Hausarbeit
Software-Qualität

Dan Helmvoigt

Torsten Windoffer

Qualitätssicherung von GUI-Anwendungen
mit Hilfe des

Google™ WindowTester Pro™

Inhaltsverzeichnis

Allgemeines

Name:	Google WindowTester Pro
Homepage:	http://code.google.com/intl/en/javadevtools/wbpro/index.html
Lizenz:	Commercial – Free
Untersuchte Version:	6.0
Eclipse Version:	3.6.1 (Helios)
Letzter Untersuchungsdatum:	07.02.2011

Kurzbeschreibung

Mit dem Google WindowTester Pro können auf Swing, SWT oder Eclipse/RCP basierende Benutzeroberflächen von Java Programmen getestet werden. Darüber hinaus gibt es die Möglichkeit von einfachen Überdeckungstests.

Der WindowTester wird als Plugin in die Eclipse IDE integriert.

Die Tests werden auf Basis von JUnit 3 geschrieben. Eine Unterstützung von JUnit 4 ist zurzeit noch nicht gegeben, befindet sich aber laut Hersteller in der Entwicklung.

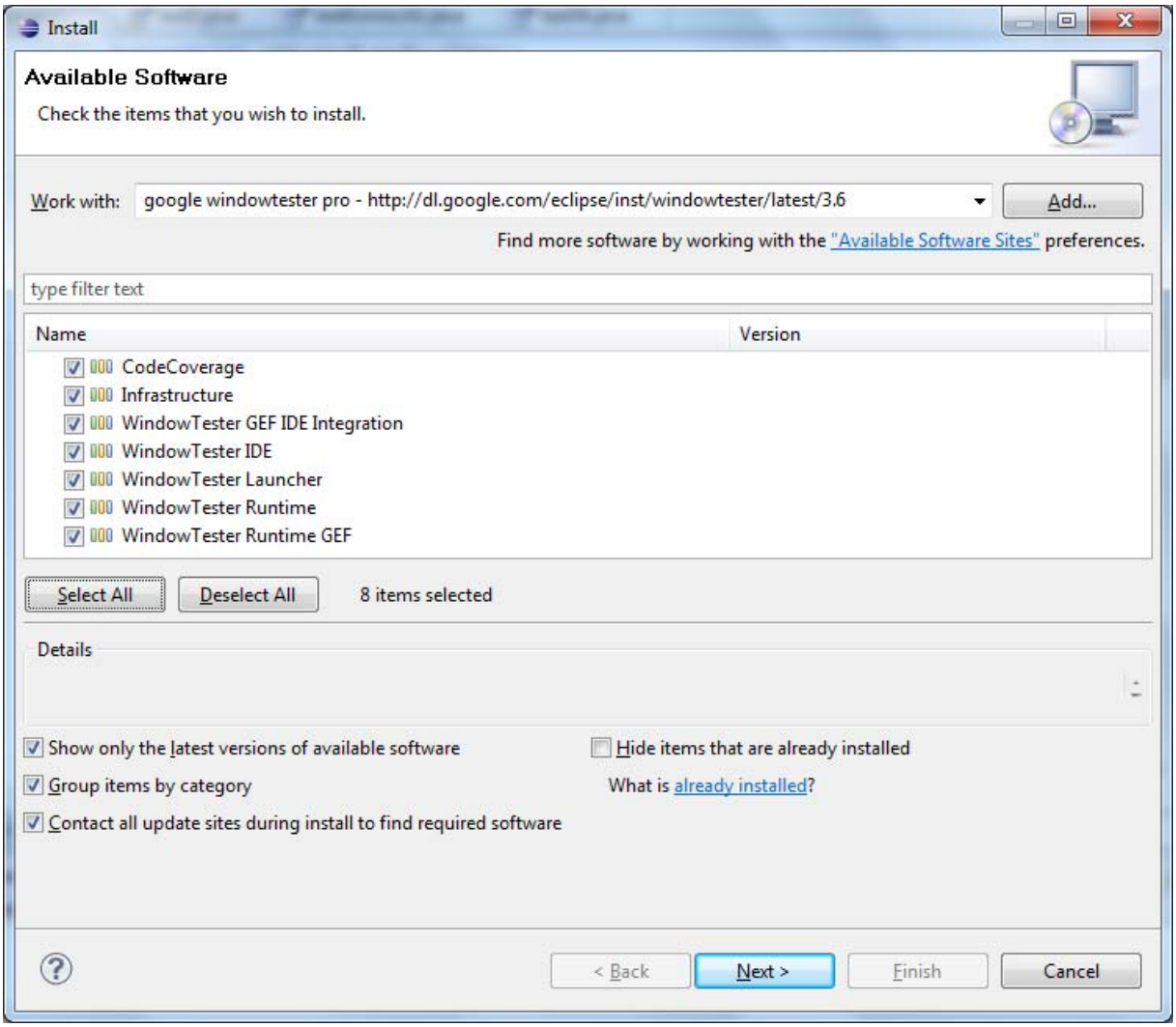
Zur einfachen Erstellung von Tests gibt es einen Recorder. Dabei werden die Eingaben und Klicks auf der Benutzeroberfläche aufgezeichnet und hieraus dann automatisch ein Test generiert wird.

