Effectief automatisch testen met Cucumber

Als ontwikkelaar wil en moet je je eigen software in redelijke mate testen voordat je die aan (acceptatie)testers overdraagt. Immers, doe je dat niet, dan is de kans groot dat de issues je om de oren gaan vliegen. Wanneer je iteratief ontwikkelt en vaak oplevert, moet je ook vaak testen. Dat lukt redelijkerwijs niet altijd meer handmatig. Automatische tests bieden dan hulp. Dit artikel kijkt kritisch naar de standaard Java-aanpak hiervoor en bespreekt een alternatief.

De standaard voor Java projecten is min of meer JUnit, en wordt ondersteund door buildtools als Maven en Ant, en IDE's als Eclipse en NetBeans. Tot zover niets nieuws onder de zon. Unittests zijn echter niet geheel waterdicht. De auteurs zijn vooral actief met het bouwen van Java-gebaseerde webapplicaties. Onze ervaring is dat zelfs projecten met een redelijke goede JUnit test-coverage (meer dan 80%), waarbij veel project-tijd wordt besteed aan schrijven en onderhoud van tests, toch geplaagd kunnen worden door flinke aantallen bugs. Het vreemde is dat dit best normaal lijkt te worden gevonden in de Java-gemeenschap. Dat is het niet. Het doel van testen is om een redelijke mate van zekerheid te krijgen over het technisch én functioneel correct functioneren van een product. Als een testmethodiek dat niet kan leveren, dan voldoet die niet. Het testen is dan niet effectief. In dit artikel bespreken we een alternatief voor de gebruikelijke JUnit-tests: Cucumber.

De unit in unittesten

Voordat we inzoomen op Cucumber, is het goed eens wat dieper te kijken waarom de effectiviteit van JUnit-tests vaak teleurstellend is. De kern van het probleem is de focus op 'unit-tests', gecombineerd met de overtuiging dat de geëigende 'unit' om te testen, een class of method is. Het unittest paradigma schrijft voor dat je units zoveel mogelijk in isolatie test. De testcode zelf verzorgt het aanroepen van de class of method. Externe afhankelijkheden van de class of method worden vaak vervangen door 'mocks', in de vorm van mock-classes, in-memory databases, webservice stubs, etc. Het idee is dat de tests daardoor onafhankelijk worden en minder breekbaar bij wijzigingen elders in de code. In de praktijk staat bij het bouwen van een Java (EE) webapplicatie het correct functioneren van een class nogal ver af van de concrete gebruikersfunctionaliteit. De echte Java-programmacode is maar een klein deel van een complex, 'polyglot' geheel. Er is ook nog front-end/view code (bijvoorbeeld JSF views of JSP's), de container doet van alles op basis van configuratie/annotaties bij de code (denk aan object-relational mapping, webservices, transactionaliteit), en soms bevat een achterliggende database ook nog een deel van de logica. De front-end code bevat steeds meer en steeds complexere client side-logica (denk aan JavaScript en AJAX), die je redelijkerwijs alleen in een browser kunt testen. Dan zijn er nog omgeving-specifieke zaken: de ontwikkelomgeving kan goed geconfigureerd zijn, terwijl de acceptatie-omgeving een fout bevat. Daardoor testen Java-unittests maar een klein deel van het hele systeem. En dus is er een grote kans dat een webapplicatie wel goed door de unittests heen komt, maar veel fouten bevat. Een ander probleem met het unittesten van classes of methods, is dat het refactoring in de weg kan staan. Bij een iteratieve werkwijze is frequent refactoren belangrijk om de kwaliteit en architectuur van de code op peil te houden en zo de productiviteit op langere termijn op peil te houden. Geautomatiseerde tests zouden daarbij moeten helpen: na een refactoring-slag moet je de tests opnieuw kunnen draaien om te controleren dat er niets stuk gegaan is. Bij class of method-gebaseerde tests lukt dat maar in beperkte mate: je kunt weliswaar de implementatie van een class of method wijzigen en dan opnieuw testen, maar als je serieus refactort en hele classes verwijdert of toevoegt, moet je de tests herschrijven. Dat is precies wat je níet wilt bij refactoring. Tests moeten onderhoudbaarheid ondersteunen, terwijl unittests de kosten van onderhoud juist kunnen verhogen. Betekent dit dat je integratietests moet gaan in doen in plaats van (of in aanvulling op) unittests? Ja en nee. Het hele onderscheid tussen unit- en integratietests is nogal contraproductief. De kern van de zaak is dat je als ontwikkelaar verantwoordelijk bent voor de realisatie van een product. Is dat product een class library? Test de classes. Is dat product een webservice? Test de webservice. Is dat product een webapplicatie? Test de webapplicatie. Beschouw je product als de unit, en test die op de meest effectieve manier voorhanden!

