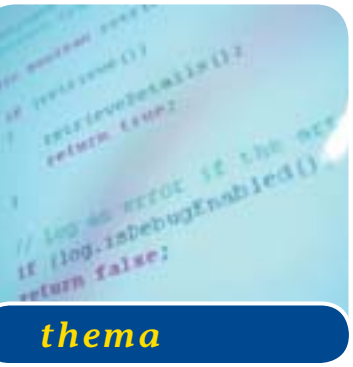


thema

Bij testgestuurde ontwikkeling (test first design) worden tijdens de software-ontwikkeling tests gemaakt nog voordat de daadwerkelijke applicatie wordt geïmplementeerd. Deze tests worden vervolgens gebruikt om de programmatuur te testen. Maken we hierbij gebruik van een test framework als JUnit (www.JUnit.org), dan kunnen deze tests ook nog eens geautomatiseerd worden uitgevoerd. Het op deze wijze ontwikkelen van software verhoogt de kwaliteit doordat de applicatie continu getest kan worden.

jcoverage

Rapportagetool voor testmetingen

Testen geeft meer inzicht in de gewenste functionaliteit. Het verschaft een goede basis om met vertrouwen te kunnen refactoren. Test driven development, JUnit tests, continue integratie, refactoring: het zijn allemaal aspecten die onderdeel uitmaken van hedendaagse systeemontwikkelprojecten.

Probleemstelling Wie doet er eigenlijk aan testgestuurde ontwikkeling? Vaak is de eerste aanzet er wel, maar vervolgens verwatert het snel. Vooral in het begin bij wat minder ervaren ontwikkelteam wordt dit snel "vergeten". Een ander probleem is de bestaande code. Hoe passen we hier een testgestuurde ontwikkelaanpak toe. Maken we voor alle bestaande code een JUnit test? Een mogelijkheid is om alleen dan een JUnit test uit te voeren op het moment dat dit nodig is. Dat betekent dat niet voor alle bestaande code een JUnit test geschreven hoeft te worden, maar alleen voor de code die nieuw is of moet worden aangepast. Indien er een fout moet worden opgelost, voer dan eerst een JUnit test uit om de fout te reproduceren. Deze zal falen. Pas vervolgens de code aan, de JUnit test zal dan met succes worden uitgevoerd. Op deze manier gaat de hoeveelheid tests automatisch groeien. De vraag rijst echter: wordt alle code nu ook daadwerkelijk door een JUnit test uitgevoerd?

Oplossing Een manier om antwoord op deze vraag te geven is een JUnit test coverage rapportage tool. Bijvoorbeeld jcoverage (www.jcoverage.com). Met jcoverage kan gemeten worden en ook gevisualiseerd in hoeverre JUnit tests volledig zijn. Door gebruik te maken van een bestaand Ant (ant.apache.org) script waarmee we onze applicatie bouwen en testen, is de JUnit test coverage meting eenvoudig toe te voegen.

Instrumentatie jcoverage werkt op basis van instrumented class files. Dit zijn class files waaraan met behulp van bytecode engineering extra code is toegevoegd. Om een coverage rapportage te verkrijgen moeten we dus eerst instrumented classes zien te verkrijgen. Dit kunnen we realiseren via een Ant taak.

```
<instrument todir="/tmp/instrumented">
    <fileset dir="classes">
        <include name="**/*.class"/>
    </fileset>
</instrument>
```

Deze taak zal alle class files die gevonden worden in de classes directory voorzien van instrumented code. Deze classes worden vervolgens in een aparte directory weggezet. De originele class files die we uiteindelijk in productie brengen blijven op deze wijze onaangepast.

