TOOLS

**Code reviews worden toegepast om de kwaliteit van software te verbeteren. Het is tevens een middel om de kundigheid van de programmeurs te ontwikkelen door elkaar te wijzen op 'best practices'. Gezien de inspanning die dit kost vinden code reviews meestal alleen plaats op de meest kritieke delen van de programmatuur. In dit artikel bekijken we de mogelijkheid om dergelijke reviews automatisch plaats te laten vinden door middel van statische analyse tools.**

# Automatische code reviews

## Softwarekwaliteit verbeteren tegen minimale inspanning

**Drs. ing. Dick Eimers**
is werkzaam bij Atos Origin BAS Emerging Technologies, waar hij zich bezighoudt met Java en open-source-technologie.

Bij een code review wordt broncode door anderen bekeken als een soort van 'second opinion'. De problemen die hierbij zijn te ontdekken zijn onder te verdelen in twee categorieën: stijl overtredingen die betrekking hebben op bepaalde code conventies en bugs of code die kan leiden tot bugs. Er zijn zogenaamde 'style checkers' beschikbaar voor de eerste categorie, zoals PMD en CheckStyle. Voor het herkennen van mogelijke bugs, ook wel 'bug checkers' genoemd, zijn er FindBugs en wederom PMD. Deze tools maken gebruik van statische analyse in dit geval het beschouwen van software zonder deze daadwerkelijk uit te voeren betekent.
Bij wijze van experiment zijn FindBugs en PMD losgelaten op een applicatie waaraan we enige tijd geleden hebben ontwikkeld. Tijdens het ontwikkelproces van deze applicatie, die bestaat uit ongeveer 8000 regels Java in 35 klassen en ontwikkeld is in een klein team, is CheckStyle (d.m.v. de Eclipse plugin die mooi met de editor integreert) naar volle tevredenheid gebruikt om Sun's code conventies af te dwingen. Hier zijn verder geen opmerkelijke bevindingen te vermelden, behalve dan dat de maximale regellengte van 80 karakters die Sun nog steeds hanteert toch echt niet meer van deze tijd is.

### PMD

Waar de afkorting 'PMD' exact voor staat lijkt niemand meer precies te weten, maar de meest logische zijn 'Programming Mistake Detector' of 'Project Meets Deadline'. Hoe dan ook, de tool is ontwikkeld met geld van DARPA, het instituut van het Amerikaanse ministerie van defensie dat verantwoordelijk is voor de ontwikkeling van technologie voor militaire doeleinden. DARPA heeft PMD beschikbaar gemaakt als open-source onder een BSD licentie. PMD was in staat een aantal verbeterpunten te identificeren:

### Non-threadsafe singleton

In de code bevond zich een singleton die lijkt op het volgende voorbeeld:

```
public class Foo {

    private static Foo singleton = null;

    private Foo() {}

    public static Foo getInstance() {
        if (singleton == null) {
            singleton = new Foo();
        }
        return singleton;
    }
}
```

PMD herkent deze constructie als het singleton design pattern en wel een singleton die onverwacht gedrag kan gaan vertonen wanneer gebruikt in een applicatie met meerdere threads. Het probleem met deze code zit in het gegeven dat meerdere threads zich tegelijkertijd kunnen bevinden in de getInstance() methode waardoor er onverwacht meerdere Foo objecten gecreëerd zouden kunnen worden.
PMD geeft ook meteen de mogelijke oplossing in het initialiseren van het veld genaamd 'singleton', dus:

```
public class Foo {

    private static Foo singleton = new Foo();

    private Foo() {}

    public static Foo getInstance() {
        return singleton;
    }
}
```

Maar ook het toevoegen van het 'synchronized' keyword aan de lijst van modifiers van getInstance() zou een oplossing bieden.