Frans van Buul is een hands-on software architect met een specialisatie in Java EE en SOA, werkzaam bij Inter Access. Hij is erg geïnteresseerd in innovatieve technologieën die de kwaliteit van software en het ontwikkelproces kunnen verbeteren.

Bert Jan Schrijver is software architect bij Inter Access met een voorliefde voor Java en OpenSource.

Cucumber: een overzicht

Cucumber is een testtool uit de Ruby-wereld. In de kern biedt Cucumber de mogelijkheid om testscripts in natuurlijke taal op te schrijven, maar wel op zo'n manier dat ze automatisch uitvoerbaar zijn. Idealiter vallen je specificaties en je testscripts dan samen. Cucumber wordt sinds 2009 ontwikkeld door Aslak Hellesøy. Hij heeft op vele conferenties gesproken over Cucumber. In 2011 was de 1.0-versie gereed. Interessant voor ons is dat in mei 2012 ook de port naar Java beschikbaar kwam onder de naam Cucumber JVM (niet te verwarren met het draaien van Ruby-Cucumber op de JVM via JRuby; dat kan ook maar is nogal complex). Een Cucumber-testscript ziet er bijvoorbeeld als volgt uit: Het mooie is dat dit er heel natuurlijk uitziet en aansluit bij de Scrum manier van werken. Je kunt zo'n testscript daarom ook makkelijk bespreken met een projectmanager of functioneel ontwerper en daarmee je specificaties valideren. Maar hoe kan dit uitvoerbaar zijn? De magie zit in de door Cucumber gebruikte uitbreidbare 'domain specific language' (DSL): Gherkin. De vocabulaire van deze taal is eigenlijk heel beperkt. Er zijn een stuk of 10 woorden, die je in een taal naar keuze kunt opgeven. Hieronder zien we opnieuw ons testscript, maar dan met de keywords gemarkeerd:

```
Functionaliteit: Inloggen
Als een geregistreerde gebruiker wil ik met mijn username en
password kunnen inloggen op de applicatie.

Achtergrond:
Gegeven een geregistreerde gebruiker 'frans' met password 'geheim'

Scenario: Succesvol inloggen
Als ik naar de startpagina ga
Dan zie ik het inlogscherm
Als ik username 'frans' invul
En ik password 'geheim' invul
En op 'inloggen' klik
Dan ben ik ingelogd als gebruiker 'frans'
```

Het woord 'scenario' betekent voor Cucumber dat er een test gaat worden beschreven van één of meer stappen. Het scenario slaagt als alle stappen slagen. Als er een stap faalt in een scenario, stopt het scenario en gaat Cucumber door met een eventueel volgend scenario. Een 'functionaliteit' is een te realiseren deel of aspect van de applicatie dat wordt getest; vanuit Cucumber-perspectief is het een combinatie van een aantal scenario's. De 'achtergrond' zijn stappen die worden uitgevoerd voorafgaand aan ieder scenario. Zo wordt duplicatie voorkomen. De andere woorden 'gegeven', 'als', 'dan', 'en' worden door Cucumber allemaal op dezelfde manier geïnterpreteerd. Ze geven een test-stap aan. Alhoewel het voor de tool niet

```
Functionaliteit: Inloggen
Als een geregistreerde gebruiker wil ik met mijn username en password kunnen
inloggen op de applicatie.

Achtergrond:
Gegeven een geregistreerde gebruiker 'frans' met password 'geheim'

Scenario: Succesvol inloggen
Als ik naar de startpagina ga
Dan zie ik het inlogscherm
Als ik username 'frans' invul
En ik password 'geheim' invul
En op 'inloggen' klik
Dan ben ik ingelogd als gebruiker 'frans'
```

uitmaakt, is het wel verstandig ze op een consistente manier te gebruiken: 'als' voor het aangeven van acties en 'dan' voor het aangeven van resultaatcontroles. Maar hoe kan Cucumber dan een zinnetje als 'Als ik naar de startpagina ga' uitvoeren? Hier komt het concept 'glue' om de hoek kijken: de spreekwoordelijke lijm waarmee je tekst aan daadwerkelijke test-stappen plakt. De glue bestaat uit Java methods (die eventueel ook in een andere JVM-taal kunnen zijn geschreven) met annotaties. In de implementatie moet je zelf iets schrijven wat de teststap implementeert. Voorbeeld:

```
@Als("ik naar de startpagina ga")
void ik_naar_de_startpagina_ga() {
/* We gebruiken hier Selenium
WebDriver om de stap te implementeren. */
WebDriver driver = new
FirefoxDriver();
driver.get("http://localhost:8080");
}
```