Durch den Einsatz von JUnit können z.B. für die Automatisierung von Tests dessen Funktionen genutzt werden.

Zusammen mit den Programmen WindowBuilder Pro (GUI-Entwicklungswerkzeug) und CodePro AnalytiX (Codeanalyse-Tool) wurde die Software bis September 2010 von der Firma Instantiatons entwickelt und kommerziell vertrieben. 2010 wurden die Tools vom Google aufgekauft und stehen seit dem kostenlos zum Download bereit. Die Versionsnummer vom WindowTester wurde nach der Übernahme direkt ohne Änderungen der Funktionalität auf 6.0 angehoben.

Installation

Die Installation unter Eclipse gestaltet sich sehr simpel und sollte auch Eclipse Einsteigern keinerlei Probleme bereiten. Über die integrierte Pluginverwaltung (Help -> Install New Software...) kann der WindowTester unter Angabe der Adresse http://dl.google.com/eclipse/inst/windowtester/latest/3.6 (für Eclipse 3.6) und Auswahl der entsprechenden Komponenten automatisch installiert werden. Mit einem Klick auf “Next” gelangt man zum nächsten Dialog, in dem man aufgefordert wird die Lizenzbedingungen zu akzeptieren, ist man damit einverstanden und tut dieses, startet die Installation mit einem Klick auf “Finish”. Nun downloadet und installiert Eclipse die neue Software, nach der Installation wird noch ein Neustart gefordert und der WindowTester Pro ist einsatzbereit.



Installationsfenster von Eclipse

Die Installation funktioniert unter Windows und Linux gleicher maßen. Probleme mit System- und Softwarevoraussetzungen sollte es keine geben, da alles was der WindowTester benötigt auch schon vom Eclipse selber vorausgesetzt wird.

Systemvoraussetzungen

Betriebsysteme:	Windows ME/2000/XP/Vista/7, Linux (incl. GTK2)
JVM:	Sun JVM 1.5/1.6
Eclipse:	3.4 - 3.6

Deinstallation

Wie auch die Installation gestaltet sich die Deinstallation denkbar einfach. Über den Menüpunkt “Help -> About Eclipse SDK -> Installation Details” können die entsprechenden Pakete ausgewählt und deinstalliert werden.

Oberflächentests

Testgenerierung

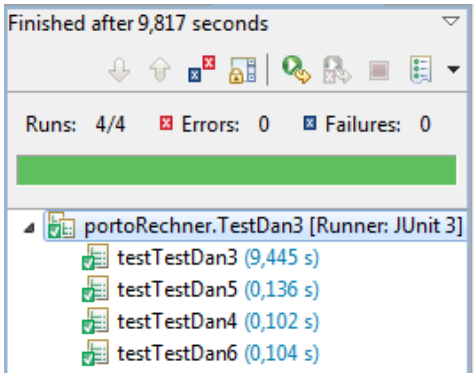
Die Funktion des Google WindowTester Pro wird anhand eines kleinen Java Programms zur Portoberechnung gezeigt. Die GUI wurde mithilfe des Google WindowBuilder Pro auf Basis der Swing-Bibliothek erstellt.

