den folgenden Abbildungen beispielhaft dargestellte Ergebnis:

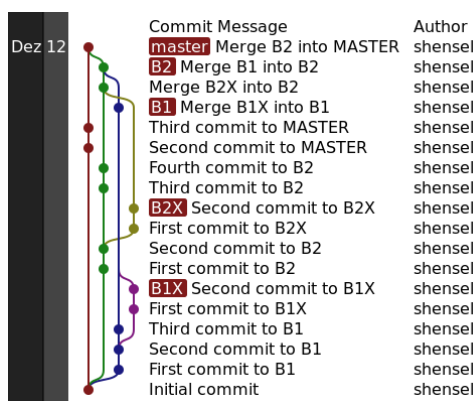


Abbildung 4.2: Bsp. Merge-Graph

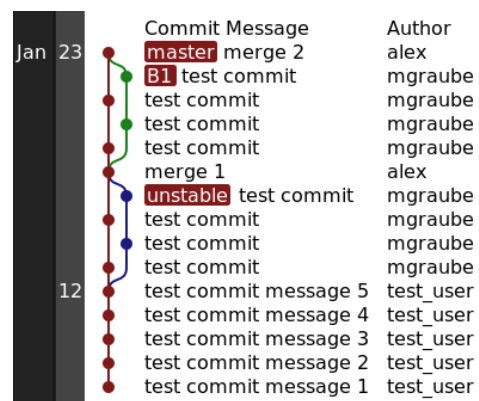


Abbildung 4.3: Bsp. Test Datensatz

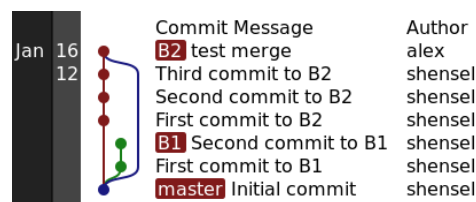


Abbildung 4.4: Bsp. zweiter Merge-Graph

Die vollständige Implementation ist im github als fork des Original-Projekts unter <https://github.com/flugtiger/r43ples> zu finden.

## 4.2 AUSBLICK

Die vorhandene Implementation hat unsere Ziele für das Projekt erreicht: Der Graph für die Darstellung der Revisionen ist Übersichtlich, Verständlich und bietet einen angemessenen Informationsgehalt.

Nach statistischer Erfassung des Nutzerverhaltens und Nutzerbefragungen könnte man noch evaluieren was an der Darstellungsform noch optimiert werden kann.

Des weiteren könnte die Darstellung beispielsweise durch mouse-over Fenster bei Commits noch mit mehr Informationen gefüttert werden.

Sollte sich die Lösung für den Kunden als eine gute bzw. die beste der zur Wahl stehenden Lösungen herausstellen, wäre außerdem noch eine nahtlosere Einbindung in die vorhandene graphische Benutzeroberfläche erstrebenswert.

## ANHANG

Hier sind noch die Programm-Ablauf-Pläne zu finden, die die Funktion der Implementation verdeutlichen:

