Continuous Testing

Automatisierte Qualitätssicherung des Entwicklungsprozesses

Der Test-First-Ansatz ist im Java-Umfeld weit verbreitet, die Nutzung von JUnit in Entwicklertests ist aus keinem Projekt mehr wegzudenken. In großen Projekten schleicht sich allerdings oftmals eine Nachlässigkeit ein: Nicht alle vorhandenen Tests werden vom Entwickler neu gestartet, z.B. weil an einer vermeintlich unbeteiligten Codestelle Änderungen vorgenommen wurden. Ein fehlschlagender Test fällt dann erst in einer späteren Integrationsphase auf und die Fehlersuche ist um ein Vielfaches aufwendiger. Das Continuous Testing-Plug-in automatisiert diesen Prozess, indem es alle Testfälle im Hintergrund anstößt, sobald der Code geändert wurde. So hat der Entwickler immer im Blickfeld, ob seine Änderungen einen Bug an einer unerwarteten Stelle hervorgerufen haben.

Das Continuous Testing-Plug-in von David Saff ist schnell installiert. Nach Angabe der URL im Update Manager [1] kann das Plug-in automatisch heruntergeladen und in die Eclipse-Umgebung eingebunden werden. Anschließend kann pro Projekt eingestellt werden, ob Continuous Testing aktiviert werden soll. Dazu wird in den Projekteigenschaften im Karteireiter Continuous Testing Properties das Flag Enable Continuous Testing aktiviert. In der Standardeinstellung werden damit alle Tests des ausgewählten Projektes für die automatische Qualitätssicherung herangezogen.

Nach dieser Aktivierung ist das Plug-in bereit. Sobald Quellen des zu überwachenden Projektes geändert werden und der Eclipse Compiler diese fertig übersetzt hat, werden auch die Testfälle automatisch angestoßen. Die Ausführung der Tests erfolgt im Hintergrund, sodass der Entwickler nicht im Fluss unterbrochen wird.

Schlägt ein Testfall fehl, so wird ein Eintrag in der Problems View erzeugt, aus dem der fehlgeschlagene Test ersichtlich wird. Damit sind fehlgeschlagene Tests genauso leicht sichtbar und navigierbar wie Compiler-Fehler bzw. -Warnungen. Die verwendeten Icons fügen sich in die gewohnte Eclipse-Umgebung ein: Ein roter Ball mit einem „T“ signalisiert hier einen fehlgeschlagenen Test. Zusätzlich zu diesem Eintrag ist auch die betroffene Testmethode rot unterkringelt. Damit ist der fehlgeschlagene Testfall schnell zu finden. Natürlich kann damit der Bug selbst nicht lokalisiert werden; dafür ist der Entwickler immer noch selbst zuständig.

Die jeweils durchzuführenden Tests werden in einer Launch Configuration definiert, ähnlich wie dies auch bei normalen JUnit-Tests geschieht. Zu den üblichen Eclipse-Parametern (Klassenpfad, Argumente etc.) sind zwei neue Karteireiter hinzugekommen:

1. Die Reihenfolge der Tests kann über die Priorisierung beeinflusst werden. Als Alternativen stehen folgende Optionen zur Verfügung: die schnellsten zuerst, alle fehlgeschlagenen Tests zuerst, die am häufigsten fehlgeschlagenen zuerst sowie eine zufällige Reihenfolge.
2. Die Auswahl der Testfälle kann ebenso flexibel eingestellt werden. Zunächst kann wie in einer JUnit-Launch-Konfiguration die zu startende Testklasse ausgewählt werden. Alternativ können auch alle Tests des Projekts verwendet werden. Für die dann durchzuführenden Tests kann noch ein Filter bestimmt werden. Beispielsweise können nur noch Tests gestartet werden, die im vorhergehenden Versuch fehlgeschlagen sind. Ein ähnliches Feature ist auch Bestandteil der kommenden Eclipse-Version 3.1. Nach einem Testdurchlauf können dann über eine neue Schaltfläche alle fehlgeschlagenen Tests nochmals durchlaufen werden.