Zu testendes Beispielprogramm

Um einen neuen Test zu schreiben wird zunächst eine neue JUnit Testklasse angelegt. Dies ist in diesem Beispiel die Klasse `TestProtoRechner`, welche von `UITestCaseSwing` erbt. Im Konstruktor muss man die zu testende Klasse an die Elternklasse übergeben. Anschließend kann die erste Methode zum Testen geschrieben werden. Dazu muss man zuerst über die Funktion `getUI()` Zugriff auf die GUI holen. Über dieses `IUIContext` Objekt lassen sich nun Aktionen wie beispielsweise Klicks ausführen. Der `.click`-Methode wird ein s.g. Locator übergeben. Dieser Locator sucht die Komponente der GUI, auf die der Klick ausgeführt werden soll. Es stehen unter anderem, je nach GUI Komponente, folgende Locators zur Verfügung:

- `JTextComponentLocator`: Sucht speziell nach einer Textkomponente wie `JTextField`, `JTextArea`, `JEditorPane` oder `JTextPane`. Solche speziellen Locators gibt es für die gängigsten Komponententypen. Je nach Typ stehen verschiedene "Identifikationsmerkmale" wie z.B. der Text zur Verfügung.
- `LabeledTextLocator`: Sucht nach einem Label mit einer bestimmten Beschriftung und wählt das im Quellcode als nächstes kommende Textfeld aus. Zu beachten ist, dass es zu Problemen kommen kann, wenn es mehrere Labels mit dem gleichen Namen gibt.
- `NamedWidgetLocator`: Wählt eine Komponente anhand des mit der setName-Methode gesetzten Names aus. Dies ist die zuverlässigste Methode zur Auswahl der gewünschten Komponente, das diese Unabhängig von der Reihenfolge im Quellcode ist. Einzig muss darauf geachtet werden, dass keine Namen doppelt vergeben wurden.

Mit Hilfe der assertThat-Methode der IUIContext-Klasse lassen sich Eigenschaften von Komponenten prüfen. Dazu wird die zu überprüfende Komponente mittels eins Locators ermittelt und z.B. mit der hasText-Methode oder mit der isEnabled-Methode geprüft. Zudem kann dem Assert ein String mit übergeben werden, welcher von JUnit angezeigt wird, falls der Test fehl schlägt. Jeder Assert sollte in einer separaten Methode liegen, da immer eine Methode als Testsfall betrachtet wird. Sollten mehrere Asserts in einer Methode liegen und diese schlägt fehl, kann nicht festgesellt werden welcher Assert den Fehler verursacht hat.



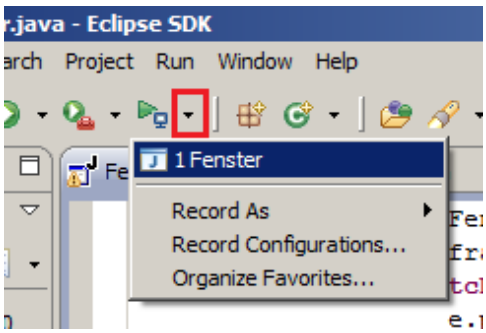
Erflogreiche JUnit Tests

```
public class TestPortoRechner extends UITestCaseSwing {
     public TestPortoRechner() {
          super(portoRechner.PortoRechner.class);
     }
     public void Test() throws Exception {
          IUIContext ui = getUI();
          ui.click(new JRadioButtonLocator("5kg bis 10kg"));
          ui.click(new LabeledTextLocator("Höhe:"));
          ui.enterText("20");
          ui.click(new LabeledTextLocator("Breite:"));
          ui.enterText("40");
          ui.click(new NamedWidgetLocator("txtDeep"));
          ui.enterText("40");
          ui.click(new JCheckBoxLocator("Express"));
          ui.click(new JButtonLocator("Berechnen"));
          ui.assertThat("Erwarteter Preis: 16.5",
               new NamedWidgetLocator("portoPreis")
               .hasText("16.5"));
     }
}
```

Dieses Beispieltestprogramm wählt den Radiobutton mit der Bezeichnung “5kg bis 10kg” aus und schreibt in die Felder “Höhe”, “Breite”, “Tiefe” die Werte 20, 40, 40. Anschließend wird das Kontrollkästchen “Express” angewählt und auf den Button “Berechnen” geklickt. Zuletzt wird verglichen ob der berechnete Preis dem erwarteten Ergebnis entspricht.

Recorder

Mit dem Recorder kann die Eingabe aufgezeichnet werden und es wird automatisch ein JUnit 3 Test generiert.



