Codefragmentsuche in der Programmhistorie

Exposé

Saman Bazrafshan

Universität Bremen

22. März 2010

Inhalt

1. Problemstellung

Die Evolution einer Software nachvollziehen zu können, ist in vielen Bereichen der Informatik eine ebenso interessante wie auch bedeutende Aufgabe. Es gibt sowohl wissenschaftliche Aspekte, zu denen vor allem die Analyse der Codequalität und die damit verbundenen Codeeigenschaften gehören, als auch praktische Einsatzgebiete, zu denen in erster Linie die Restrukturierung einer Software gehört. Das Projekt *Bauhaus*, das in Zusammenarbeit der Universität Bremen und der Universität Stuttgart entwickelt und gepflegt wird, beschäftigt sich genau mit dieser Aufgabenstellung und bietet verschiedene Werkzeuge zur Analyse von Code und Architektur einer Software.

Ein Aspekt einer solchen Analyse ist das Wiederfinden von spezifischen Codefragmenten in der gesamten Programmhistorie. Änderungen und vor allem Fehler, die im Laufe des Entwicklungsprozesses oder bei Wartungs- bzw. Änderungsarbeiten von den Entwicklern durchgeführt werden, müssen eventuell an verschiedenen Stellen im Programmcode vorgenommen werden, weil die betreffenden Codefragmente mehrmals unverändert verwendet wurden. Besonders wichtig ist die nachträgliche Korrektur bei Revisionen, die sich bereits im Produktiveinsatz befinden. Diese müssen mit möglichst geringer Latenzzeit mit der korrigierten Revision synchronisiert werden, um Fehler und damit Folgeschäden bzw. Verluste im Zusammenhang mit der Software zu vermeiden. Diese Aufgabe kann speziell bei großen Programmen mit vielen Revisionen und Branches zu einem wesentlichen Kostenfaktor werden.

2. Ziel

Ziel dieser Forschungsarbeit ist es, das Projekt *Bauhaus* um eine möglichst effiziente Methode zu erweitern, die es ermöglicht, nach spezifischen Codefragmenten in der gesamten Entwicklungshistorie einer Software zu suchen. Unabhängig davon, ob das gesuchte Codefragment in einer Revision der Software als Klon auftritt oder nicht, soll jedes Vorkommen ermittelt und das Ergebnis in einer verständlichen Ausgabeform präsentiert werden.

Der Algorithmus zur Suche gleicher Codefragmente soll dabei auf Basis bestehender Techniken von *Bauhaus* entwickelt werden. Zu diesem Zweck eignet sich primär das Werkzeug *clones*, dessen Aufgabe es ist, Klone zu finden. Umgesetzt wurde die Klonerkennung in *clones* mit Hilfe von Suffixbäumen, die eine zeit-effiziente Suche, auf Kosten eines erhöhten Speicherbedarfs, erlauben. Aus diesem Grund gilt es insbesondere Aufmerksamkeit darauf zu verwenden, dass der Speicherbedarf nicht mehr als unbedingt nötig erhöht wird und ein Einsatz in der Praxis weiterhin gewährleistet ist.

3. Vorgehensweise

Der Algorithmus, zur Suche nach gleichen Codefragmenten, verwendet die inkrementelle Klonerkennung von *clones* als Grundlage. Zu diesem Zweck sind Anpassungen an dem vorhandenen *clones*-Algorithmus sowie eine Erweiterung, welche die Ergebnisse von *clones* zur eigentlichen Suche nutzt, notwendig. Der gesamte Vorgang wird entsprechend in die zwei Phasen „Vorbereitung“ und „Suche“ unterteilt.

3.1. Vorbereitung

In der ersten Phase wird der in *clones* umgesetzte Algorithmus verwendet, um die benötigten Datenstrukturen, d.h. die Tokentabelle und den generalisierten Suffixbaum, zu konstruieren. Zusätzlich wird der Algorithmus dahingehend geändert, dass alle Änderungen $diff_i$ am Suffixbaum GST_i zwischen zwei Revisionen i und i + 1 in der Tabelle DST gespeichert und zu einem späteren Zeitpunkt abgerufen werden können.

