Testen

Er bestaan verschillende veel gebruikte tools en frameworks om de kwaliteit van Java-code te meten (Checkstyle, FindBugs, Sonar, etcetera). Ook voor het meten van de testcoverage zijn diverse tools beschikbaar (JUnit, Cobertura, etcetera). Sinds kort is er een vergelijkbare tool om de traceability tussen de requirements en code te meten: JavaRequirementsTracer. Deze tool maakt het mogelijk om heel makkelijk en nauwkeurig de traceability van een softwaresysteem in kaart te brengen.

Komen de requirements wel in de code terug?

JavaRequirementsTracer is een handige testhulp

Ronald Koster
is Software Architect bij Logica. Hij is te bereiken via ronald.koster@logica.com.

Ruud de Jong
is Software Architect bij Logica. Hij is te bereiken via ruud.a.de.jong@logica.com.

Requirements traceability wordt vaak vergeten, of is een heel moeizaam handmatig proces wat gedoemd is te mislukken. Terwijl het eigenlijk misschien wel de belangrijkste van alle bovengenoemde kwaliteitsaspecten is, immers waarom bouwt men het systeem überhaupt?
Als we het hebben over requirements-traceability, dan praten we over de mogelijkheid om code te relateren aan de requirements waarvoor die code geschreven is en vice versa. Vaak wordt zoiets gedaan met een traceability-matrix.
Zo'n matrix wordt al snel erg groot met veel lege cellen. Daarom is het meestal handiger om twee afbeeldingstabellen te hanteren: één van code-elementen naar requirementslabels en één van labels naar code-elementen. De labels verwijzen naar een (stuk van een) bepaalde requirement, bijvoorbeeld UC-Zoeken-zaak, UC-Aanmaken-zaak.
Een requirement bestaat meestal uit een stuk tekst, bijvoorbeeld een Use Case beschrijving. De labels verwijzen naar die tekst of stukken daarvan. Die stukken zijn dan bijvoorbeeld gemarkeerd met dezelfde labels. De fijnmazigheid van markeren is naar eigen inzicht. Er zijn geen harde regels voor.

Er zijn twee maten te definiëren die samenhangen met traceability:

- requirementscoverage = percentage requirements dat gebouwd is
- codecoverage = percentage code dat naar een requirement te traceren is

NB. Verwar codecoverage niet met testcoverage wat ook vaak codecoverage wordt genoemd.

Bij een systeem dat af is horen beide maten op 100% te staan. De eerste omdat alle requirements gebouwd zouden moeten zijn. Deze geeft dus een maat voor hoe ver het systeem af is. De tweede omdat er alleen code hoort te zijn gebouwd ten behoeve van de requirements. Alle andere code is overbodig. Deze geeft dus een maat voor hoeveel overbodige code er is. Een kwaliteitsindicator dus.

JavaRequirementsTracer

De JavaRequirementsTool genereert automatisch traceability rapporten voor Java-code. Hij werkt op basis van Java-annotaties. Annoteer de 'public API' code van het systeem met @Requirements annotaties. Die annotaties bevatten de requirementslabel(s) die geïmplementeerd worden door dat stukje code. Met 'public API' worden alle public en protected types en methodes van de presentatie- en servicelaag van het softwaresysteem (ook wel applicatie) bedoeld. Algemene types die niet direct aan een requirement zijn te koppelen (meestal utility en helper klassen) kunnen geannoteerd worden met @SuppressTraceabilityWarnings om de rapporten niet te vervuilen. In figuur 1 staat een voorbeeld van geannoteerde code.

```
/**
 * Class for generating a traceability report. It scans
 * for {@link Requirements} annotations.
 * @see #init()
 *
 * @author Ronald Koster
 */
@Requirements("UC-Generate-Report")
public final class JavaRequirementsTracerBean {
```

Figuur 1: een stukje code met een @Requirements annotatie.