Ein weiteres Highlight dieses Plug-ins ist die Erweiterung der JUnit View. Mit dem Continuous Tester steht nun auch in dieser View eine History-Funktion zur Verfügung. So sind nicht nur der aktuelle Testdurchlauf sichtbar, sondern über eine Auswahlliste auch alle älteren Testdurchläufe. Vielleicht findet sich ja diese Erweiterung auch bald im Hauptest des Eclipse-Projektes wieder, ein dafür entsprechender Change Request ist gestellt.

Dr. Ullrich Hafner

Links & Literatur

[1] David Saff, Continuous Testing-Plug-in: pag.csail.mit.edu/continuoustesting/

FindBugs

Programmierfehler finden

Wer kennt das nicht: Da hat ein Programm monatelang wie gewünscht funktioniert, aber plötzlich häufen sich aus unerfindlichen Gründen im Betrieb die Ausfälle. Langes Suchen fördert dann irgendwann die Ursache zu Tage: An einer bestimmten Stelle im Code wurden zwei Strings nicht mit der *equals()*-Methode verglichen, sondern mit dem Operator. Oder es wurde nicht geprüft, ob beim Zugriff auf eine Methode ein Nullpointer auftrat. All diese Fehler kann der Compiler nicht aufdecken, da das Programm syntaktisch korrekt formuliert ist. Ein Unit-Test könnte solche Fehler durchaus aufdecken, wenn er

denn alle möglichen und unmöglichen Fehlerfälle überprüfen würde – leider ist dies normalerweise nicht der Fall, da man sich bei der Formulierung von Unit-Tests (leider allzu oft) auf die offensichtlichen Fehlerfälle konzentriert.

An dieser Stelle springt FindBugs [1] in die Bresche. FindBugs ist ein Programm zum Auffinden von so genannten Bug Patterns. Genauso wie es bei der Entwicklung bestimmte Muster gibt, welche die Software im positiven Sinne strukturieren, gibt es auch Muster für schlechtes Verhalten. Dazu zählen die Nullpointer Dereference, Illegal String Compare sowie eine ganze Reihe weiterer Fehler, die Java-Programmierer gern machen. FindBugs analysiert den Bytecode eines Programms mithilfe der Bytecode Engineering Library (BCEL) und sucht nach dem Vorkommen solcher Bug Patterns. Eigene Bug Patterns können ergänzt werden, indem man einen so genannten Bug Detector schreibt.

