Anwendungen der Softwarearchäologie
Einführung und Grundlagen

Bastian Venthur

Freie Universität Berlin
Institut für Informatik

2005-06-11

Übersicht

Motivation

- Traditionelles Problem der Datenbeschaffung in der empirischen Forschung
- Entwicklung von F/OSS im Gegensatz, stellt riesige und frei zugängliche Datenmengen zur Verfügung
- Allein auf sourceforge zur Zeit über 101.000 Projekte

Gesucht wird eine Möglichkeit *Fragen* an das System zu stellen, wie:

- Welche Klassen/Methoden/Pakete bilden ein funktionales Konzept?
- Welche Art von Bugs/Defekten tritt am häufigsten auf?
- Gibt es Metriken, mit denen man die Wahrscheinlichkeit von Defekten in bestimmten Code-Teilen vorhersagen kann?
- Welcher Autor hat welches Spezialwissen?
- In welcher Phase steckt das Projekt?
- . . .

Motivation

- Traditionelles Problem der Datenbeschaffung in der empirischen Forschung
- Entwicklung von F/OSS im Gegensatz, stellt riesige und frei zugängliche Datenmengen zur Verfügung
- Allein auf sourceforge zur Zeit über 101.000 Projekte

Gesucht wird eine Möglichkeit *Fragen* an das System zu stellen, wie:

- Welche Klassen/Methoden/Pakete bilden ein funktionales Konzept?
- Welche Art von Bugs/Defekten tritt am häufigsten auf?
- Gibt es Metriken, mit denen man die Wahrscheinlichkeit von Defekten in bestimmten Code-Teilen vorhersagen kann?
- Welcher Autor hat welches Spezialwissen?
- In welcher Phase steckt das Projekt?
- . . .

Motivation (cont)

Grundsätzliche Probleme:

- Daten liegen nicht in automatisch auswertbarer Form bereit
- Die Gesamtheit der Daten Umfasst oft: SCS, Mailinglisten, Bugtrackingsystem, Hompage

Ergo:

Per Hand lassen sich unmöglich alle Daten effektiv nutzen.

Große Frage:

Wie kann ich diese Datenmengen effektiv nutzen?

Motivation (cont)

Grundsätzliche Probleme:

- Daten liegen nicht in automatisch auswertbarer Form bereit
- Die Gesamtheit der Daten Umfasst oft: SCS, Mailinglisten, Bugtrackingsystem, Hompage

Ergo:

Per Hand lassen sich unmöglich alle Daten effektiv nutzen.

Große Frage:

Wie kann ich diese Datenmengen effektiv nutzen?

Übersicht

Begrifflichkeiten

Für source control systeme gibt es verschiedene synonyme Begriffe:

- source control system (SCS)
- version control system (VCS)
- revision control system (RCS)
- ...

zentralisiert vs dezentralisiert

Wir unterscheiden grudsätzlich zwei Arten von SCS:

- zentralisierte SCS (CVS, subversion)
- dezentrale SCS (GNU-Arch, Bitkeeper)

Die Grundoperationen sind aber prinzipiell gleich.

Repository

- engl. Behälter, Aufbewahrungsorg
- großes Archiv aller (Re)Visionen eines Projektes
- verschiedene SCS setzen sehr unterschieldiche Dateiformate oder gar eine Datenbank ein
- üblicherweise nur diffs zwischen zwei aufeinanderfolgenden Versionen

Arbeitskopie, checkout, checkin

Arbeitskopie

lokale Kopie einer bestimmten (meist die letzte) Version des Projektes

checkout/update

Vorgang, bei dem eine Arbeitskopie erstellt/aktualisiert wird.

checkin/commit

- Vorgang, bei dem Repository auf Arbeitskopie aktualisiert wird
- Meist mit Möglichkeit einen Kommentar abzusetzen

Arbeitskopie, checkout, checkin

Arbeitskopie

lokale Kopie einer bestimmten (meist die letzte) Version des Projektes

checkout/update

Vorgang, bei dem eine Arbeitskopie erstellt/aktualisiert wird.

checkin/commit

- Vorgang, bei dem Repository auf Arbeitskopie aktualisiert wird
- Meist mit Möglichkeit einen Kommentar abzusetzen

Arbeitskopie, checkout, checkin

Arbeitskopie

lokale Kopie einer bestimmten (meist die letzte) Version des Projektes

checkout/update

Vorgang, bei dem eine Arbeitskopie erstellt/aktualisiert wird.

checkin/commit

- Vorgang, bei dem Repository auf Arbeitskopie aktualisiert wird
- Meist mit Möglichkeit einen Kommentar abzusetzen

diff

Zeigt den Unterschied zwischen zwei (frei wählbaren) Versionen auf.