Neue Aufnahme starten

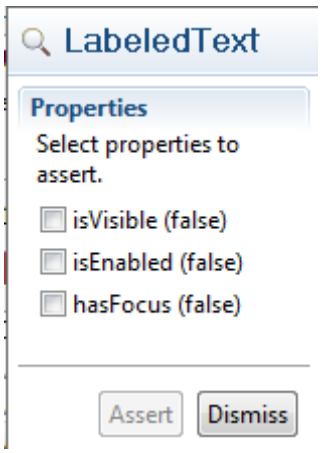
Wurde das Programm bereits einmal ausgeführt, so kann man den Recorder durch einen Klick auf den Pfeil neben dem Recorder-Symbol und Auswahl der GUI-Klasse (hier “Fenster”) starten. Daraufhin öffnet sich ein kleines Recorderfenster mit Steuerelementen sowie eine Verlaufsspalte.

Recorder Konsole / Verlauf und Aufnahmefenster

Über den “Record”-Button kann die Aufzeichnung gestartet werden. Läuft die Aufzeichnung, kann die zu testende GUI ganz normal bedient und Eingaben gemacht werden. Im Recorder-Verlauf werden die Aktionen aufgelistet und können z.B. bei Fehleingabe oder falschem

Klick wieder gelöscht werden. Beendet wird die Aufzeichnung durch den “Pause”-Button und Schließen des zu testenden Fensters. Im nächsten Dialog gibt man einen Namen für den Test und das Package, in dem gespeichert werden soll an.

Über den Button mit der Lupe lassen sich so genannte “Inspections” einfügen. Ist dieser Punkt während einer Aufnahme aktiv, so erscheint, wenn die Komponenten der Test-GUI mit dem Cursor überfahren werden, ein kleines Fenster. Über dieses Fenster lassen sich automatisch Asserts erzeugen, die beispielsweise prüfen ob ein Element aktiv oder sichtbar ist. Die Möglichkeiten sind leider recht eingeschränkt, so lassen sich keine Texte vergleichen. Zu beachten ist auch, dass alle diese Asserts in derselben Methode landen, womit kein sauberes Testszenario mehr gegeben ist. Diese Funktion ist nicht verfügbar wenn der “Recorder Mode” auf “Classic” eingestellt ist (Window -> Preferences -> WindowTester -> Recorder UI).

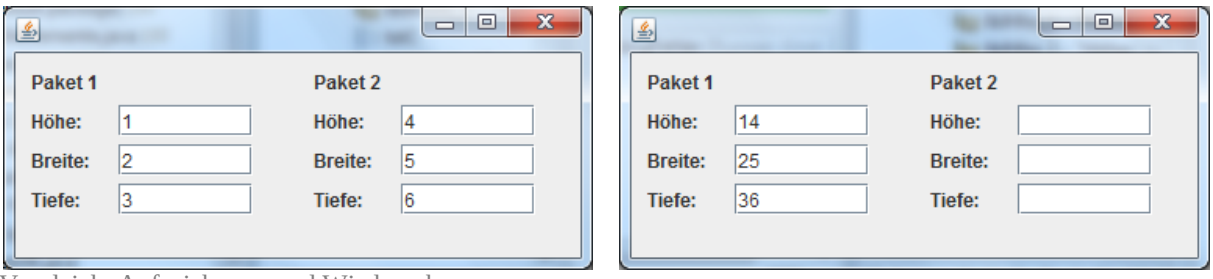


Inspector Fenster

Mit dem letzten Button (Stift mit Pfeil) lassen sich “Assertion Hooks” anlegen. Wird dieser Knopf während einer Aufnahme gedrückt, wird im Quellcode des Tests eine leere Methode angelegt. Der Methodenaufruf erfolgt später in entsprechender Reihenfolge, abhängig davon wann dieser bei Aufnahme angelegt wurde. Diese Methode muss anschließend manuell ausprogrammiert werden.

