HANSER

Pragmatisch Programmieren

Projekt-Automatisierung

von Michael Clark

ISBN 3-446-40008-7

Leseprobe Kapitel 5.4

Weitere Informationen oder Bestellung
unter http://www.hanser.de/3-446-40008-8
sowie im Buchhandel

5.4 Fehlerbekämpfung mit Diagnosetests

Wenn dasselbe Installationsproblem bei mehreren Nutzern auftaucht, versucht es uns damit etwas zu sagen. Der Installationsprozess könnte zu kompliziert sein, um ihm genau folgen zu können. Er sollte daher im nächsten Release vereinfacht werden. Aber wir sind mit unserem Job immer noch nicht fertig. Unbemerkte Konfigurationsänderungen nach der Installation, wie wir sie bei der Umgebungsvariable `JAVA_HOME` in der Geschichte vom technischen Support erlebt haben, können Probleme verursachen. Die richtige Java-Version mit unserer Anwendung auszuliefern, wäre eine Möglichkeit, aber das schützt uns nicht vor menschlichen Fehlern. Diese können wir nur mit guter Automatisierung minimieren.

Die übliche Reaktion ist, allen Nutzern eine Checkliste zum Fehlersuchen zu geben, eine Liste der häufigsten Installationsprobleme zusammenzustellen und dann auf das Beste zu hoffen. Aber das können wir besser. So wie es hieß, als der Sechs-Millionen-Dollar-Mann gebaut wurde: „Wir haben die Technologie.“ Und wahrhaftig schreit unsere Checkliste zur Fehlersuche nach Automatisierung in Form von Diagnosetests.

Einen Diagnosetest schreiben

Sie benutzen bereits JUnit zum Schreiben von Unittests für Ihre Anwendung. Diese Tests vergleichen ein erwartetes Ergebnis mit dem tatsächlichen Wert und wenn die Werte nicht übereinstimmen, schlägt der Test fehl. Wenn ein Unittest fehlschlägt, wissen Sie, dass etwas in Ihrem Code außer Kontrolle geraten ist. Unglücklicherweise erfahren Sie das von einer Nutzerinstallation nur, wenn das Telefon klingelt.

```
package com.pragprog.dms.selftest;

import java.io.File;
import java.io.IOException;
import junit.framework.TestCase;

public class InstallTests extends TestCase {

  public void testJavaVersion() {
    String version = System.getProperty("java.version");
    assertTrue("Incompatible Java version. " +
      "Requires version 1.4.x, but found " + version,
      version.startsWith("1.4"));
  }

  public void testIndexDirectory() throws IOException {
    File dmsDir = getDirectory("dms.dir");
    File indexDir = new File(dmsDir, "index");
    assertDirectoryExists(indexDir);
    assertDirectoryReadable(indexDir);
    assertDirectoryWritable(indexDir);
  }

  void assertDirectoryExists(File dir) throws IOException {
    assertNotNull(dir);
    assertTrue("Directory doesn't exist: " + dir.getCanonicalPath(),
      dir.exists());
  }

  void assertDirectoryReadable(File dir) throws IOException {
    assertTrue("Directory not readable: " + dir.getCanonicalPath(),
      dir.canRead());
  }

  void assertDirectoryWritable(File dir) throws IOException{
    assertTrue("Directory not writable: " + dir.getCanonicalPath(),
      dir.canWrite());
  }

  File getDirectory(String propertyName) {
    String dirName = System.getProperty(propertyName);
    assertNotNull("'" + propertyName + "' not defined", dirName);
    return new File(dirName);
  }
}
```

dms/test/com/pragprog/dms/selftest/InstallTests.java

Abbildung 5.1: Diagnosetest einer Installation

Was wäre, wenn Sie JUnit dazu bringen könnten, dem Nutzer bei der Diagnose zu helfen, wann und warum etwas mit seiner Installation schief gelaufen ist? Abbildung 5.1 auf Seite 113 zeigt den JUnit-Test `InstallTests` mit zwei Testmethoden.

- `testJavaVersion` überprüft die System-Property `java.version` und stellt sicher, dass die Java-Version 1.4.*x* läuft.
- `testIndexDirectory` lokalisiert das Installationsverzeichnis mit Hilfe der System-Property `dms.dir`, die dem Test beim Aufruf mitgegeben wird. Dann prüft es, ob das Unterverzeichnis `index` existiert und mit Lese- und Schreibrechte für den aktuellen Nutzer versehen ist.

Dieser einfache Diagnosetest sieht vielleicht nicht nach viel aus. Er überprüft eine System-Property und ein Verzeichnis. Und trotzdem kann er all die Probleme diagnostizieren, die während des Telefongespräches mit dem technischen Support entdeckt wurden, und das *viel* schneller. Jetzt benötigen Sie nur noch einen Weg, den Diagnosetest im Bedarfsfall an den Nutzer auszuliefern.

Verteilen der Diagnosetests

Wenn Anwender Probleme haben, brauchen Sie eine Möglichkeit, die Diagnosetests in die fehlerhafte Installation einzuschleusen. Eine Möglichkeit ist, die Tests jeder Standarddistribution hinzuzufügen, aber je nach Größe der Distributionsdatei und der Häufigkeit von Problemen, könnte das auch zuviel des Guten sein.