Ein Beispiel:

```
bventhur@phoibe:~/robocup/observer/src/decoder$ svn diff decoder.cpp -r 2370
Index: decoder.cpp
===================================================================
--- decoder.cpp (Revision 2370)
+++ decoder.cpp (Arbeitskopie)
@@ -70,9 +70,9 @@
     Frame* frame = new Frame();
     size_t offset=0;
     float posx,posy;
-    float timestamp;
-    memcpy(&timestamp,buffer+offset,sizeof(float));
-    offset += sizeof(float);
+    int timestamp;
+    memcpy(&timestamp,buffer+offset,sizeof(int));
+    offset += sizeof(int);
     memcpy(&posx,buffer+offset,sizeof(float));
     offset += sizeof(float);
     memcpy(&posy,buffer+offset,sizeof(float));
```

Übersicht

Was ist MSR?

MSR

engl. Mining of Software Repositories

- MSR bezeichnet das *Graben* in SCS nach Informationen
- Wonach man gräbt kommt auf die Fragen an
- Verschiedene Fragen begünstigen verschiedene Ansätze für MSR

Was ist MSR?

MSR

engl. Mining of Software Repositories

- MSR bezeichnet das *Graben* in SCS nach Informationen
- Wonach man gräbt kommt auf die Fragen an
- Verschiedene Fragen begünstigen verschiedene Ansätze für MSR

Annotations?

Zeilenweise Ausgabe einer Datei, zusammen mit:

- Revisionsnummer
- Autor
- Zeitstempel

pro Zeile

Annotations? (cont)

Ein Beispiel:

```
bventhur@phoibe:~/robocup/venthur/observer/src/decoder$ svn blame decoder.cpp
  ...
  2283    venthur
  2283    venthur     Frame* frame = new Frame();
  2283    venthur     size_t offset=0;
  2283    venthur     float posx,posy;
  2504    venthur     int timestamp;
  2504    venthur     memcpy(&timestamp,buffer+offset,sizeof(int));
  2504    venthur     offset += sizeof(int);
  2283    venthur     memcpy(&posx,buffer+offset,sizeof(float));
  2283    venthur     offset += sizeof(float);
  2283    venthur     memcpy(&posy,buffer+offset,sizeof(float));
  2283    venthur     offset += sizeof(float);
  2283    venthur
  2283    venthur     frame->addBall( Ball(posx, posy) );
  2370    venthur     frame->setTimeStamp(timestamp);
  ...
```

Prinzip

- Merke alle Revisionen einer Klasse/Datei
- Vergleiche die Revisionen mit Revisionen anderer Dateien

→ Klassen mit gleichen sichtbaren Revisionen wurden zusammen geändert

Das lässt sich noch verfeinern:

- Beziehe die Zeiten mit ein
- Revisionen innerhalb eines Zeitfensters (z.B. 5min) werden als *eine* Revision betrachtet

Prinzip

- Merke alle Revisionen einer Klasse/Datei
- Vergleiche die Revisionen mit Revisionen anderer Dateien

→ Klassen mit gleichen sichtbaren Revisionen wurden zusammen geändert

Das lässt sich noch verfeinern:

- Beziehe die Zeiten mit ein
- Revisionen innerhalb eines Zeitfensters (z.B. 5min) werden als *eine* Revision betrachtet

Welche Fragen können beantwortet werden?

- Welche Klassen ändern sich zusammen?
- Wie oft wurde eine bestimmte Klasse geändert?
- Wieviele Änderungen traten in einem Subsystem (z.B. Verzeichnis) auf?
- Wieviele Änderungen traten zwischen Subsystemen auf?
- Welcher Zusammenhang besteht zwischen einem Autor und einer Klasse?

Was kann Data Mining?

- U.A.: Assoziationsregeln erstellen
- z.B. A,B $\rightarrow$ C
- Dazu müssen die Daten in eine DM-verträgliche Form gebracht werden: (filename, type, identifier)
- Granularität von Datei- über Klassen- zu Funktions- und Variablenebene

Ergebnis:

- Versteckte Abhängigkeiten können gut aufgedeckt werden
- Typische Frage: Welche Änderungen traten zusammen auf?