Fehler beim Aufzeichnen

Beim Aufzeichnen sollte darauf geachtet werden, dass die GUI-Komponenten über die setName-Methode einen eindeutigen Namen haben. Ist dies nicht der Fall, können zwischen dem Aufgezeichnetem und der Wiedergabe Differenzen auftreten, da der Recorder, wenn vor einem Textfeld ein Label liegt (Reihenfolge im Quellcode), immer mit der Klasse `LabeledTextLocator` arbeitet. `LabeledTextLocator` bekommt als Parameter den Labeltext und gibt ein Objekt auf das darauffolgende Textfeld im Quellcode zurück. Wenn mehrere gleichbenannte Label in einem View vorkommen kann nicht zwischen diesen unterschieden werden und der WindowTester greift beim Abspielen immer auf das im Quellcode zuerst vorkommende zu.



Vergleich: Aufzeichnung und Wiedergabe

Im linken Bild ist zu sehen, wie die Progammeingaben aufgezeichnet wurden und im rechten wie diese beim Abspielen reproduziert wurden. Die Reihenfolge der Zahlen ist dabei gleich der Reihenfolge der Eingaben.

Ein kurzer Blick in den vom Recorder erzeugten Quellcode zeigt klar das Problem auf.

```
...
ui.click(new LabeledTextLocator("Höhe:"));
ui.enterText("1");
ui.click(new LabeledTextLocator("Breite:"));
```

```
ui.enterText("2");
ui.click(new LabeledTextLocator("Tiefe:"));
ui.enterText("3");
ui.click(new LabeledTextLocator("Höhe:"));
ui.enterText("4");
...
```

Abhilfe würde hier ein index-Paramter wie bei der Klasse `JTextComponentLocator` schaffen, welcher angibt das wievielte Vorkommen gemeint ist.

Unterstützte Komponenten

Vom Recorder werden nicht alle Swing-Komponenten erkannt bzw. unterstützt.

Wird während eines Records eine Aktion auf eine nicht unterstütze Komponente ausgeführt, so wird dies im Quelltext vom generierten Text durch einen Kommentar markiert. Zu beachten ist nun, dass die Aktion, die in der GUI ausgeführt wurde, auf die vorherige Komponente angewandt wird und es so zu Fehlern im Testablauf kommen kann.

Solche vom Recorder nicht erkannten Komponenten müssen manuell dem Test hinzugefügt werden. Die einfachste Methode ist die Komponente über ihren Namen mit Hilfe des `NamedWidgetLocators` anzusprechen.

Verhalten bei nicht sichtbaren Komponenten

Ist eine Komponente nicht sichtbar, wird aber z.B. über einen `NamedWidgetLocator` angesprochen, so kommt es bei der Ausführung des Tests zu einem Timeout und der Test wird mit einem Fehler abgebrochen.

Beispiel:

GUI-Quellcode:

```
...
hiddenTextField = new JTextField();
hiddenTextField.setVisible(false);
...
```

Test-Quellcode:

```
...
ui.click(new NamedWidgetLocator("hiddenTextField"));
ui.enterText("Ich bin versteckt!");
...
```

Fehlermeldung:

```
abbot.tester.Robot.waitForComponent(Robot.java:740): Component
'hiddenTextField' (JTextField) (3fe2670b) not ready after
30000ms: showing=false win ready=true
```

Verhalten bei mehreren Fenstern

Generell wird der Umgang mit mehreren Fenstern vom WindowTester unterstützt. Der Tester arbeitet immer in dem aktuell aktiven Fenster. Wird z.B. das Löschen eines Datensatzes in einem Ja / Nein Dialog abgefragt kann diese Frage beantwortet und anschließend im vorherigen Fenster weitergearbeitet werden. Es ist allerdings nicht möglich Anwendungen zu testen, bei denen interaktiv zwischen mehreren Fenster gewechselt werden soll, da in den

Testfällen nicht beschrieben werden kann, dass der WindowTester ein anderes Fenster durch klicken aktiv setzt.

Vergleich mit Konkurrenzprodukten

Zum Vergleich werden die Tools Fest und Marathon herangezogen, da diese im Rahmen der Vorlesung Softwarequalität WS2010/11 näher betrachtet wurden.

Der Google WindowTester Pro vereint die Funktionalitäten von Fest und Marathon in einem Tool. Es können Tests in Java manuell geschrieben oder mit Hilfe des Recorders automatisch erstellt werden.

Ein zusätzliches Feature, welches nur der WindowTester bietet, ist die integrierte Möglichkeit der Durchführung von Überdeckungstests.

Fest unterstützt Swing und JavaFX als GUI-Toolkit und Marathon nur Swing. Der WindowTester hingegen Swing, SWT und Eclipse/RCP.