Runtime zal Cucumber de value van de @Als annotatie gebruiken om dit stukje code te koppelen aan het testscript. Een belangrijk deel van de kracht van Cucumber schuilt erin dat dit ook werkt met regular expressions, waarbij de capturing groups (de concrete strings die matchen met variabele gedeelten van de regular expression) als argument aan de method worden meegegeven. Voorbeeld:

```
@Als("ik username '(.*)' invul")
void ik_username_invul(String
username) {
WebDriver driver = new
FirefoxDriver();
WebElement element = driver.
findElement(By.name("username"));
element.sendKeys(username);
}
```

Met een vergelijkbare syntax is het ook mogelijk om hele lijsten en tabellen aan glue-methods mee te geven. Het uitvoeren van testcontroles kan op dezelfde manier als met JUnit, via assertions. Dit op zichzelf eenvoudige concept is in de praktijk enorm krachtig. Na de investering in het bouwen van een aantal voldoende generieke teststappen, kun je eindeloos variëren bij het maken van testscripts. Het maken van nieuwe testscripts kan dan zelfs door niet-programmeurs worden uitgevoerd! Het eindresultaat van een Cucumber-run is een testrapportage (zie hieronder). Het standaard testrapport lijkt sterk op de testscripts.

De stappen krijgen een achtergrondkleur: groen als het goed is, rood als het fout gegaan is, geel voor niet-geïmplementeerd en blauw voor overgeslagen. We zien natuurlijk het liefst een testrapport dat helemaal groen is, net als een komkommer. Daarnaast kun je ook screenshots of andere gegenereerde informatie laten opnemen in het testrapport. Wellicht is je opgevallen dat een deel van de informatie in het testscript niet werd gebruikt bij testuitvoering, zoals de beschrijving van de functionaliteit. Deze tekst wordt wel overgenomen in het testrapport en zorgt voor leesbaarheid van testscripts en –rapport. Je kunt op deze manier zelfs het testrapport als combinatie van specificaties, testscripts en testrapportage gebruiken.

```
Functionaliteit: Inloggen
Als een geregistreerde gebruiker wil ik met mijn username
en password kunnen inloggen op de applicatie.

Achtergrond:
Gegeven een geregistreerde gebruiker 'frans' met password
'geheim'

Scenario: Succesvol inloggen
Als ik naar de startpagina ga
Dan zie ik het inlogscherm
Als ik username 'frans' invul
En ik password 'geheim' invul
Dan ben ik ingelogd als gebruiker 'frans'
```

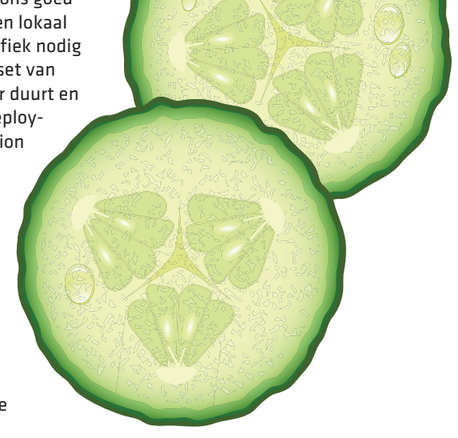
Relatie met andere tools

Cucumber is een tool die één ding goed doet, namelijk een structuur bieden voor het maken van uitvoerbare specificaties. Daardoor gebruik je het eigenlijk nooit op zichzelf. Aan de ene kant is er een manier nodig om de Cucumber-tests als onderdeel van een (automatische) build te draaien. Het gemakkelijkst is om de JUnit-runner voor Cucumber te gebruiken, dan kan vervolgens iedere omgeving die met JUnit overweg kan ook met Cucumber overweg. Bijvoorbeeld een Maven-build met de Surefire-plugin, die je dan weer kunt aansturen vanuit een continuous-integration omgeving als Jenkins. Aan de andere kant moeten de teststappen geïmplementeerd worden. Hier kun je gebruiken wat je wilt. Wij gebruiken veel Selenium WebDriver voor browser-tests, Apache HttpComponents Client voor tests van webservices, en gewone JDBC voor het implementeren van stappen die tot doel hebben het systeem in een gedefinieerde toestand te brengen voor het uitvoeren van een test, bijvoorbeeld 'Gegeven een geregistreerde gebruiker 'frans' met password 'geheim''. Dit wordt ook wel een 'fixture' genoemd. Maar het zou bijvoorbeeld ook prima mogelijk zijn om de Java-API van Sikuli (Java Magazine 2013-1) te gebruiken om browser-tests te implementeren. Een tool die een beetje in dezelfde hoek zit als Cucumber, is FitNesse. Deze tool is ook geschikt voor het maken van leesbare, executeerbare specificaties, waarbij programmeurs stukjes code maken om teststappen uit te voeren. Het grote verschil is dat FitNesse hiervoor een centrale webserver gebruikt met een Wiki-achtige structuur. De Cucumber testscripts zijn daarentegen gewoon plain-text files die je typisch via een versiebeheertool als Git of Subversion zult beheren. Cucumber is een wat eenvoudiger concept.