## Non-final methode aanroep in constructor

Een constructor wordt gebruikt om objecten te initialiseren. Een constructor zal altijd eerst een andere constructor in de betreffende klasse of de constructor van de directe superklasse aanroepen. Deze zal op zijn beurt hetzelfde doen, totdat de constructor van java.lang.Object is bereikt. Zo wordt altijd eerst de constructor van java.lang.Object volledig uitgevoerd en vervolgens steeds verder de hiërarchie afgedaald. Een constructor mag andere methoden aanroepen voor de initialisatie van een object. Echter, er ontstaat een waarschijnlijk niet wenselijke situatie wanneer zo'n methode niet 'final' is. Dit kan namelijk tot gevolg hebben dat de implementatie van deze methode zich bevindt op een lager niveau in de hiërarchie dan de constructor die de aanroep heeft gedaan waardoor de velden op dit niveau nog niet door een contructor geinitialiseerd zijn.
In ons geval kon PMD gerust gesteld worden door de methoden die aangeroepen worden door de constructor 'final' te maken, maar mogelijk is dit niet in lijn met het ontwerp.

## Non-optimale Collection.toArray() aanroep

Eerdere verbeterpunten waren nog specifiek voor de programmeertaal Java. Dit is een mogelijke performance verbetering waar PDM blijk geeft van goede kennis van de Java SE API. Collection.toArray() retourneert een array met alle elementen die aanwezig zijn in de Collection implementatie waarop deze wordt aangeroepen. Het element type van deze array is afhankelijk van de klasse van het object dat als argument aan deze methode wordt meegegeven. In de code bevond zich code die lijkt op het volgende voorbeeld:

```
class Foo {
    void bar(Collection x) {
        x.toArray(new Foo[0]);
    }
}
```

Hierin herkent PMD dat er een optimalisatie mogelijk is. De API documentatie vermeldt dat wanneer het aantal elementen in de collectie niet groter is dan de grootte van de array die doorgegeven is bij de aanroep deze zal worden hergebruikt om de elementen in te retourneren.

De volgende code-aanpassing zorgt voor een array van voldoende omvang en kan PMD tevreden stellen:

```
class Foo {
    void bar(Collection x) {
        x.toArray(new Foo[x.size()]);
    }
}
```

PMD herkende nog veel meer verbeterpunten, zoals een waslijst aan tips die te maken hebben met internationalisatie (i18n) en velden en variabelen die 'final' gemaakt zouden moeten worden. De laatstgenoemde is echter nogal omstreden.

## FindBugs

FindBugs is ontwikkeld door de Universiteit van Maryland en beschikbaar als open-source onder de LGPL. Een verschil met PMD is dat deze tool de analyse uitvoert op Java bytecode en niet op de Java broncode (zie Figuur 1). FindBugs gebruikt een concept dat ze 'bug patterns' hebben gedoopt. Bug patterns zijn door FindBugs te herkennen structuren in Java bytecode die mogelijk een bug zijn of hier toe kunnen leiden. FindBugs was in staat onder meer de volgende verbeterpunten te identificeren:

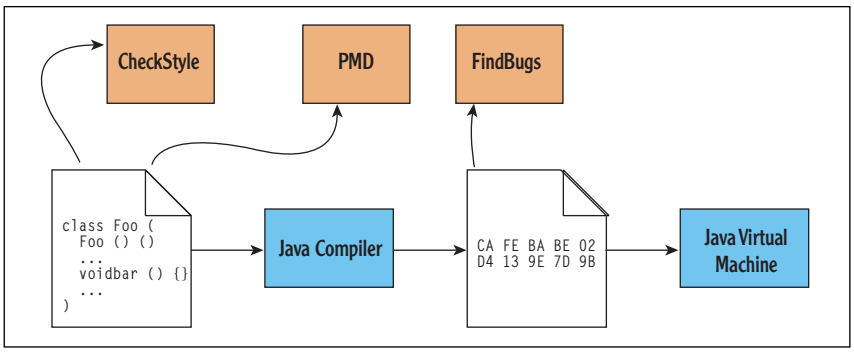
## Equals contractbreuk

Het Object.equals() contract, zoals beschreven in de Java SE documentatie en de Java Language Specification, beschrijft de gelijkheidsrelatie van objecten die worden vergeleken met deze methode. Een van de regels uit dit contract stelt, "For any non-null reference value x, x.equals(null) should return false." FindBugs heeft in de applicatie die als experiment voor dit artikel is gebruikt een aantal equals methoden herkent die zich niet aan deze regel houden.

## SQL-injectie kwetsbaarheid

Bij deze bevinding weet FindBugs ons toch enigs-

**Bij een code review wordt broncode door anderen bekeken als een soort van 'second opinion'**




Figuur 1. Automatische code reviewers en hun invoer.