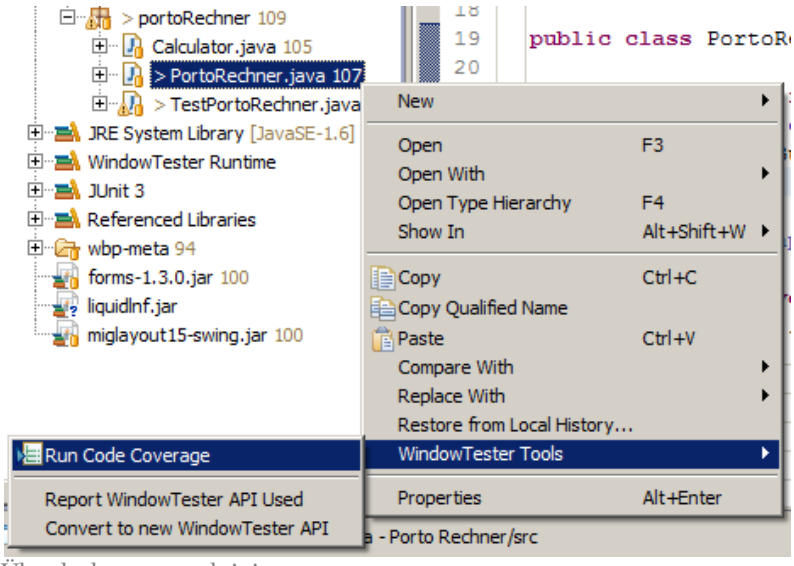
Vergleichsmatrix
(auf Basis von http://home.edvsz.fh-osnabrueck.de/skleuker/CSI/Werkzeuge/vergleichGUITester.html)

Name	Fest	Marathon	WindowTester Pro
Grundlage	TestNG oder JUnit 3.8.1	JUnit 3.8.1	JUnit 3.8.x
Skriptrekorder	Nein	Ja	Ja
Manuelle Tests	Ja	Ja (Begrenzt mit Jython)	Ja
Automatisierbarkeit	Ja (über JUnit)	Ja (über JUnit)	Ja (über JUnit)
Einsatzumgebung	Einzelanwendung	Einzelanwendung	Einzelanwendung
Bedienung	Sehr gut	Sehr gut	Sehr gut (nach Einarbeitung)
Intuitive Nutzbarkeit	Sehr gut, sehr zugänglich	Sehr gut, mächtig aber übersichtlich	Gut, es ist ein gewisse Einarbeitung in die API, speziell bei den Locators notwendig.
Installation	Keine Installation erforderlich	Keine Installation erforderlich	Als Eclipse Plugin
Dokumentation	Durchschnittlich, eher zu knapp	Sehr gut, umfangreich mit vielen Beispielen	Gut, es gibt zu wenig prägnante Beispiele
Fazit	Empfehlenswert	Empfehlenswert	Empfehlenswert

Überdeckungstests

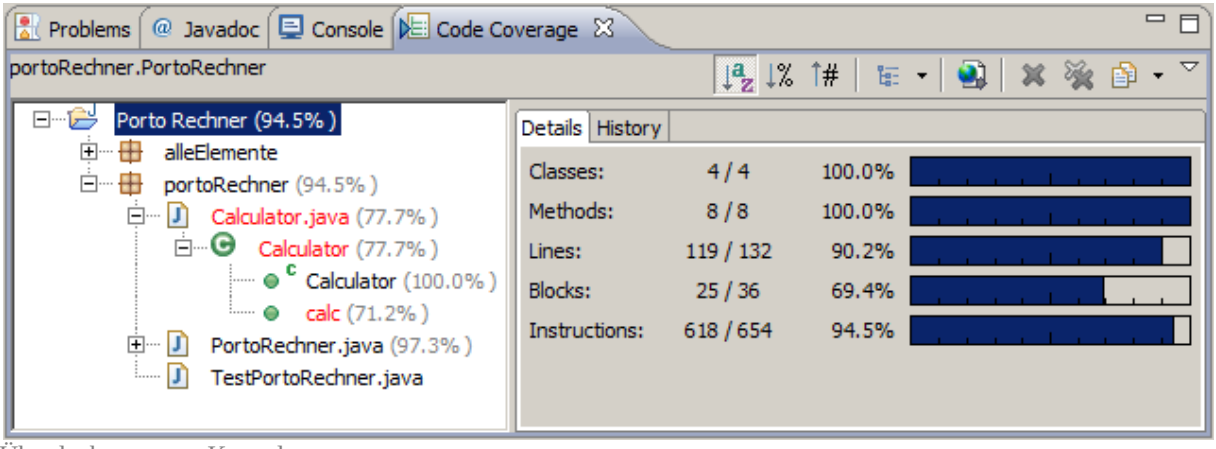
Der Google WindowTester Pro bietet neben den Oberflächentests auch die Möglichkeit von einfachen Überdeckungstests.

