Einführung:

Bei der Entwicklung von Software-Produkt-Line gibt es im Lauf der Entwicklung viele Features. Die Kombinationen von denen lassen neue Varianten der Software entstehen. In vielen Fällen können diese Features nicht frei zu einer neuen Variante kombiniert werden. Man muss extra Code einbauen, welcher entscheidet unter welchen Bedingungen ein Feature Teil einer Variante sein kann. Mit zunehmender Features Anzahl wächst auch die Größe dieses Codes. Die Entwickler sind dran interessiert, den Code besser zu verstehen und analysieren den deshalb.

(Allgemeiner DiffDetectiv beschreiben)

Zur Unterstützung der Analyse kann man Tools verwenden, solche wie DiffDetective. Dieses Tool hat viele verschiedene Funktionen, welche bei der Analyse helfen. Das Hauptmerkmal liegt aber auf zwei Graphenstrukturen und das Arbeiten/Interagieren mit denen. Auf diese Graphenstrukturen selbst und dessen Erzeugung gehen wir weiter unten genauer ein, da auch für uns diese Graphenstrukturen von Bedeutung sind.

(Erklärung von Variation-Tree/diff)

DiffDetective implementiert zwei für uns wesentliche Datenstrukturen, das sind Variation-Tree und Variation-Diff. Ein Variation-Tree ist ein Baum, welcher die Verzweigungen/Variation eines C Preprozessor Code darstellt. Ein Variation-Diff ist ein Graph, welcher die Unterschiede zwischen zwei Variation-Trees zeigt. In beiden Fällen sind die Bedingungsknoten Blau umrandet und die Code-Knoten schwarz. Beim Variation-Diff sind dazu die eingefügten Knoten Grün, die gelöschten Knoten Rot und, die unveränderten Grau.

(Parsen in DiffDetectiv)

Das Parsen führt den C Preprozessor annotierten Code von der konkreten Syntax in die abstrakte Syntax um. Das Parsen in DiffDetective funktioniert für Variation-Trees und für Variation-Diffs über einen Algorithmus. Der Algorithmus ist an sich für Parsen von textbasierten Diffs in Variation-Diffs ausgelegt, da aber jeder C Preprozessor Code in ein nicht verändertes textbasiertes Diff umgewandelt werden kann, funktioniert dieser Algorithmus auch für das Parsen von C Preprozessor Code in Variation-Trees. Bei dem Parsen wird nur der C Preprozessor Code in seine abstrakte Syntax überführt, der C bzw. C++ Code wird als Text behandelt und wird nicht geparst. Der Algorithmus geht über alle Zeilen des Codes/Textes und schaut sich an für jede Zeile, wie diese Zeile manipuliert wurde, ob die Zeile unverändert geblieben ist, gelöscht wurde oder, neu ist. Dazu wird festgelegt von welchen Typen die Zeile ist, also ob diese Zeile C/C++ Code enthält oder eine C Preprozessor Kontrollstruktur. Danach wird festgelegt, welche von den zwei erstellten Stacks für diese Zeile von Bedeutung sind, anhand dessen wie, diese manipuliert wurde. Als Nächstes wird geprüft, ob die Zeile #endif enthält, wenn ja, dann wird aus den relevanten Stacks solange Einträge herausgeholt bis ein #if kommt. Wenn die Zeile kein #endif enthält, dann wird ein neuer Knoten aus am Anfang

gesammelten Daten erstellt, wenn dieser erstellter Knoten kein Code Konten ist, dann wird, der Knoten den relevanten Stacks hinzugefügt.

Problemstellung:

Obwohl DiffDitective Funktion zum Parsen hat, hat dieses Tool aber keine Funktion zum Unparsen von Variation-Trees und Variation-Diffs. Unser Ziel ist, das zu ändern. In diesen Fall müssen wir in anderer Richtung vorgehen und die abstrakte Grammatik des Variation-Trees bzw. Variation-Diffs in die konkrete Grammatik des C Preprozessor Codes bzw. Textes überführen.

Einige mögliche Einsatzmöglichkeiten von Unparsen wären. Das Überführen von Variation-Tree in C Preeprozessor Code, falls dieser nicht vorliegt oder das Variation-Tree verändert wurde und es ist gewollt diese Änderung, als ausführbaren C Preprozessor Code zu haben und, das Überführen von Variation-Diffs in Text, falls dieser auch nicht vorhanden ist.