Eine Alternative ist das Bereitstellen einer speziellen Testdistributionsdatei für Kunden, die anrufen. Die Release-Prozedur aus dem letzen Kapitel ermöglicht uns genau das. Sie erstellt eine Datei `dms-1_0-tests.zip`, die alle Tests aus unserem Verzeichnis `build/test` und ein Skript zum Ausführen enthält. Beide Dateien werden zusätzlich zur normalen Distributionsdatei `dms-1_0.zip` erstellt.

Sobald ein Nutzer ein Problem bemerkt, bitten Sie ihn, die Testdistributionsdatei herunterzuladen und sie über seine Standarddistribution zu installieren. Wenn er in seinem Nutzerverzeichnis die Version 1.0 von *DMS* installiert hat, dann würde er die Datei `dms-1_0-tests.zip` in seinem Nutzerverzeichnis so entpacken:

```
$ cd $HOME
$ unzip dms-1_0-tests.zip
Archive: dms-1_0-tests.zip
  inflating: dms-1_0/bin/selftest.bat
  inflating: dms-1_0/bin/selftest.sh
  inflating: dms-1_0/lib/dms-tests.jar
  inflating: dms-1_0/vendor/lib/junit-3.8.jar
```

Da der Inhalt dieser ZIP-Datei im gleichen Verzeichnis wie der der Standarddistribution liegt, werden die Testdateien in das Verzeichnis der Standarddistribution entpackt. Stellen Sie sich das Ganze als Hinzufügen einer Testschicht zur Standarddistribution vor, wie Abbildung 5.2 verdeutlicht.

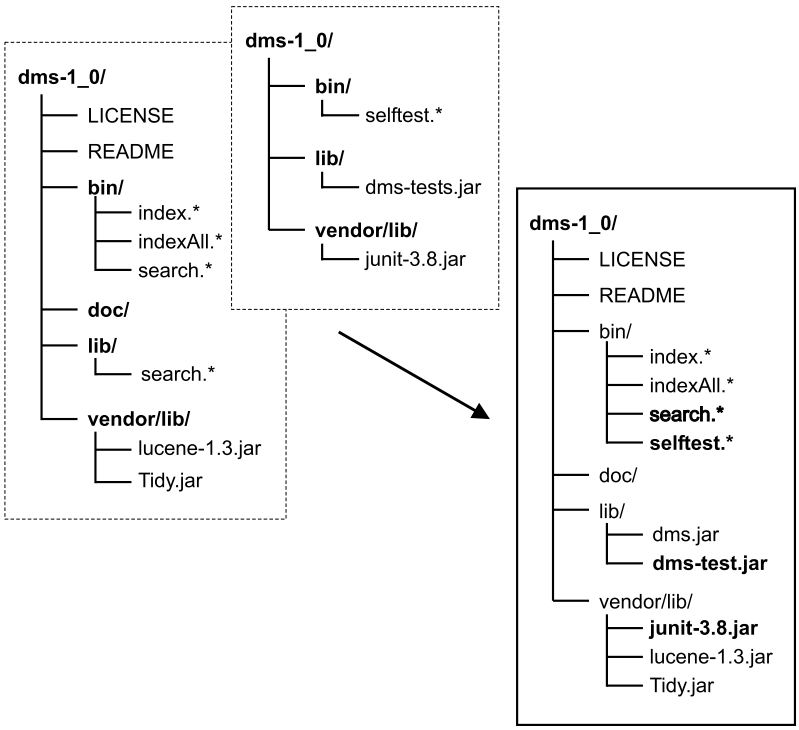

Abbildung 5.2: Hinzufügen von Tests zur Standarddistribution

Ausführen der Diagnosetests

Der Nutzer ist jetzt so weit, die Diagnosetests auf seine Installation loszulassen. Dafür startet er das Skript oder die Batch-Datei `selftest` aus der Testdistributionsdatei.

```
$ sh bin/selftest.sh
.F.F
Time: 0.092
There were 2 failures:

1) testJavaVersion(com.pragprog.dms.selftest.InstallTests)
   junit.framework.AssertionFailedError:
   Incompatible Java version. Requires version 1.4.x, but found 1.3.1

2) testIndexDirectory(com.pragprog.dms.selftest.InstallTests)
   junit.framework.AssertionFailedError:
   Directory not readable: /Users/somebody/dms-1_0/index

FAILURES!!!
Tests run: 2,  Failures: 2,  Errors: 0
```

Aha! Der Test bringt ans Licht, dass Java in der Version 1.3.1 verwendet wird, obwohl Version 1.4.*x* Voraussetzung ist. Und wenn der Diagnosetest selbst nicht gestartet werden kann, weil *gar keine* Version von Java gefunden wurde, kann `selftest.sh` den Nutzer höflich darüber informieren. Außerdem hat dieser Nutzer keine Leserechte für das Verzeichnis `index`. So wird ein 15-minütiges Telefongespräch auf einen Knopfdruck und das Ablesen des Ergebnisses reduziert.