Definition: GST_i bezeichnet den generalisierten Suffixbaum der Revision i mit $1 \leq i \leq n$, für n = Anzahl aller Revisionen einer Software S

Definition: $DST = \{diff_1, \ldots, diff_{n-1}\}$ enthält jede Änderung $diff_i$ am generalisierten Suffixbaum GST_i mit $1 \leq i < n$, für n = Anzahl aller Revisionen einer Software S

Die so konstruierten Datenstrukturen am Ende der inkrementellen Klonerkennung werden nicht wie bisher verworfen, sondern ebenfalls für die weitere Verwendung gespeichert. Abbildung 1 zeigt die relevanten Datenstrukturen, die von *clones* genutzt und erstellt werden, und wie diese von dem Suchalgorithmus weiter verwendet werden.

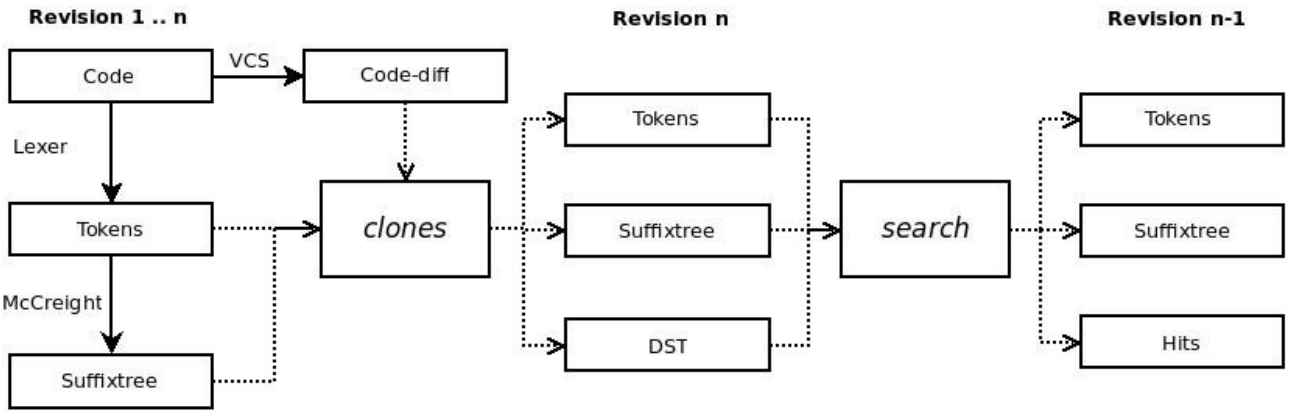

Abbildung 1: Integration des Suchalgorithmus' auf Basis der Datenstrukturen, welche von *clones* erstellt werden.

3.2. Suche

In der zweiten Phase findet die eigentliche Suche nach einem Codefragment statt. Dazu werden die DST-Tabelle, die $Tokentabelle_n$ und der Suffixbaum ST_n geladen. Anschließend läuft die Suche nach dem Codefragment chronologisch absteigend von der Revision n zu der Revision 1 der Software. In Abbildung 2 ist der Pseudocode des Suchalgorithmus' zu sehen.

```
1 for revisions n .. 1 loop
2   if i = n
3     do nothing;
4   else
5     reconstruct Suffixtree Ti from Ti+1 using di;
6   end if
7
8   begin
9     search codefragment in Suffixtree Ti;
10    if codefragment found
11      save hit;
12    end if
13  end begin
14 end loop
```

Abbildung 2: Pseudocode des Suchalgorithmus' zum Auffinden eines Codefragments in allen Revisionen einer Software.

Das Prinzip der Suche basiert darauf, dass mit Hilfe der DST-Tabelle jede Tokentabelle und jeder Suffixbaum rekonstruiert werden kann und somit nicht neu konstruiert werden muss. Die Suche in den jeweiligen Bäumen kann anschließend mit einem beliebigen Algorithmus, vorzugsweise mit dem Suchalgorithmus der in *clones* eingesetzt wird, realisiert werden, so dass voraussichtlich kein spezieller Suchalgorithmus entworfen werden muss.