Runnen van JUnit Voor het maken van een uiteindelijke rapportage moeten we nu nog al onze JUnit test runnen. Deze moeten dan wel gebruik maken van de instrumented classes. Door de instrumented classes vooraan in het classpath te plaatsen zullen deze indien aanwezig gebruikt worden.

```
<junit fork="yes" haltonfailure="no"
failureproperty="error.in.tests">
        <classpath>
            <pathelement location="/tmp/
instrumented"/>
            <pathelement
location="classes"/>
```

```
                <pathelement
location="JUnit/classes"/>
                <pathelement path="${build.
classpath}"/>
            </classpath>

            <formatter type="plain"
usefile="false"/>
            <formatter type="xml"/>
            <test todir="${junit.result.
dir}" name="AllTests"/>
</junit>
```

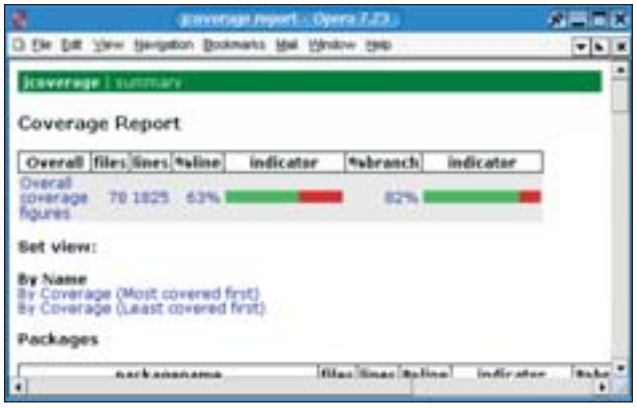
Deze JUnit test taak kan dan ook gebruikt worden voor het uitvoeren van JUnit tests zonder instrumented classes door de directory /tmp/instrumented leeg te maken.

RAPPORTAGE Tijdens het uitvoeren van de instrumented classes zal er een bestand worden aangemaakt, waarin staat welke regels code zijn uitgevoerd. Met behulp van een andere Ant taak kan hier nu een rapport uit gegenereerd worden.

```
<report format="html" srcdir="src"
destdir="jcoverage-report"/>
```

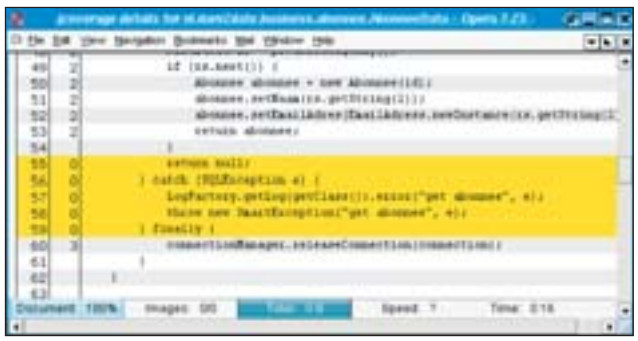
Het resultaat is een aantal html pagina's waar de JUnit test coverage van onze code getoond wordt. Op de indexpagina is in drie stappen de JUnit coverage van onze applicatie te zien. Als eerste zijn er de Overall coverage indicatoren. Deze geven een overall indicatie. Coverage wordt gemeten naar regels code, maar ook of alle mogelijke branches in de code worden getest. Hierna wordt op package niveau een indicatie gegeven van de coverage.

Als laatste wordt op deze pagina voor iedere Java-class aangegeven wat de coverage is. Door te klikken



FIGUUR 1. Coverage wordt onder meer bepaald aan de hand van het aantal regels code

FIGUUR 2. Op package niveau wordt een indicatie gegeven van de coverage



FIGUUR 3. De Java class toont de Java-code in een html pagina

vanuit aan package geeft een overzicht van het geselecteerde package, door klikken op een Java-class toont de Java-code in een html-pagina met in geel de regels code die niet zijn getest.

CONCLUSIE Met behulp van een JUnit coverage tool als jcoverage is het mogelijk om inzichtelijk te krijgen of een applicatie volledig getest wordt. Ook voor leken is het duidelijk wat wel en wat niet getest wordt. Indicatieve cijfers als 10% tegen 99% moeten ook voor hen duidelijk zijn. Daarmee is tegelijk een kwaliteitscijfer te vergeven.

Philippe Tjon-A-Hen is ICT consultant op het terrein van Java en architectuur. Hij is werkzaam bij Ordina (e-mail: philippe.a.hen.tjon@ordina.nl)