Was kann Data Mining?

- U.A.: Assoziationsregeln erstellen
- z.B. A,B $\rightarrow$ C
- Dazu müssen die Daten in eine DM-verträgliche Form gebracht werden: (filename, type, identifier)
- Granularität von Datei- über Klassen- zu Funktions- und Variablenebene

Ergebnis:

- Versteckte Abhängigkeiten können gut aufgedeckt werden
- Typische Frage: Welche Änderungen traten zusammen auf?

Funktionsweise:

- Im Prinzip wie MSR via Annotations
- nur werden diesmal die Zeilen lexigraphisch analysiert d.h. Heuristiken *erkennen* bestimmte Änderungen

Frage:

- Erkennung und Vorhersage von Änderungen

Funktionsweise:

- Im Prinzip wie MSR via Annotations
- nur werden diesmal die Zeilen lexigraphisch analysiert d.h. Heuristiken *erkennen* bestimmte Änderungen

Frage:

- Erkennung und Vorhersage von Änderungen

Funktionsweise:

- Untersuche diff jeder Revision
- Erstelle ASG
- wende differencing-Algorithmus an

differencing-Algorithmus

- Wird an je zwei ASG angewendet
- gibt zurück was der Unterschied zwischen den beiden ist
- z.B. Knoten hinzu, entfernt, verschoben, geändert

Funktionsweise:

- Untersuche diff jeder Revision
- Erstelle ASG
- wende differencing-Algorithmus an

differencing-Algorithmus

- Wird an je zwei ASG angewendet
- gibt zurück was der Unterschied zwischen den beiden ist
- z.B. Knoten hinzu, entfernt, verschoben, geändert

Auswertung der Ausgabe des differencing-Algorithmus

- Die Knoten im ASG entsprechen syntaktischen Elementen im Code
- Die Ausgabe ist also in der Art: If-Anweisung geändert/entfernt/hinzu, Variable umbenannt usw.

Typische Fragen:

- Wieviele Funktionen oder Funktionsaufrufe wurden eingefügt?
- Wieviele if-Statements wurden verändert?
- Und viele andere gleichartige Fragen. . .

Auswertung der Ausgabe des differencing-Algorithmus

- Die Knoten im ASG entsprechen syntaktischen Elementen im Code
- Die Ausgabe ist also in der Art: If-Anweisung geändert/entfernt/hinzu, Variable umbenannt usw.

Typische Fragen:

- Wieviele Funktionen oder Funktionsaufrufe wurden eingefügt?
- Wieviele if-Statements wurden verändert?
- Und viele andere gleichartige Fragen. . .

Übersicht

Motivation

Hauptsächlich

- eine einfache Möglichkeit ein SCS zu befragen
- eine Möglichkeit historische (davor, danach) Zusammenhänge zu erfragen

Nebenbei

Ein Standard um SCS-Informationen uniform zu speichern und abzufragen

Also was ist SCQL?

- source control query language
- Prädikatenlogik erster Stufe ++

Motivation

Hauptsächlich

- eine einfache Möglichkeit ein SCS zu befragen
- eine Möglichkeit historische (davor, danach) Zusammenhänge zu erfragen

Nebenbei

Ein Standard um SCS-Informationen uniform zu speichern und abzufragen

Also was ist SCQL?

- source control query language
- Prädikatenlogik erster Stufe ++

Lösungsweg

- Erstelle ein Modell das die Informationen eines SCS repräsentiert (gerichteter Graph)
- Erstelle eine Sprache, mit der man Fragen an dieses Modell richten kann (Präd'Logik erster Stufe)

Model als gerichteter Graph

Formal ist das Modell also als gerichteter Graph G eines SCS mit $G = (V, E)$ definiert. Wobei die Kanten und Knoten folgendermaßen definiert sind:

$$V = MR \cup File \cup Author \cup Revision \tag{1}$$

$$E = (v_1 \in MR, v_2 \in MR) \cup (v_1 \in MR, v_2 \in Revision) \tag{2}$$

$$\cup (v_1 \in MR, v_2 \in Author) \cup (v_1 \in Revision, v_2 \in Revision) \tag{3}$$

$$\cup (v_1 \in Revision, v_2 \in Author) \cup (v_1 \in Revision, v_2 \in File) \tag{4}$$