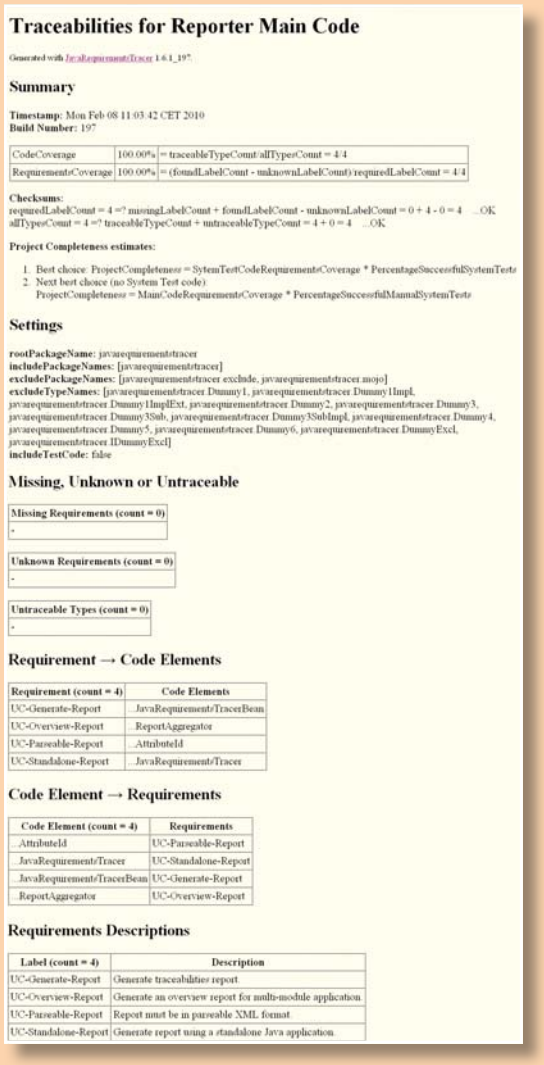
Alleen de package API klassen en methodes moeten public of protected zijn.

De code in figuur 1 is van JavaRequirementsTracer zelf. Die is ook geannoteerd en kan daardoor als voorbeeld dienen. JavaRequirementsTracer kan vervolgens een traceabilityrapport voor de applicatie genereren zoals weergegeven in figuur 2.

In het rapport staan onder andere de gemeten requirements- en codecoverage, twee afbeeldingstabellen (van requirements naar code en vice versa) en een legenda voor de labels. De volgende tabellen staan ook in het rapport: 'Missing Requirements', 'Unknown Requirements' en 'Untraceable Types'. Deze drie tabellen zijn in feite de to-do-lijst om tot 100% requirements- en codecoverage te komen. De eerste geeft aan welke requirementslabels blijkbaar nog niet gebouwd zijn. De tweede geeft meestal typefouten in labels of afgeschafte labels weer. De derde geeft schijnbaar overbodige of onterecht zichtbare code weer. Met name dat laatste is vaak het geval. Veel programmeurs maken alle klassen en methodes public, terwijl alleen de package API klassen en methodes public of protected zouden moeten zijn. De meeste klassen in een package zijn implementatiedetails en horen hoogstens package-private zichtbaarheid te hebben. Op deze wijze test de tool of er voldoende inkapseling is. Iets wat veel andere tools niet kunnen.

Combinatie met systeemtests

Een zeer goede methode om te testen of een systeem aan de gestelde requirements voldoet is door het maken van automatische systeemtests. Allereerst het verschil tussen unit- en systeemtests. Een unittest is een 'white box test': hij test een specifiek stuk code, meestal niet via de public API en meestal niet direct gerelateerd aan een requirement. De code van een unittest staat in dezelfde package als de code die hij test. Een systeemtest is een 'black box test': hij test via de public API of de code aan een bepaalde requirement voldoet. De code van een systeemtest staat daarom expres in een andere package dan de code die hij test, bijvoorbeeld 'mysystemname.systemtest'.



Traceabilities for Reporter Main Code

Generated with JavaRequirementsTracer 1.6.1_197.

Summary

Timestamp: Mon Feb 08 11:03:42 CET 2010
Build Number: 197

CodeCoverage	100.00%	= traceableTypeCount/allTypesCount = 4/4
RequirementsCoverage	100.00%	= (foundLabelCount - unknownLabelCount)/requiredLabelCount = 4/4

Checksums:
requiredLabelCount = 4 =? missingLabelCount + foundLabelCount - unknownLabelCount = 0 + 4 - 0 = 4 ...OK
allTypesCount = 4 =? traceableTypeCount + untraceableTypeCount = 4 + 0 = 4 ...OK

Project Completeness estimates:

1. Best choice: ProjectCompleteness = SytemTestCodeRequirementsCoverage * PercentageSuccessfulSystemTests
2. Next best choice (no System Test code):
   ProjectCompleteness = MainCodeRequirementsCoverage * PercentageSuccessfulManualSystemTests

