Software Engineering in der Praxis

Aufgabenblatt 8: Software-Metriken

Die *statische Codeanalyse* umfaßt Methoden zur Analyse von niedergeschriebenem Code. Es sollen Werkzeuge erprobt werden, mit denen schlechter Programmierstil (Stichwort *Codekonventionen*), problematische Programmkonstrukte (Stichwort *Bug Pattern*) oder schlecht strukturierter Code aufgezeigt werden.

Aufgabe 1: Bug Pattern

Es soll das OpenSource-Projekt *JAP AN.ON* analysiert werden.

a) Laden Sie den Quellcode der *AnonLib* mit den benötigten Bibliotheken von unseren Netzlaufwerken herunter.

b) Erstellen Sie ein neues Java-Projekt in Borland Together. Importieren Sie die *AnonLib*-Quellen und geben Sie unter Projekteigenschaften unter *Build Path* die Bibliotheken an. Es sollten nun keine Fehler mehr angezeigt werden.

c) Schalten Sie Warnungen zu überflüssigem Code und Typ-Sicherheit für dieses Projekt (Projekteigenschaften/Java Compiler) ab.

d) Starten Sie die das Java-Programm *FindBugs* aus dem Netzlaufwerk und analysieren Sie damit die *AnonLib*.

e) Studieren Sie die gefundenen Pattern und ihre Beschreibung. Welche erscheinen Ihnen besonders aussagekräfig, welche eher fragwürdig? Wählen Sie einige Pattern aus den verschiedenen Gruppen aus, diskutieren sie Sie und halten Sie Ihre Argumente in einer Textdatei fest.

Aufgabe 2: Software Metriken mit Together

a) Berechnen Sie mit Hilfe von Together die voreingestellten Software-Metriken für die *AnonLib*.

b) Finden Sie Klassen, die gleichzeitig groß (LOC) und komplex (CC) relativ zu den anderen Klassen im Projekt sind. Vergleichen Sie dazu die Klassen mit Hilfe einer Balkengrafik.

c) Wie hoch ist die maximale Verschachtelungstiefe?

d) Wie ist der Wert für *CBO* zu beurteilen?

Aufgabe 3: Beeinflussung von Metriken

An Hand eines kleinen Beispiels soll die Beeinflussbarkeit von Metriken und die Korrelation zwischen Metriken gezeigt werden.

a) Legen Sie mit Together ein Projekt an und erstellen Sie eine Klasse *Sort* mit einem einfachen Sortieralgorithmus:

b) Implementieren Sie eine zweite Klasse, die eine main-Methode enthält um den Algorithmus zu testen. Aktivieren Sie die Plugins *FindBugs* und *Checkstyle* für das Projekt.

c) Stellen Sie mit Together einen Satz Metriken zusammen, mit dem Sie Ihre Implementierung messen. Der Satz Metriken sollte folgende Metriken auf jeden Fall enthalten: LOC, CC, HPLen, HPVol. Führen Sie die Messung aus, und exportieren Sie die entstandene Tabelle im html-Format in eine Datei.

d) Legen Sie zwei Kopien der Klasse *Sort* an (*Sort1* und *Sort2*). Verändern Sie in Klasse *Sort1* Ihre Implementierung bei gleichbleibender Funktionalität dahingehend, dass die Halstead-Metriken möglichst einen Wert kleiner dem Wert von *Sort* haben. Verändern Sie entsprechend *Sort2* so, dass die Werte grösser denen von *Sort* sind.

e) Bauen Sie die Klasse *JDKSort* in Ihr Projekt ein und bestimmen Sie auch hierzu die Werte der Metriken. Dokumentieren Sie in einer Textdatei, in wie weit die Aussage der Metriken mit Ihrem Aufwand, den JDKSort-Algorithmus zu verstehen, übereinstimmt.

f) Exportieren Sie die Ergebnisse der Berechnung in eine html-Tabelle und speichern Sie zu jeder Klasse das Kiviat-Diagramm.