Gerichteter Graph

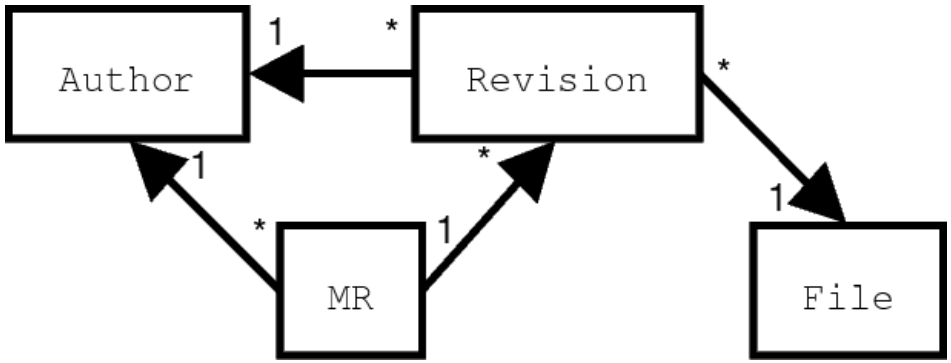
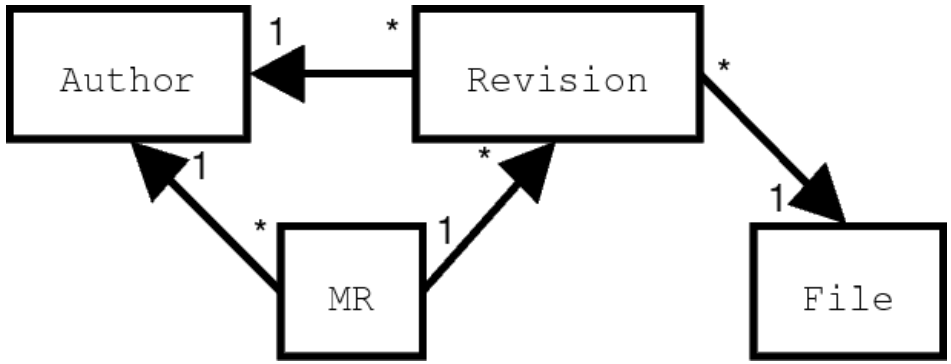


Abbildung: Kardinalitäten und Richtungen der Knoten im Model

Bestehend aus:

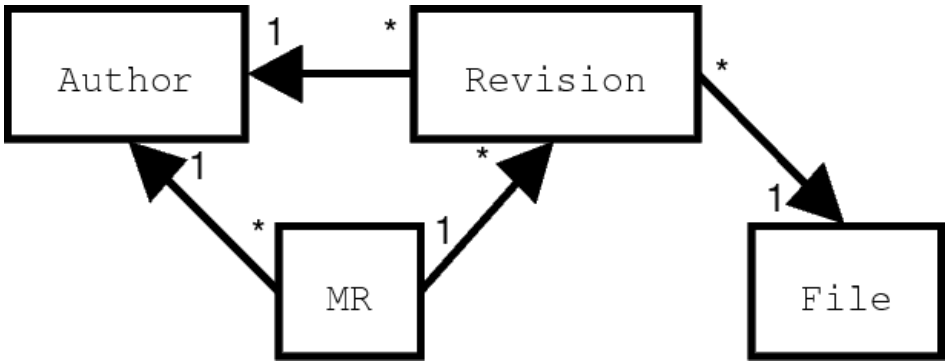
- MRs
- Revisions
- Files
- Authors

modification requests (MRs)



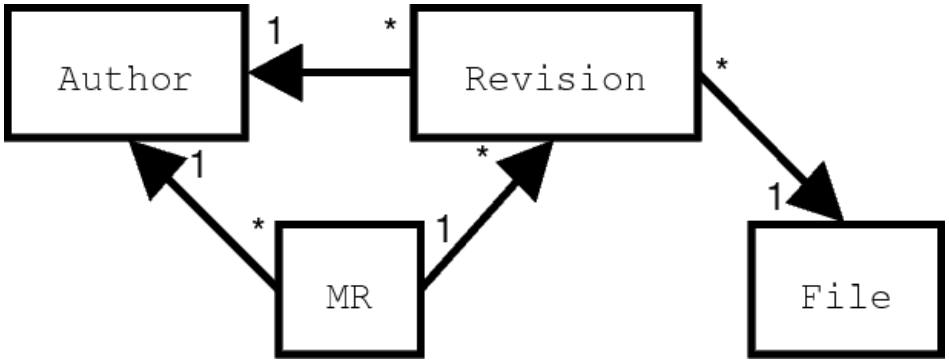
- haben Attribute wie: Kommentar, Zeitstempel, eindeutige ID
- Ausgehende Kanten: 1 zum nächsten MR, 1 zum Autor und mind. 1 zur Revision

