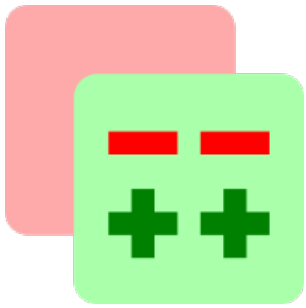
Warning Deltas

Berechnung neuer Warnungen über Git Diffs



Hintergrund

Das Jenkins Warnings Plugin [1] bietet für alle statische Codeanalyse Tools eine Delta Analyse an: damit ist es möglich, den Unterschied zwischen zwei verschiedenen Messpunkten der Analyse zu ermitteln:

- Welche Warnungen sind neu?
- Welche Warnungen wurden behoben?
- Welche Warnungen sind unverändert?

Diese Delta Analyse arbeitet in den einfachen Fällen problemlos, da sie schon jetzt Warnungen tracken kann. Leider werden z.B. bei Refactorings Zeilennummern, Namen von Identifiern, etc. Verändert, wodurch ein Tracking nicht mehr so leicht möglich ist. Neue Warnungen entstehen aber i.A. nur an veränderten Code Stellen. Daher wäre eine Kombination von Git Diffs und dem bisherigen Algorithmus eine gute Basis um die Vorhersagen zu verbessern.

Aufgabenstellung

In dieser Bachelorarbeit sollen die bestehenden Jenkins Plugins (Warnings, Analyse Modell, Git Forensics) so kombiniert werden, dass die Analyse eine verbessertes Erkennen von neuen Warnungen ermöglicht.

Dazu muss zunächst der aktuelle Algorithmus untersucht werden und parallel dazu die bestehende Schnittstelle zu Git erweitert werden, um an die Differenzen zweier Versionsstände zu kommen. Im letzten Schritt würden dann beide Verfahren miteinander kombiniert werden, so dass die Treffsicherheit verbessert wird.

Es soll hierbei ein Open Source Plugin für den Jenkins CI Server weiterentwickelt werden, das diese Ergebnisse ermittelt und visualisiert. Am Ende soll die Praxistauglichkeit dieses Ansatzes überprüft werden.

Warning Deltas

Berechnung neuer Warnungen über Git Diffs

Referenzen

[1] Jenkins Warnings Plugin, https://github.com/jenkinsci/warnings-ng-plugin
[2] Jenkins Git Forensics Plugin, https://github.com/jenkinsci/git-forensics-plugin
[3] Analysis Model, https://github.com/jenkinsci/analysis-model

Vorgehensweise

Grundverständnis des Warnings Delta Algorithmus erlangen:

- Build #1: Report mit W1 Warnings
- Build #2: Report mit W2 Warnings

Delta Report zeigt Unterschied zwischen W1 und W2:

- Neue Warnungen (nur in W2, nicht in W1)
- Behobene Warnungen (nur in W1, nicht in W2)
- Bestehende Warnungen (in W1 und in W2)

Vergleich der Warnings miteinander mehrstufig (in der Klasse IssueDifference im analysis-model):

1. Equals Methode einer Warning (z.B. Typ, Zeile, Dateiname). Alle die zueinander passen, werden als bestehende Warnungen markiert.
2. Für die restlichen erfolgt ein Fingerprint Vergleich des Source Codes aus Version 1 und 2. Ist der Fingerprint gleich, dann wird die Warnung als bestehend markiert

Fehlerquellen:

- Mehr als eine Warnung aus W1 ist equals zu einer Warnung aus W2
- Durch Refactoring ist equals und Fingerprint unterschiedlich
  - Zeilen Einfügen oder Löschen ändert schon equals
  - Rename von Variablen ändert Fingerprint
- Fingerprint kann identisch sein, obwohl Warnung nicht gleich

Neue Idee:

- Zu jeder Warnung, die nach Schritt 1. übrig bleibt, wird Git Blame aufgerufen.
- Alle Warnungen, die im Git Blame als geändert zählen, werden als neu markiert
- Nebeneffekt: Git Blame Informationen können an den Warnings gespeichert werden

Zu beachtende Punkte:

- Ein Build kann mehrere Commit IDs enthalten, d.h. die Blames sind eher eine Menge als ein einzelner Wert
- Aktuell werden die Blames im Warnings Plugin geholt und nach dem Delta ermittelt. Das muss nun vorher passieren. Damit das analysis-model unabhängig bleibt, muss dazu eine Schnittstelle definiert werden