Wenn die Diagnoseergebnisse leicht verständlich sind, können wir hoffen, dass der Nutzer die richtigen Fehlerbehebungsmaßnahmen ergreift. Wenn das nicht hilft, sendet er Ihnen die Diagnoseergebnisse zur Analyse per E-Mail. Aber selbst das kostet Sie immer noch weniger Zeit, als die Problemlösung am Telefon.

Sie wissen, dass die Anwendung wieder in sicheren Fahrwassern ist, wenn der Nutzer nach dem Ausführen der Tests die sehnsüchtig erwartete Meldung `OK` sieht.

```
$ sh bin/selftest.sh
..
Time: 0.082

OK (2 tests)
```

Wenn Sie diese Diagnosetests einsetzen, haben Sie quasi einen Mitarbeiter vom technischen Support zum Kunden geschickt. Dabei arbeiten die Tests genau wie Sie am Telefon nur die Checkliste ab.

Schreiben einer Sammlung von Diagnosetests

Die zwei Tests in `InstallTests` können selbstverständlich nicht alle möglichen Installationsprobleme erkennen. Aber zum jetzigen Zeitpunkt decken sie alle uns bekannten, wiederkehrenden Installationsprobleme ab. Was passiert nun, wenn ein neues Installationsproblem von mehreren Nutzern gemeldet wird? Sie fügen einfach weitere Diagnosetests hinzu. Natürlich müssen Sie aufpassen, dass die Tests von nichts abhängen, was ihre Ausführung verhindern könnte. Installationsfragen, die von Tests beantwortet werden können, sind:

- Sind alle notwendigen Dateien und Verzeichnisse in der richtigen Reihenfolge im Klassenpfad (System-Property `java.class.path`) enthalten?
- Kann die Anwendung eine Verbindung zur Datenbank herstellen?
- Wurde die WAR- oder EAR-Datei richtig konfiguriert und auch im richtigen Verzeichnis installiert?
- Ist ein erforderliches System oder eine notwendige URL erreichbar?

Wenn Sie neue Tests schreiben, erstellen Sie wahrscheinlich neue JUnit-Testfälle. Damit das gemeinsame Ausführen aller Diagnosetests für den Nutzer kein Problem darstellt, fassen Sie alle Tests in einer JUnit-Testsuite zusammen. In Abbildung 5.3 auf Seite 118 sehen Sie die Testsuite `AllTests`, die einen neuen Testfall `ClasspathTests` beinhaltet.

Die Methode `suite` fügt die Testfälle `InstallTests` und `ClasspathTests` zur Testsuite hinzu. So kann der Nutzer durch Starten des Skriptes `selftest.sh` alle Diagnosetests der Installation ausführen.

```
$ ./bin/selftest.sh
...
Time: 0.075

OK (3 tests)
```

Warum sollten wir an dieser Stelle aufhören? Wenn alle Installationsdiagnosetests erfolgreich durchlaufen, der Nutzer aber weiterhin ein Problem hat, sind weitere Tests sinnvoll. Und da Sie schon Zeit in das Schreiben von Unittests gesteckt haben, die Fehler beim Ent-

```
package com.pragprog.dms.selftest;

import junit.framework.Test;
import junit.framework.TestSuite;

public class AllTests {

  public static Test suite() {
    TestSuite suite = new TestSuite("DMS Selftests");
    suite.addTestSuite(InstallTests.class);
    suite.addTestSuite(ClasspathTests.class);
    // Add more diagnostic tests here
    return suite;
  }

  public static void main(String[] args) {
    junit.textui.TestRunner.run(suite());
  }
}
```

dms/test/com/pragprog/dms/selftest/AllTests.java

Abbildung 5.3: Testsuite mit allen Diagnosetests

wickeln finden sollen, können Sie diese ebenso beim Nutzer ausführen lassen. Sie müssen den Nutzer nur auffordern, die Suite `AllUnitTests` mit dem Skript `selftest.sh` und einer zusätzlichen Option auszuführen oder ein anderes Skript zu starten.

Jetzt haben wir eine Testsammlung, die ihren Aufwand wert ist. Jedes Mal, wenn sie bei einem Nutzer ausgeführt wird, testet sie automatisch eine Menge wichtiger Dinge. Und wenn etwas aus der Reihe tanzt, schlägt sie mit einem detaillierten Bericht über das Problem fehl. Das Beste daran ist: unabhängig davon, wie viele Tests wir hinzugefügt haben, startet sie der Nutzer alle mit einem einzigen Befehl.

5.5 Aufbereiten Ihres Installationsprozesses

Sind wir doch mal ehrlich, das Entpacken einer Distributionsdatei ist kein ansprechender Installationsprozess. Er erfüllt seinen Zweck, aber er fällt wahrscheinlich nicht so positiv auf, dass unsere Software sich dadurch von der Masse abhebt. Der Prozess ist außerdem nicht erweiterbar – wir haben keine Möglichkeit, nach dem Auspacken noch weitere Installationsschritte zu ergänzen. Wenn die ZIP-