HET MAKEN VAN NIEUWE TESTSCRIPTS KAN DAN ZELFS DOOR NIET-PROGRAMMEURS WORDEN UITGEVOERD!

Praktijkervaringen en conclusies

Dankzij het geautomatiseerd doortesten van functionaliteit met Cucumber, kan het aantal bevindingen dat menselijke testers nog vinden drastisch worden teruggebracht. Het maken van Cucumber testscripts gaat snel, alhoewel er in het begin wel even een drempeltje moet worden genomen als je nog geen bestaande glue-methods hebt voor je project. Zo af en toe glipt er toch een bug tussendoor. De procedure om daarmee om te gaan is vrij eenvoudig. We maken een nieuw Cucumber-script dat het issue reproduceert. Dan lossen we het probleem op zodat de test slaagt. En dankzij de bestaande tests weten we dat er niets anders is omgevallen. We maken in de praktijk niet meer mee dat een hertest door een menselijke tester faalt. Hiermee is een belangrijke bron van project-frustratie geëlimineerd. Wij gebruiken Cucumber vooral voor integratie-achtige tests met een browser tegen een draaiende omgeving. Dat doen we met een apart Maven test-project dat los staat van het webapplicatie-project. De test wordt dus ook niet automatisch gedraaid als onderdeel van de Maven-build. Dat werkt in de praktijk heel natuurlijk, omdat het ook niet zozeer de build is die je test, maar het hele systeem. Zo'n test draaien we ook niet alleen bij het maken van een nieuwe build, maar ook na het deployen van een build op een bepaalde omgeving, tegen die specifieke omgeving. Zo weten we zeker dat een omgeving die vrijgegeven is voor test, het ook echt doet. Dit geeft enorm veel vertrouwen. Doordat de tests zo effectief zijn, is de drempel om zaken te refactoren veel lager geworden. Dat doen we dan ook regelmatig. Er worden ook nauwelijks meer specifieke projectafspraken voor gemaakt. Refactoren is gewoon onderdeel van het dagelijks werk, zoals het hoort. De leesbaarheid van de Cucumber-testscripts en –rapportages maakt dat ze ook relevant worden voor niet-programmeurs. Dat is een enorm verschil met standaard JUnit-tests. Onze ervaring is dat menselijke testers kunnen bekijken wat al automatisch getest is, en dan vervolgens risico-gericht kunnen bekijken wat zij nog aanvullend willen doen. Dit komt de efficiëntie van het handmatig testen dus ten goede. Maar het is niet zo dat handmatig testen overbodig wordt. Overigens zouden de Cucumber-rapportages nog wel een stukje fraaier kunnen. Standaard is het recht-toe-recht-aan op basis van de testscripts, maar je krijgt geen mooie inhoudsopgave, overzichtstabellen of dat soort dingen. Cucumber biedt echter voldoende toegang tot de ruwe gegevens (als JSON) om zoiets zelf te scripten. Een 'probleem' bij het gebruiken van Cucumber is dat het maken van nieuwe testscripts zo gemakkelijk gaat, dat het uitvoeren van alle tests zo meerdere uren kan duren. Nu biedt Cucumber wel mogelijkheden om verschillende subsets van tests uit te voeren. De volgende werkwijze blijkt voor ons goed te werken: ontwikkelaars voeren lokaal alleen die tests uit die ze specifiek nodig achten. We gebruiken een subset van tests die ongeveer een kwartier duurt en die we kunnen gebruiken bij deployments. De continuous integration server voert de hele dag door een volledige set tests uit die ongeveer 1,5 uur duurt. Zodra die tijd te lang wordt, gaan we kritisch door de tests heen en archiveren we een deel van de tests. Alles bij elkaar zijn onze ervaringen met Cucumber dus zeer positief. Geautomatiseerd testen levert nu eindelijk de voordelen op die altijd zijn beloofd. ■



REFERENTIES

Cucumber website:
cukes.info
Aslak Hellesøy's blog (de maker van Cucumber)**:**
aslakhellesoy.com