Revisions



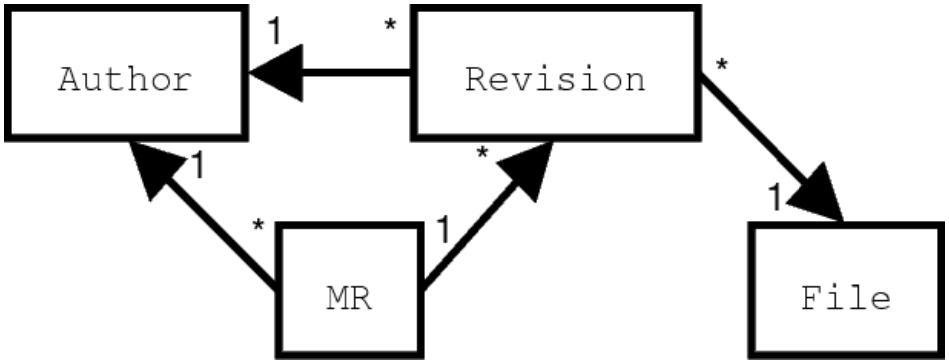
- Attribute: eindeutige Zeitstempel, ID, diffs, veränderte Zeilen, ...
- Ausgehende Kanten: 1 zum Autor, 1 zur Datei, 1 zu jeder direkt nachfolgenden Revision (bei branches)
- Bei Merge: zwei Revisionen zeigen auf ihren gemeinsamen Nachfahren

Files



- Attribute: Pfad, Dateiname, Verzeichniss, eindeutiger voller Pfadname, eindeutigen Zeitstempel
- Dateien sind durch Revisionen verbunden

Authors



- Attribute: ID, Name, email
- Zeitlich mit ihrer ersten Revision assoziiert
- Nur ein Autor pro MR und Revison

Wie bekommt man den Graphen aus SCS?

1. Jede Datei wird ein Knoten der Menge File
2. Jeder Autor wird ein Knoten der Menge Author
3. Jede Revision wird ein Knoten der Menge Revision. Ordne Revisionen eineindeutige Zeitstempel zu und verbinde jede Revision zu ihrer entsprechenden Datei und ihrem Autor
4. Erstelle Knoten für jedes MR. Das MR erbt den Zeitstempel seiner ersten Dateirevision. Verknüpfe MR zum entsprechenden Autor (Vermutlich Fehler im Paper: Man muss noch die MRs mit den Revisionen verbinden)
5. Verbinde alle MRs zu ihren jeweils nächsten MRs (nacht Zeitstempel), sofern sie existieren.
6. Für jede Datei: Verbinde jede Revision dieser Datei zur jeweils nächsten dieser Datei. Wenn gebranched wurde, werden nur Revsionen einer Brach auf diese Weise verbunden. Branch- und Mergepoints werden verbunden.

Wie bekommt man den Graphen aus SCS?

1. Jede Datei wird ein Knoten der Menge File
2. Jeder Autor wird ein Knoten der Menge Author
3. Jede Revision wird ein Knoten der Menge Revision. Ordne Revisionen eineindeutige Zeitstempel zu und verbinde jede Revision zu ihrer entsprechenden Datei und ihrem Autor
4. Erstelle Knoten für jedes MR. Das MR erbt den Zeitstempel seiner ersten Dateirevision. Verknüpfe MR zum entsprechenden Autor (Vermutlich Fehler im Paper: Man muss noch die MRs mit den Revisionen verbinden)
5. Verbinde alle MRs zu ihren jeweils nächsten MRs (nacht Zeitstempel), sofern sie existieren.
6. Für jede Datei: Verbinde jede Revision dieser Datei zur jeweils nächsten dieser Datei. Wenn gebranched wurde, werden nur Revsionen einer Brach auf diese Weise verbunden. Branch- und Mergepoints werden verbunden.

Wie bekommt man den Graphen aus SCS?