4. Erwartete Ergebnisse

Folgende Ergebnisse werden im Einzelnen erwartet:

1. Die Zeit- und Platzkomplexität des *clones*-Algorithmus' wird durch die Erstellung der DST-Tabelle jeweils um einen konstanten Faktor verlängert bzw. vergrößert. Das Abspeichern der DST-Tabelle sowie von Tokentabelle_n und Suffixbaum_n sind mit unwesentlichen Kosten verbunden, die vernachlässigt werden können.
2. Die Rekonstruktion von Tokentabelle_i und Suffixbaum_i mit Hilfe der DST-Tabelle benötigt weniger Zeit, als der Algorithmus von *clones* zur Konstruktion der jeweiligen Datenstrukturen.
3. Die Suche nach einem Codefragment benötigt gleich viel Zeit wie in *clones* - vorausgesetzt der gleiche Suchalgorithmus kann verwendet werden.

5. Evaluation

Die Annahmen eins und drei können durch direkte Vergleiche des aktuellen *clones*-Algorithmus' und dessen modifizierter Version validiert werden. Zu diesem Zweck müssen geeignete Referenzprogramme gewählt und von beiden Algorithmen verarbeitet werden. Anschließend lassen sich benötigte Zeit und Platz miteinander vergleichen und Rückschlüsse können gezogen werden.

Die zweite Annahme lässt sich durch den direkten Vergleich der Konstruktionszeiten beider Algorithmen belegen. Dabei werden ebenfalls verschiedene Programme anhand spezifischer Parameter zur Untersuchung herangezogen.

6. Zeitplan / Meilensteine

- 01. April 2010: Abschluss der Vorbereitungsphase
  (a) Vorläufiger Titel ist gewählt.
  (b) Inhalt und Ziel der Arbeit sind erschlossen.
  (c) Exposé ist fertig.
  (d) Realisierbarkeit ist geprüft.
- 15. Juni 2010: Abschluss der Implementierung
  (a) Relevante Abschnitte der Ausarbeitung sind niedergeschrieben.
  (b) Relevanter Code des *Bauhaus*-Projektes wurde verstanden.
  (c) Algorithmus von *clones* ist angepasst.
  (d) Rekonstruktionsalgorithmus ist implementiert.
  (e) Ergebnisausgabe ist implementiert.
  (f) Diplomarbeit ist angemeldet.
- 01. Juli 2010: Abschluss der Evaluation
  (a) Alle drei erwarteten Ergebnisse wurden evaluiert.
- 15. September 2010: Fertigstellung der Ausarbeitung
  (a) Gesamter Inhalt der Arbeit ist niedergeschrieben.
  (b) Ausarbeitung ist mindestens von einer zweiten Person gelesen.
- 31. September 2010: Fertigstellung der Arbeit

(a) Ausarbeitung ist korrigiert.
(b) Ausarbeitung ist gedruckt.
(c) Ausarbeitung ist eingereicht.

7. Risiken

Folgende Risiken bestehen bei der Realisierung der Forschungsarbeit:

- Erlernen der Programmiersprache Ada stellt sich als problematisch heraus: Trotz vorhandener Programmierfähigkeiten im Allgemeinen, können spezifische Eigenschaften der Programmiersprache zu Problemen im Verständnis und / oder der Implementierung führen. Eine mögliche Folge könnte Zeitmangel sein, wenn dieses Risiko auftritt und im Zeitplan nicht genügend einkalkuliert wurde.
- Einarbeitung in den *Bauhaus*-Code stellt sich als problematisch heraus: Das Projekt *Bauhaus* ist über einen relativ langen Zeitraum entwickelt wurden und umfasst dementsprechend viel Sourcecode. Das Filtern und Verstehen relevanten Codes könnte länger dauern, als ursprünglich geplant, und sich dementsprechend negativ auf die eingeplante Zeit für andere Aufgaben auswirken.
- Änderungen am *clones*-Algoritmus sind nicht ohne weiteres möglich: Die Implementierung und damit auch Abhängigkeiten von dem Tool *clones* sind zu diesem Zeitpunkt nicht bekannt und könnte während der Implementierungsphase zu Problemen führen. Mögliche Folgen sind Zeitknappheit und die Umsetzung einer alternativen Lösung, die keine Änderungen an *clones* voraussetzt.
- Zeit- und / oder Platzkomplexität sind unerwartet hoch: Die Annahmen über Zeit- und Platzbedarf des Algorithmus' könnten sich als unwahr herausstellen. Sollte einer der Faktoren so groß werden, dass ein Einsatz in der Praxis unmöglich wird, muss eine Lösung, eventuell ein alternativer Algorithmus, entwickelt werden.

Abbildungsverzeichnis