Settings

rootPackageName: javarequirementstracer
includePackageNames: [javarequirementstracer]
excludePackageNames: [javarequirementstracer.exclude, javarequirementstracer.mojo]
excludeTypeNames: [javarequirementstracer.Dummy1, javarequirementstracer.Dummy1Impl, javarequirementstracer.Dummy1ImplExt, javarequirementstracer.Dummy2, javarequirementstracer.Dummy3, javarequirementstracer.Dummy3Sub, javarequirementstracer.Dummy3SubImpl, javarequirementstracer.Dummy4, javarequirementstracer.Dummy5, javarequirementstracer.Dummy6, javarequirementstracer.DummyExcl, javarequirementstracer.IDummyExcl]
includeTestCode: false

Missing, Unknown or Untraceable

Missing Requirements (count = 0)

-

Unknown Requirements (count = 0)

-

Untraceable Types (count = 0)

-

Requirement → Code Elements

Requirement (count = 4)	Code Elements
UC-Generate-Report	...JavaRequirementsTracerBean
UC-Overview-Report	...ReportAggregator
UC-Parseable-Report	...AttributeId
UC-Standalone-Report	...JavaRequirementsTracer

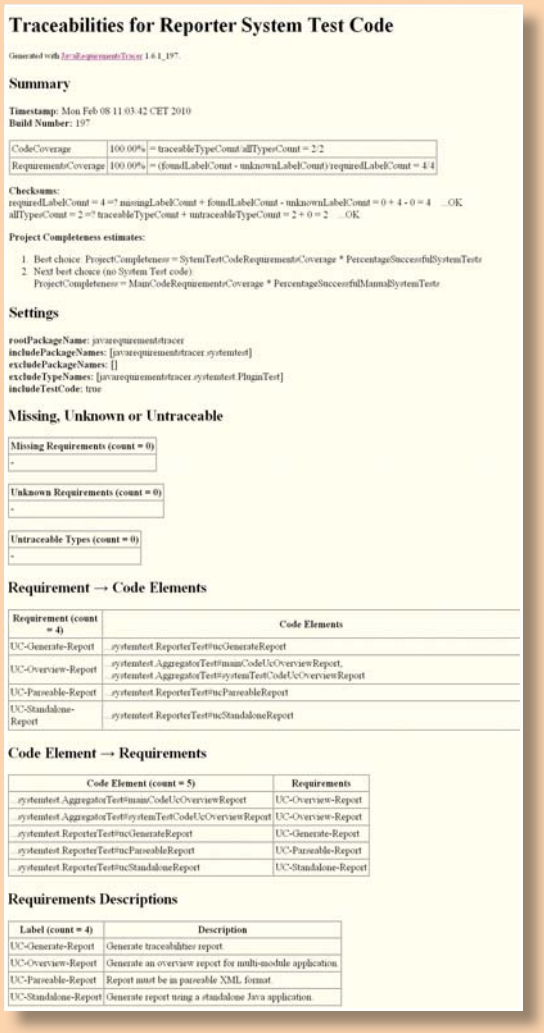
Code Element → Requirements

Code Element (count = 4)	Requirements
...AttributeId	UC-Parseable-Report
...JavaRequirementsTracer	UC-Standalone-Report
...JavaRequirementsTracerBean	UC-Generate-Report
...ReportAggregator	UC-Overview-Report

Requirements Descriptions

Label (count = 4)	Description
UC-Generate-Report	Generate traceabilities report.
UC-Overview-Report	Generate an overview report for multi-module application.
UC-Parseable-Report	Report must be in parseable XML format.
UC-Standalone-Report	Generate report using a standalone Java application.

Figuur 2: een voorbeeld van een traceabilityrapport.



Traceabilities for Reporter System Test Code

Generated with JavaRequirementsTracer 1.6.1_197.

Summary

Timestamp: Mon Feb 08 11:03:42 CET 2010
Build Number: 197

CodeCoverage	100.00%	= traceableTypeCount/allTypesCount = 2/2
RequirementsCoverage	100.00%	= (foundLabelCount - unknownLabelCount)/requiredLabelCount = 4/4

Checksums:
requiredLabelCount = 4 =? missingLabelCount + foundLabelCount - unknownLabelCount = 0 + 4 - 0 = 4 ...OK
allTypesCount = 2 =? traceableTypeCount + untraceableTypeCount = 2 + 0 = 2 ...OK

Project Completeness estimates:

1. Best choice: ProjectCompleteness = SytemTestCodeRequirementsCoverage * PercentageSuccessfulSystemTests
2. Next best choice (no System Test code):
   ProjectCompleteness = MainCodeRequirementsCoverage * PercentageSuccessfulManualSystemTests

Settings

rootPackageName: javarequirementstracer
includePackageNames: [javarequirementstracer.systemtest]
excludePackageNames: []
excludeTypeNames: [javarequirementstracer.systemtest.PluginTest]
includeTestCode: true