1. Jede Datei wird ein Knoten der Menge File
2. Jeder Autor wird ein Knoten der Menge Author
3. Jede Revision wird ein Knoten der Menge Revision. Ordne Revisionen eineindeutige Zeitstempel zu und verbinde jede Revision zu ihrer entsprechenden Datei und ihrem Autor
4. Erstelle Knoten für jedes MR. Das MR erbt den Zeitstempel seiner ersten Dateirevision. Verknüpfe MR zum entsprechenden Autor (Vermutlich Fehler im Paper: Man muss noch die MRs mit den Revisionen verbinden)
5. Verbinde alle MRs zu ihren jeweils nächsten MRs (nacht Zeitstempel), sofern sie existieren.
6. Für jede Datei: Verbinde jede Revision dieser Datei zur jeweils nächsten dieser Datei. Wenn gebranched wurde, werden nur Revsionen einer Brach auf diese Weise verbunden. Branch- und Mergepoints werden verbunden.

Wie bekommt man den Graphen aus SCS?

1. Jede Datei wird ein Knoten der Menge File
2. Jeder Autor wird ein Knoten der Menge Author
3. Jede Revision wird ein Knoten der Menge Revision. Ordne Revisionen eineindeutige Zeitstempel zu und verbinde jede Revision zu ihrer entsprechenden Datei und ihrem Autor
4. Erstelle Knoten für jedes MR. Das MR erbt den Zeitstempel seiner ersten Dateirevision. Verknüpfe MR zum entsprechenden Autor (Vermutlich Fehler im Paper: Man muss noch die MRs mit den Revisionen verbinden)
5. Verbinde alle MRs zu ihren jeweils nächsten MRs (nacht Zeitstempel), sofern sie existieren.
6. Für jede Datei: Verbinde jede Revision dieser Datei zur jeweils nächsten dieser Datei. Wenn gebranched wurde, werden nur Revsionen einer Brach auf diese Weise verbunden. Branch- und Mergepoints werden verbunden.

Wie bekommt man den Graphen aus SCS?

1. Jede Datei wird ein Knoten der Menge File
2. Jeder Autor wird ein Knoten der Menge Author
3. Jede Revision wird ein Knoten der Menge Revision. Ordne Revisionen eineindeutige Zeitstempel zu und verbinde jede Revision zu ihrer entsprechenden Datei und ihrem Autor
4. Erstelle Knoten für jedes MR. Das MR erbt den Zeitstempel seiner ersten Dateirevision. Verknüpfe MR zum entsprechenden Autor (Vermutlich Fehler im Paper: Man muss noch die MRs mit den Revisionen verbinden)
5. Verbinde alle MRs zu ihren jeweils nächsten MRs (nacht Zeitstempel), sofern sie existieren.
6. Für jede Datei: Verbinde jede Revision dieser Datei zur jeweils nächsten dieser Datei. Wenn gebranched wurde, werden nur Revsionen einer Brach auf diese Weise verbunden. Branch- und Mergepoints werden verbunden.

Wie sieht der Graph aus?