Um diese Funktion zu nutzen, muss sie für die Java-Klasse, welche die Main-Methode, Code-Coverage aktiviert werden.



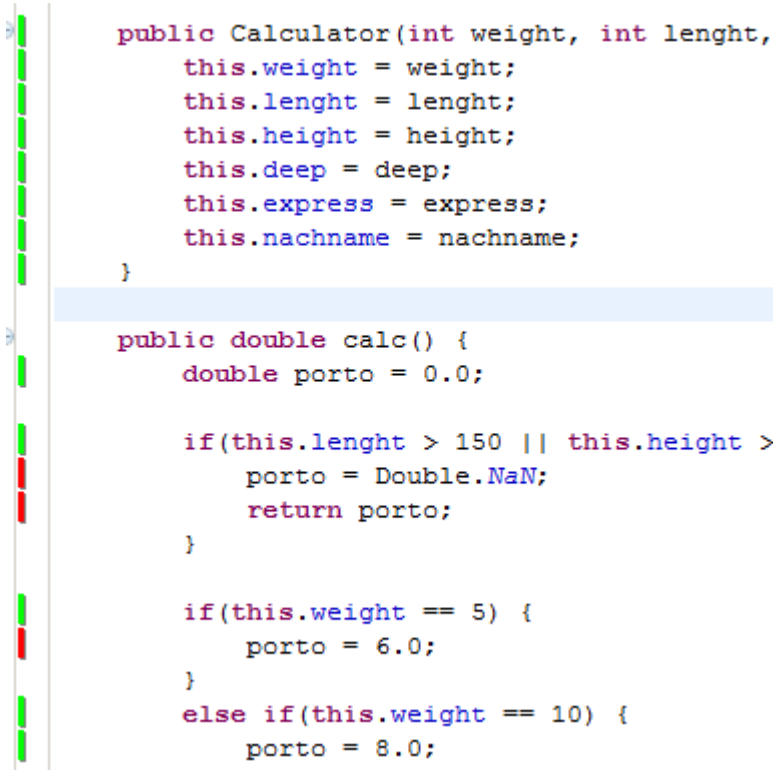
Überdeckungstest aktivieren

Nun wird das Programm gestartet und es kann ganz normal genutzt werden. Wurde das Programm beendet, so kann man nun in der Code Coverage Console sehen, wie viel Prozent der Codezeilen in den jeweiligen Methoden ausgeführt wurden. Klassen und Methoden bei denen weniger als 80% ausgeführt wurden, werden rot markiert. Dieser Schwellwert lässt sich über eine Einstellung anpassen.



Überdeckungstests Konsole

Im Quellcode werden die Zeilen, welche ausgeführt wurden, grün und die nicht ausführten rot markiert. Es ist jedoch nicht ersichtlich, welche Atome einer if-Abfrage berücksichtigt wurden.



```
    public Calculator(int weight, int lenght,
        this.weight = weight;
        this.lenght = lenght;
        this.height = height;
        this.deep = deep;
        this.express = express;
        this.nachname = nachname;
    }

    public double calc() {
        double porto = 0.0;

        if(this.lenght > 150 || this.height >
            porto = Double.NaN;
            return porto;
        }

        if(this.weight == 5) {
            porto = 6.0;
        }
        else if(this.weight == 10) {
            porto = 8.0;
```

Markierung der benutzten Quellcodezeilen

Vergleich mit Konkurrenzprodukten

Die Code Coverage Funktion des WindowTesters bietet nicht den Funktionsumfang wie z.B. CodeCover (http://www.codecover.org). Um zu sehen welcher Code Ausgeführt wurde, reicht aber der WindowTester aus. Sind jedoch weiterreichende Funktionen gewünscht, ist der Einsatz anderer Tools wie z.B. CodeCover notwendig.