Missing, Unknown or Untraceable

Missing Requirements (count = 0)

-

Unknown Requirements (count = 0)

-

Untraceable Types (count = 0)

-

Requirement → Code Elements

Requirement (count = 4)	Code Elements
UC-Generate-Report	...systemtest.ReporterTest#ucGenerateReport
UC-Overview-Report	...systemtest.AggregatorTest#mainCodeUcOverviewReport, ...systemtest.AggregatorTest#systemTestCodeUcOverviewReport
UC-Parseable-Report	...systemtest.ReporterTest#ucParseableReport
UC-Standalone-Report	...systemtest.ReporterTest#ucStandaloneReport

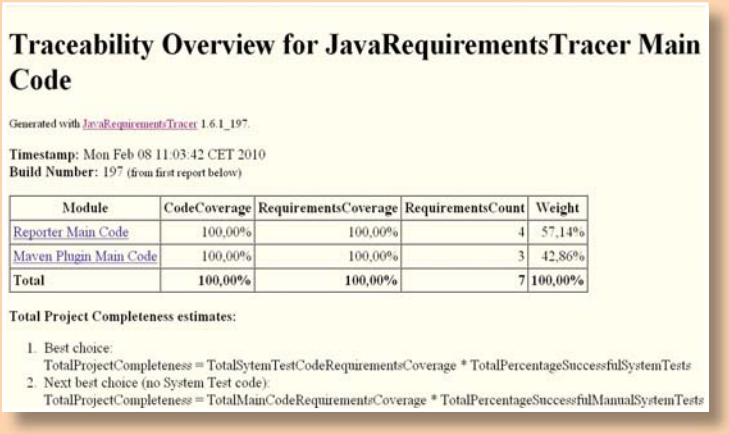
Code Element → Requirements

Code Element (count = 5)	Requirements
...systemtest.AggregatorTest#mainCodeUcOverviewReport	UC-Overview-Report
...systemtest.AggregatorTest#systemTestCodeUcOverviewReport	UC-Overview-Report
...systemtest.ReporterTest#ucGenerateReport	UC-Generate-Report
...systemtest.ReporterTest#ucParseableReport	UC-Parseable-Report
...systemtest.ReporterTest#ucStandaloneReport	UC-Standalone-Report

Requirements Descriptions

Label (count = 4)	Description
UC-Generate-Report	Generate traceabilities report.
UC-Overview-Report	Generate an overview report for multi-module application.
UC-Parseable-Report	Report must be in parseable XML format.
UC-Standalone-Report	Generate report using a standalone Java application.

Figuur 3: een voorbeeld van een rapport van de systeemtestcode.



Traceability Overview for JavaRequirementsTracer Main Code

Generated with JavaRequirementsTracer 1.6.1_197.

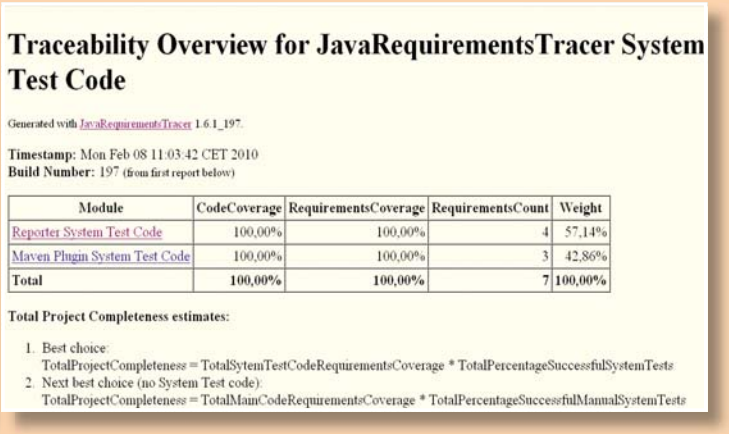
Timestamp: Mon Feb 08 11:03:42 CET 2010
Build Number: 197 (from first report below)

Module	CodeCoverage	RequirementsCoverage	RequirementsCount	Weight
Reporter Main Code	100,00%	100,00%	4	57,14%
Maven Plugin Main Code	100,00%	100,00%	3	42,86%
Total	**100,00%**	**100,00%**	**7**	**100,00%**

Total Project Completeness estimates:

1. Best choice:
   TotalProjectCompleteness = TotalSytemTestCodeRequirementsCoverage * TotalPercentageSuccessfulSystemTests
2. Next best choice (no System Test code):
   TotalProjectCompleteness = TotalMainCodeRequirementsCoverage * TotalPercentageSuccessfulManualSystemTests

Figuur 4: overzichtsrapport van de main-code.