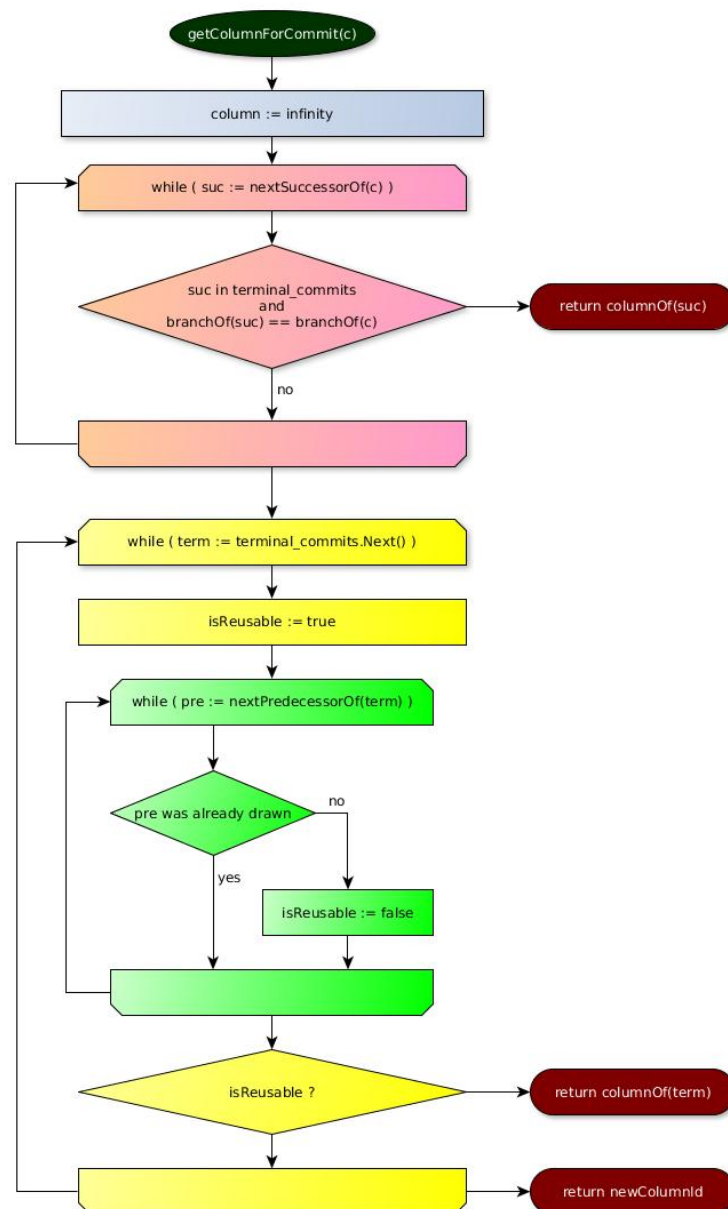


Abbildung 4.5: Abschnitt des PAPs für die Column-generierung

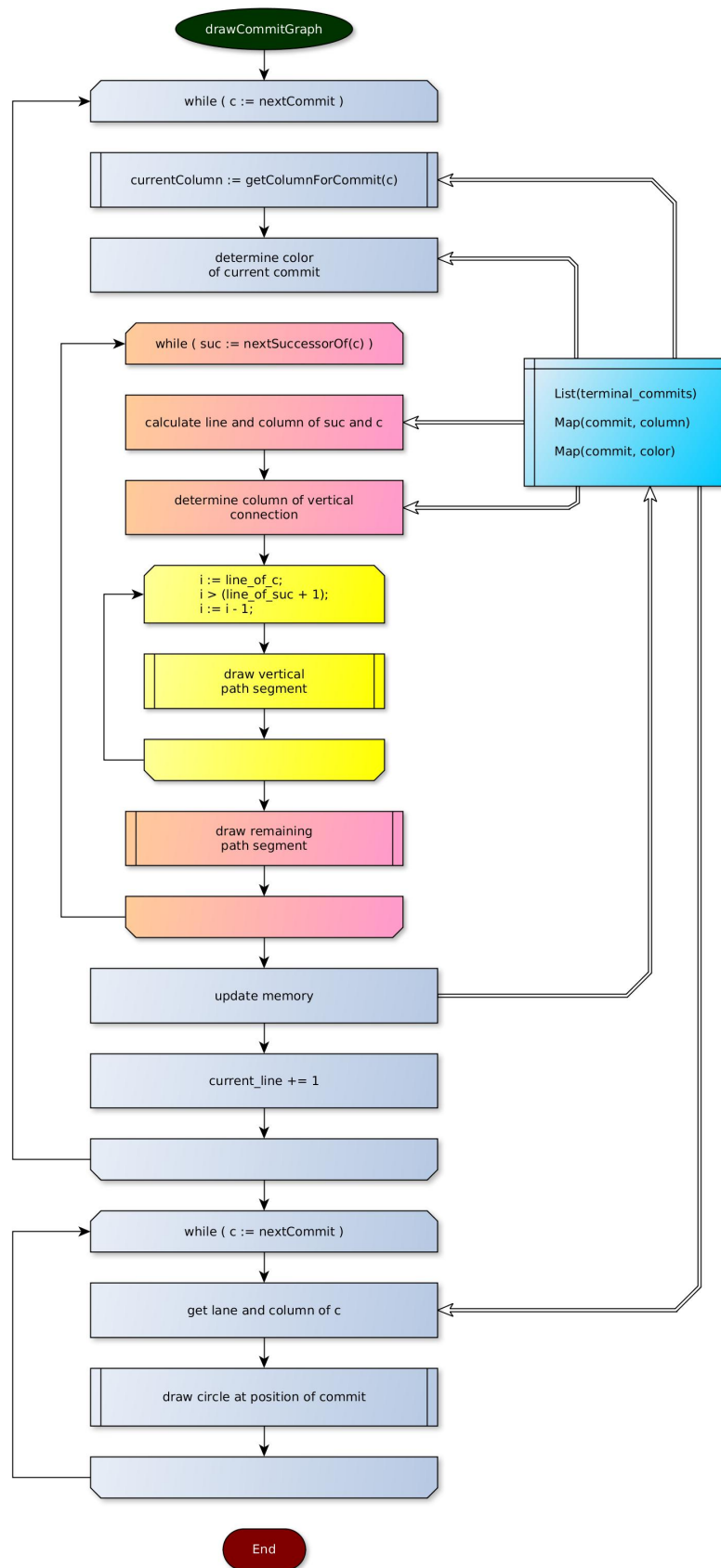


Abbildung 4.6: PAP der Darstellung des Commitgraphs