Dokumentation / Projektpflege

Die nur in englischer Sprache verfügbare Dokumentation ist über die Hilfe Funktion von Eclipse sowie auch Online erreichbar. Sie bieten einen guten Überblick über die Funktionen des Google WindowTester Pro, könnte jedoch mehr Beispiele beinhalten. Um effektive Tests zu schreiben ist ein genauer Blick in die API-Reference unabdingbar.

Unter der Adresse http://forums.instantiations.com/viewforum.php?f=5 ist ein Support- und Diskussionsforum des Herstellers erreichbar, ansonsten ist nicht sehr viel im Internet zu finden.

Zum jetzigen Zeitpunkt sind die Projektseiten aktuell, wie die Weiterentwicklung nach der Übernahme durch Google vorangetrieben wird bleibt abzuwarten. Es ist jedoch zu erwarten, dass durch die seit der Übernahme kostenlose Verfügbarkeit die Community wächst.

Fazit

Der Google WindowTester Pro ist ein sehr mächtiges Werkzeug zum Testen von GUIs. Es ist allerdings eine gewisse Einarbeitung und ein genauer Blick in die Dokumentation, insbesondere der API-Dokumentation nötig, um möglichst effektive und leistungsfähige Tests zu schreiben.

Die verschiedenen Locators bieten umfangreiche Möglichkeiten einzelne Komponenten der GUI zu finden. So lassen sich z.B. auch Komponenten mit der gleichen Beschriftung finden.

Die unterschiedlichen Conditions bieten eine gute Möglichkeit z.B. Eigenschaften von Steuerelementen zu prüfen und darauf zu reagieren. Der Umfang der Locators und Conditions ist jedoch stark abhängig vom verwendeten GUI-Toolkit.

Mit der Aufnahmefunktion des Recorders können Klickfolgen schnell und komfortabel aufgezeichnet werden. Die eigentlichen Testfälle sollten aber manuell aufschrieben werden. Funktionen, wie der Inspector, mit denen sich automatisch Asserts erzeugen lassen, sind zum aktuellen Entwicklungszeitpunkt nur mäßig zu gebrauchen. Sie bieten aber viel Potential, um in kommenden Versionen ausgebaut zu werden. So wäre es denkbar, dass durch Klicken Asserts erzeugt werden, welche auf erwartete Ausgaben prüfen.

Das die Software nur als Erweiterung für Eclipse verfügbar ist schränkt das Einzugsgebiet an Benutzern natürlich ein, bietet dem User aber auch viele Vorteile, da keine neue Anwendung erlernt werden muss, sondern einfach die gewohnte Umgebung erweitert wird. Außerdem können alle Vorzüge der IDE genutzt werden.

Trotz des großen Funktionsumfangs des WindowTesters wird es in größeren Softwareprojekten immer Testfälle geben, die von diesem nicht abgebildet werden können. Dabei kann es sich beispielsweise um nicht steuerbare GUI Elemente oder dem Wechsel zwischen Fenstern handeln. Irgendwo werden aber alle Oberflächentestprogramme an ihre Grenzen stoßen. Für möglichst einfache und effiziente Tests sollte schon während der Entwicklungsphase ein Blick auf die Einschränkungen und Besonderheiten bei der Testserstellung geachtet werden.

Die vom Entwicklerteam geplante Umrüstung von JUnit 3 auf 4 lässt auf komfortablere Testformulierbarkeit in Zukunft hoffen.

Wer allerdings ein Tool für Überdeckungstests sucht, sollte auf andere dafür spezialisierte Programme zurückgreifen, da man mit dem vom WindowTester recht beschränkten Funktionsumfang auf diesem Gebiet schnell an die Grenzen der Möglichkeiten stößt.

Diese Hausarbeit kann nur einen kleinen Überblick über die Funktionen des WindowTesters bieten. So wurden bei den Beispielen nur Swing basierende Oberflächen betrachtet. SWT und Eclipse/RCP Oberflächen wurden nicht gezeigt, die Beispiele sind aber ähnlich hierauf anzuwenden.

Quellenangaben

- http://code.google.com/intl/de/javadevtools/wintester/html/index.html
- http://forums.instantiations.com/