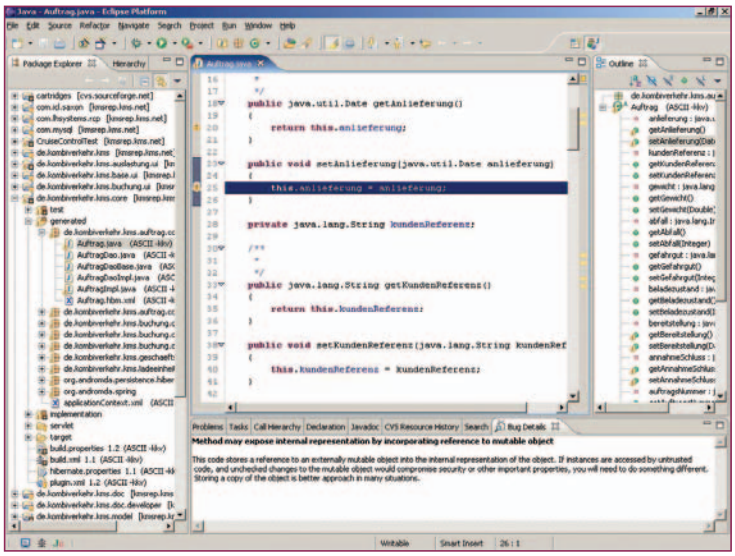

Abb. 1: FindBugs erklärt gefundene Fehler und gibt Korrekturvorschläge

FindBugs verfügt über insgesamt vier dem jeweiligen Einsatzgebiet angepasste User Interfaces. Für die automatisierte Analyse von Programmen z.B. im Rahmen einer Continuous Integration gibt es neben dem obligatorischen Ant-Task auch ein Plug-in für Maven. Für die dialoggestützte Analyse von Programmen existierte lange Zeit nur ein Swing-Frontend. Mittlerweile gibt es jedoch auch ein Eclipse-Plug-in. In seiner aktuellen Version erlaubt das Plug-in die Überprüfung der eigenen Programme auf Anforderung (d.h. durch expliziten Aufruf über das Kontextmenü eines Java-Projekts) oder automatisch bei Abspeichern einer Java-Datei. Nachdem Eclipse mithilfe des internen Java-Compiler die Java-Dateien übersetzt hat, startet das FindBugs-Plug-in die FindBugs Engine, um in den erzeugten Klassendateien nach Fehlern zu suchen. Wird ein Bug Pattern gefunden, so wird die entsprechende Stelle im Quellcode mit einem Warnungsmarker versehen. Durch das Anklicken eines Markers wird die Bug Details View aktiviert, eine Ansicht, in welcher der Fehler kurz beschrieben wird.

Der erwähnte Marker erscheint übrigens automatisch in der Problems View, in der auch die Fehlermeldungen des Java Compiler ausgegeben werden – so hat man alle Fehler und Warnungen immer im Blick. Auch die Filter- und Sortiermöglichkeiten der Problems View können so genutzt werden, z.B. um nur die Fehler der aktuell angezeigten Datei aufzulisten. Da FindBugs mittlerweile über 60 Bug Detectors enthält, kann ein FindBugs-Lauf einige Zeit in Anspruch nehmen. Daher lässt sich über eine Eigenschaftenseite (Abb. 2) projektspezifisch definieren, welche Bug Detectors überhaupt ausgeführt werden sollen. Auch das automatische Ausführen von FindBugs sollte man ab einer gewissen Projektgröße besser abschalten und nur von Zeit zu Zeit einen manuellen Lauf starten.

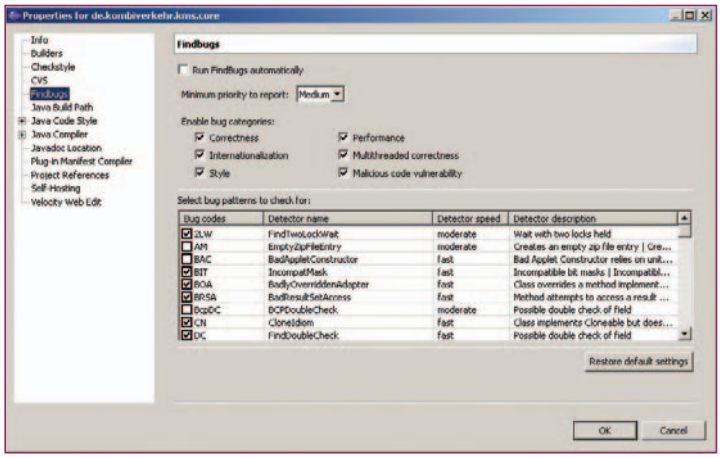

Abb. 2: Konfiguration des Plug-ins auf Projektebene

Fazit

FindBugs ist an sich ein sehr nützliches Tool, das durch das Eclipse-Plug-in eine noch bessere Integration in den Entwicklungsprozess erfährt. Gerade die automatische Überprüfung der just gespeicherten Datei ist wirklich praktisch und gehört zu meinen Lieblingsfeatures. Bei der Arbeit mit FindBugs wird jeder Entwickler feststellen, dass er ganz bestimmt „Lieblingsfehler" hat, die er immer wieder macht. Durch das sofortige Feedback kann das Eclipse-FindBugs-Plug-in helfen, sich solche Bug Patterns abzugewöhnen.

Unter [2] gibt es eine Preview des GUI der nächsten Version des Eclipse-FindBugs-Plug-in, mit der es dann auch möglich sein wird, die Konfiguration des Plug-in zu exportieren, um sie auf einem anderen Rechner wieder zu importieren – eine Funktion, die insbesondere von Teams immer wieder gefordert wurde.

Peter Friese

Links & Literatur

[1] FindBugs: findbugs.sourceforge.net

[2] www.tobject.de