Traceability Overview for JavaRequirementsTracer System Test Code

Generated with JavaRequirementsTracer 1.6.1_197.

Timestamp: Mon Feb 08 11:03:42 CET 2010
Build Number: 197 (from first report below)

Module	CodeCoverage	RequirementsCoverage	RequirementsCount	Weight
Reporter System Test Code	100,00%	100,00%	4	57,14%
Maven Plugin System Test Code	100,00%	100,00%	3	42,86%
Total	**100,00%**	**100,00%**	**7**	**100,00%**

Total Project Completeness estimates:

1. Best choice:
   TotalProjectCompleteness = TotalSytemTestCodeRequirementsCoverage * TotalPercentageSuccessfulSystemTests
2. Next best choice (no System Test code):
   TotalProjectCompleteness = TotalMainCodeRequirementsCoverage * TotalPercentageSuccessfulManualSystemTests

Figuur 5: overzichtsrapport van de systeemtestcode.

Het JUnit framework is ondanks de naam ook geschikt voor het maken van systeemtests. Een groot voordeel hiervan is bovendien dat de systeemtests dan met elke build meelopen. Systeemtesten kan helaas niet altijd buiten de webcontainer. Maar voor in-container testen kun je bijvoorbeeld gebruik maken van het JWebUnit framework. Op Java gebaseerde systeemtestcode kan vervolgens ook geannoteerd worden met requirementslabels. Een goed getest systeem heeft 100% requirements- en codecoverage voor zowel de main- als de systeemtestcode. In figuur 3 staat een voorbeeld van een traceabilityrapport van systeemtestcode.

De requirementscoverage van de systeemtestcode geeft aan in hoeverre het systeem getest wordt. Er is nu een nog betere schatting te geven voor hoe ver het systeem af is:
*compleetheid = systeemtestrequirementscoverage * systeemtestsuccespercentage*

Geavanceerde opties

De JavaRequirementsTracer tool wordt geconfigureerd met twee propertiesbestanden: één met de parameters voor een rapport en één met labels die verwacht worden. Met het parametersbestand kun je configureren welke code wel en niet wordt gescand. Het scannen gebeurt via het classpath. Per parametersbestand ontstaat één rapport. Main- en systeemtestcode hebben daarom elk een eigen parametersbestand, maar natuurlijk wel een gedeeld labelsbestand. Als het softwaresysteem uit meerdere modules (subsystemen) bestaat, kun je per module een parameters- en labelsbestand maken. Het is ook mogelijk om een overzichtsrapport te maken die de informatie van de rapporten van de verschillende modules samenvat. Voorbeelden van zulke overzichtsrapporten staan in figuur 4 en 5.

De eenvoudigste manier om de rapporten te genereren is met behulp van de bijbehorende Maven-plugin, maar de rapportgenerators kunnen ook als los Java-programma (en dus ook via Apache Ant) of als unittest draaien. Het genereren van een detail- of overzichtsrapport duurt meestal slechts enkele seconden.

De gegenereerde rapporten zijn in XHTML-formaat. Ze zijn daardoor met een standaardbrowser te lezen, maar ook gemakkelijk door een tool te verwerken (parsen). Op die manier kan de informatie van een rapport ook gebruikt worden door andere tools, bijvoorbeeld door Sonar (een populaire tool voor het meten van softwarekwaliteit). Maar momenteel bestaat er nog geen Sonar-plugin voor JavaRequirementsTracer.

Conclusie

Met de JavaRequirementsTracer tool kan de traceability van een softwaresysteem eenvoudig worden opgezet en nauwkeurig worden gemeten. Dit helpt de programmeurs bij het bewaken van de codekwaliteit en de projectmanager bij het bewaken van de compleetheid en daarmee de voortgang. Als de systeemtestcode ook is gebaseerd op Java-code kan die code op dezelfde wijze gerelateerd worden aan de requirements. De testers kunnen dan een nauwkeurige uitspraak doen in hoeverre het systeem ge(systeem)test wordt. Bovendien is er dan een nog betere schatting voor de compleetheid van het systeem te geven. «

JavaRequirementsTracer helpt zowel de programmeur als de manager van een project.

Referenties

- JavaRequirementsTracer: http://reqtracer.sourceforge.net
- JUnit: http://www.junit.org
- JWebUnit: http://jwebunit.sourceforge.net
- Maven: http://maven.apache.org
- Sonar: http://www.sonarsource.org