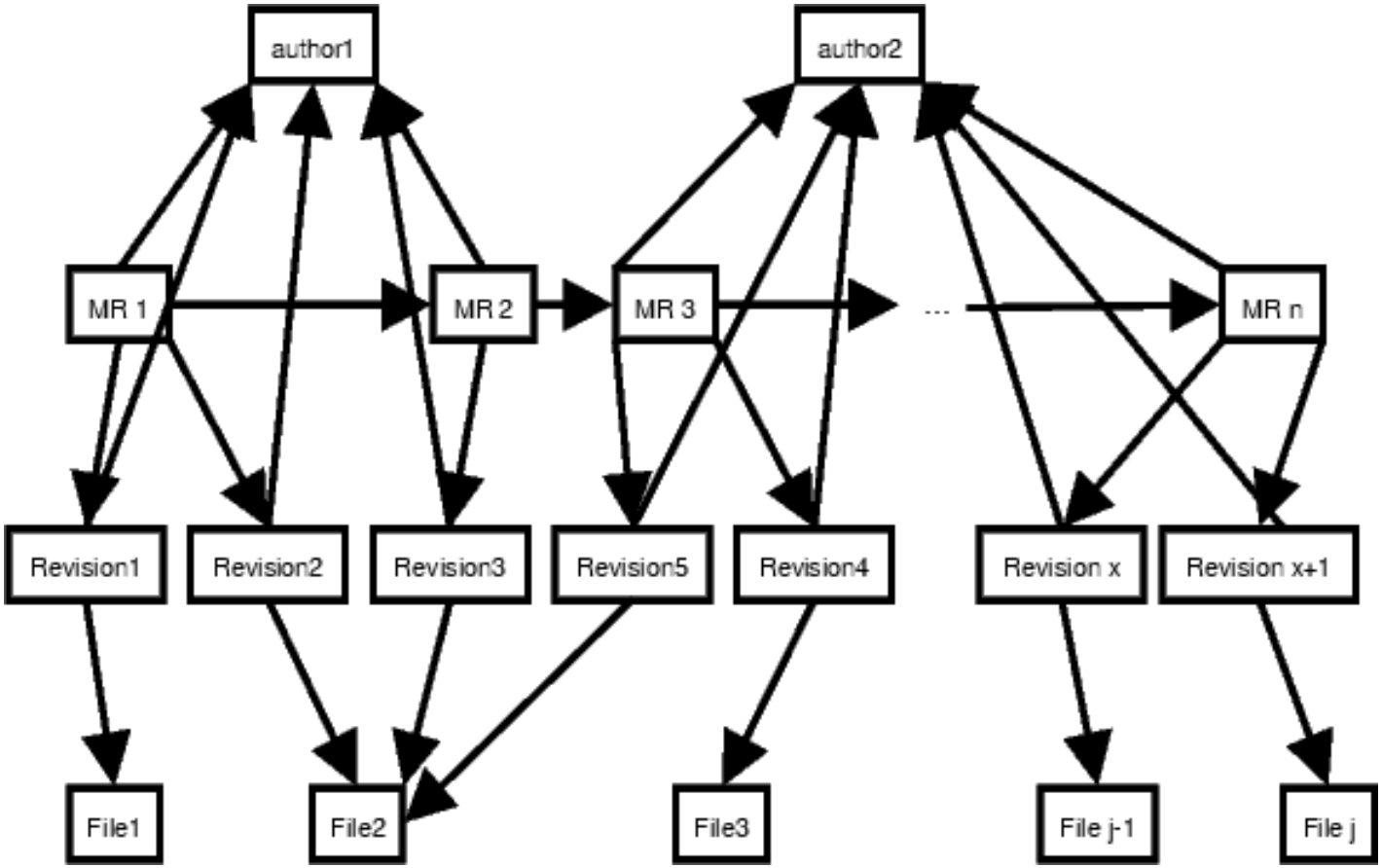


Abbildung: Model Untergraph

Wie sieht der Graph aus? (cont)