Für das Unparsen stellt die Tatsache, dass die Knoten von Variation-Trees und Variation-Diffs keine oder nicht glaubwürdige Einträge für Zeilen haben können, ein größeres Problem dar. Ohne die Zeilenangaben müssen wir einige Annahmen treffen und das Einhalten von Voraussetzungen für das Funktionieren unseren Unparsers verlangen. Das kann die Einsatzmöglichkeiten des Unparsers eingrenzen und kann sich auch auf den Grad der Rekonstruktion auswirken.

Beitrag:

Der Beitrag setzt sich zusammen aus Konzept, Implementierung und Auswertung. Bein dem Konzept wird ein Vorgehen zum Unpasren von Variation-Trees und Variation-Diffs in das ursprüngliche Textformat ausgearbeitet. In der Implementierung wird ausgehend von dem Vorgehen, dies in das DiffDetective Tool eingebaut. Bei der Auswertung wird anhand der, in der Bachelorarbeit spezifizierten Metrik, festgelegt, wie korrekt das Vorgehen ist. Zurzeit wird in Betracht gezogen, die Korrektheit, der Implementierung anhand folgender Kriterien festzustellen, syntaktische Gleichheit, syntaktische Gleichheit ohne Whitespace, und semantische Gleichheit.

Arbeitspakete:

Literaturrecherche: Suche der passenden Literatur.

Konzept: Ausarbeitung eines Vorgehens zum Unparsen von Vatiation-Trees/Variation-Diffs.

Metrik: Metrik oder Metriken für die Korrektheit werden festgelegt.

Implementierung: Implementierung des Vorgehens in das DiffDetectiveTool.

Auswertung: Anhand der Metrik/Metriken entscheiden, wie korrekt die Implementierung ist.

Schreiben: Die Bachelorarbeit wird geschrieben.

Inhaltilich:

- Konkreter erklären was genau ein Variation Tree ist (erstmal die Trees erklären, dann die Diffs).
- Beispiel? Also sagen, dass man C-Präprozessor-annotierten Code damit abstrahiert darstellen und analysieren kann.
- Bei der Auswertung wird anhand der Metrik festgelegt, wie korrekt das Vorgehen ist. -> Hier musst du klarer sagen, dass die Metriken erst in der Bachelorarbeiten noch analysiert werden. Hier könntest du auch schon erwähnen, dass wir uns verschiedene Äquivalenzen anschauen wollen, um die Korrektheite zu testen: syntaktische Gleichheit, syntaktische Gleichheit ohne Whitespace, semantische Gleichheit
- Zwei Schedules: einmal Nicht-Schreib-Pakete und eine Schedule nur für das Schreiben (Arbeitspakete sind Kapitel).
- verschachtelte Anweisungen -> Erst noch etwas mehr Kontext dazu: Was sind das für Anweisungen? Was tun sie?

Schreibtipps:

- TextDiff -> textbasierte Diffs (Was ist das, wo kommt das her? -> z.B.: Git-Diff oder Unix-Diff)
- auf die Stuktur eine Änderung hat -> eine Änderung auf die Struktur hat
- Ein Wort immer nur für den gleichen Zweck verwenden: Variante und Feature sind als Begriffe schon definiert für SPLs und dann sollten sie auch nur für genau diesen Zweck in deiner Arbeit benutzt werden.
- Schließlich will man aber, wieder die Textvariante haben, -> Warum will ich zurück zum Eingabtext? Was bedeutet das?
- Unparsing ist der zentrale Fachbegriff hier -> Im Topic-Sentence des jeweiligen Absatzes direkt erwähnen, dann erläutern was es ist.
- Wofür brauchgt man Unparsing?

- Der erste Satz in Absatz sagt worum es in diesem Absatz geht -> #TopicSentence
- Der Rest des Absatzes erklärt die Aussage des ersten Satzes.
- Jeder Absatz hat genau eine Botschaft => Sobald du was neues sagen willst, machst du einen neuen Absatz.

Rechtschreibung:

- Englisch: Software Product Line
- Deutsche Schreibweise: Software-Produkt-Linie
- Deutsche Schreibweise: Softwareproduktlinie

- englisch: variation trees
- deutsch: Variation-Trees

Graf -> Adliger
Graph -> Datenstruktur
denen
Manuel -> manuell

Absätze:

- Motivation: Was sind SPLs und was ist das Problem?
- Was sind Variation Trees und Diffs
- Wie werden sie geparst in DiffDetective
- Problemstellung: Es gibt aber kein Unparsing
- Warum ist Unparsing schwierig?
- Beitrag: Wie willst du es lösen?