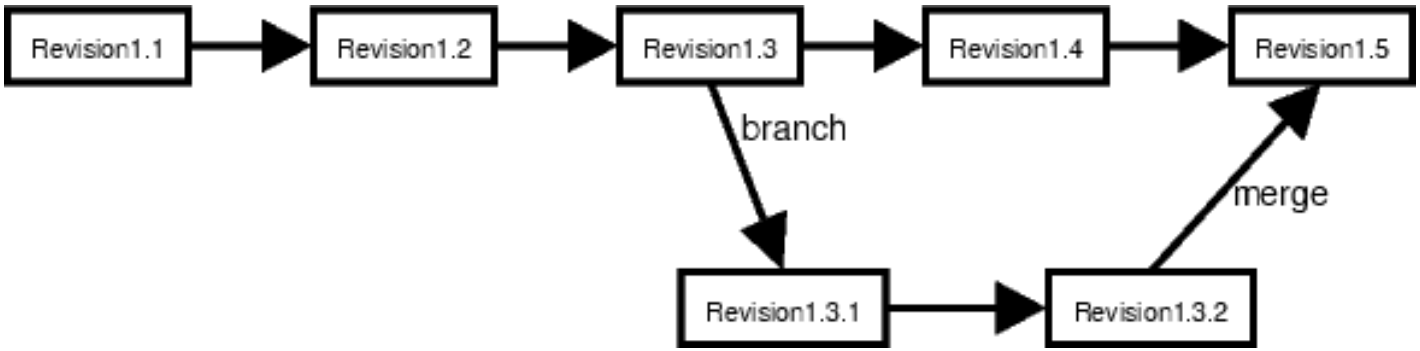


Abbildung: Revisions Untergraph

Die wichtigsten Sprachelemente

Name	Language	Explanation
$\mathbb{MR}$	MR	Set of Modification Requests
$\mathbb{R}evision$	Revision	Set of Revisions
$\mathbb{A}uthor$	Author	Set of Authors
$\mathbb{F}ile$	File	Set of Files
Universal	$A(\phi,\delta)\{P(\phi)\}$	For all ϕ in the set δ is the predicate $P(\phi)$ true?
Existential	$E(\phi,\delta)\{P(\phi)\}$	Does ϕ exist in set δ where predicate $P(\phi)$ is true?
Attribute	$\phi.\zeta$	Given an entity ϕ return its attribute ζ
Function	$\gamma(P)$	Evaluate the function γ with P as the parameter
Universal Before	$Abefore(\phi,\delta,\theta)\{P(\phi,\theta)\}$	For all ϕ in δ before θ is the binary predicate $P(\phi,\theta)$ true?
Universal After	$Aafter(\phi,\delta,\theta)\{P(\phi,\theta)\}$	For all ϕ in δ after θ is $P(\phi,\theta)$ true?
Existential Before	$Ebefore(\phi,\delta,\theta)\{P(\phi,\theta)\}$	Does ϕ exist in δ before θ where $P(\phi,\theta)$ is true?
Existential After	$Eafter(\phi,\delta,\theta)\{P(\phi,\theta)\}$	Does ϕ exist in δ after θ where $P(\phi,\theta)$ is true?
Subset	$S(\phi,\delta)\{P(\phi)\}$	Create a subset of δ, such that for each element ϕ in that subset, $P(\phi)$ is true.
Universal From Subset	$A(\theta,S(\phi,\delta)\{P(\phi)\})\{Q(\theta)\}$	For each elements θ in the set δ for which $P(\phi)$ is true, $Q(\theta)$ is also true
Anchor Select	$Anchor(\phi,MR,''mrid'')P(\phi)$	Evaluate $P(\phi)$ on the entity of type $\mathbb{MR}$ with id "mrid"
count	$count(\delta)$	Count the number of elements of the subsets δ
Sum	$Sum(\phi,\delta)\{P(\phi)\}$	Summate the predicate $P(\phi)$ for all ϕ in δ
Average	$Avg(\phi,\delta)\{P(\phi)\}$	Get the average of the predicate $P(\phi)$ for all ϕ in δ
Count	$Count(\phi,\delta)\{P(\phi)\}$	Count the number of elements ϕ in δ where $P(\phi)$ is true.

Abbildung: Sprachbeschreibung von SCQL

Die Sprache selbst ist als Prädikatenlogik erster Stufe realisiert.

Beispiel 1

Frage

Gibt es einen Author a welcher nur Dateien ändert die zuvor von Author b verändert wurden?

SCQL

```
E(a, Author) {
    E(b, Author) {
        a != b && A(r, a.revisions) {
            A(f, r.file) {
                Ebefore(r2, f.revisions, r) {
                    isAuthorOf(b, r2)
                }
            }
        }
    }
}
```

Beispiel 2

Frage

Berechne das Verhältniss der MRs welche eine eindeutige Menge von Dateien haben, die niemals zuvor als Teil eines anderen MR aufgetreten sind.

SCQL

```
1 - Count(mr, MR) {
    Ebefore(a, MR, mr) {
        A(r, mr.files) {
            isFileOf(f, a)
        }
    }
}
```

Beispiel 3

Frage

Gibt es einen Autor, dessen Änderungen nur in einem Verzeichniss stattfinden?

SCQL

```
E(a, Author) {
    A(f, author.files) {
        A(f2. author.files {
            eq(f.directory, f2.directory)
        }
    }
}
```

Auswertung

Diese drei Beispiele wurden auf fünf verschiedene Projekte angewendet: Evolution (Emailclient), Gnumeric (Spreadsheet), OpenSSL, Samba und modperl:

	evolution	gnumeric	openssl	samba	modperl
Ex 1	true	true	false	false	true
Ex 2	0.002	0.004	0.003	0.002	0.015
Ex 3	false	false	false	false	true
$\|\mathbb{F}ile\|$	4748	3685	3698	4246	300
$\|\mathbb{MR}\|$	18573	11337	10847	27413	1398

Abbildung: Auswertung der drei Beispielanfragen

Zusammenfassung

1. Motivation für SA/MSR
2. Kurze Einführung in SCS
3. Vier kurze Beispiele für MSR-Systeme
4. Ein detailliertes Beispiel

Fragen?

Danke!