**Automatische code reviews bieden extra houvast bij offshoring van systeemontwikkeling**

zins te verbazen door blijk te geven van kennis van de gevaren rond SQL-injectie. FindBugs is namelijk van mening dat de create van PreparedStatement objecten uit strings die niet-constant zijn een mogelijk lek vormen en uitnodigen tot ongewenste SQL-injectie met alle gevolgen van dien. De oplossing is even simpel als doeltreffend; De String objecten zijn immutable 'by specification' en de String variabelen kunnen we als 'final' aanmerken om FindBugs gerust te stellen. Baat het niet dan schaadt het in dit geval zeker niet. Iedere ontwikkelaar weet immers dat erg opgepast moet worden met het opbouwen van SQL query's uit invoer van gebruikers.
Verder was FindBugs in staat om een aantal mogelijke problemen te herkennen die voortkomen uit het aanroepen van een methode op een variabele die mogelijk een 'null'-waarde bevat. Daarnaast vond hij nog een 'inner class' die 'static' gemaakt kon worden, omdat deze geen gebruik maakte van de referentie naar de 'outer class' die hierdoor onnodig doorgegeven werd. Erg behulpzaam.

## Configuratie en integratie

Een automatische code review tool is vooral nuttig als deze niet te veel meldingen geeft die bij nadere inspectie niet zinvol zijn. PMD kwam met een kleine honderd meldingen waarvan ongeveer de helft van dit soort 'false positives'. FindBugs herkende beduidend minder resultaten, maar was daarentegen niet op een enkele 'false positive' te betrappen. De striktheid van de automatische reviewers is te beïnvloeden en ook is het mogelijk de verschillende 'rules', 'rule sets' (PMD) en 'bug patterns' (FindBugs) desgewenst in en uit te schakelen.

Het is duidelijk gebleken dat PMD en FindBugs allebei andere problemen herkennen en daarom is het aan te bevelen om beide in te zetten. Ook bij automatische code reviewers geldt dus dat meerdere reviewers meer zien dan één. Maar hoe kunnen we dit het beste inrichten?
Wanneer gebruik gemaakt wordt van CheckStyle's Eclipse plugin dan controleert deze de Java code tijdens het programmeren. FindBugs en PMD kennen niet zo'n interactieve integratie met hun Eclipse plugins. De analyse van deze tools is te initiëren door rechts te klikken op een package in de 'Package Explorer' en naar het menu-item van de betreffende tool te navigeren.
Er zijn 'Ant tasks' en 'Maven plugins' beschikbaar voor ontwikkelaars die liever zonder IDE werken of niet afhankelijk willen zijn van de IDE om het bouwproces te managen. In dit geval zijn ze ook erg handig om automatische code reviewers te integreren met een continuous build oplossingen om rapportages te genereren in HTML, XML en platte tekst. Het is aan te bevelen deze rapportages vervolgens te plaatsen op een lokatie waar projectleden ze kunnen bekijken.
In onze huidige configuratie is CheckStyle in gebruik om Sun's code conventies af te dwingen tijdens het programmeren. Vervolgens worden zowel PMD als FindBugs vanuit Eclipse uitgevoerd alvorens code in het versiebeheersysteem wordt ingecheckt. Bij PMD zijn een aantal 'rules' uitgeschakeld omdat ze 'practices' afdwingen waarmee we het niet helemaal eens zijn.

## Conclusies

FindBugs en PMD zijn in staat om ontwikkelaars te wijzen op 'best practices' die bugs kunnen voorkomen of de performance kunnen verbeteren. Niet alle problemen die deze tools vinden zijn zo lastig als beschreven in dit artikel. Ze herkennen ook problemen die eerder voorkomen bij minder ervaren ontwikkelaars, zoals twijfelachtige 'casts' en obscure 'if statements'. De geautomatiseerde reviewers zijn daarbij goedkoop en de kwaliteit van de reviews is constant. De geautomatiseerde reviewer heeft nooit een zwak moment, maar als hij een probleem nu niet kan vinden dan zal hij het nooit vinden.
Bovendien kunnen automatische code reviews een extra houvast bieden bij de verdere globalisering en offshoring van systeemontwikkeling. Het vastleggen van kwaliteitseisen, bijvoorbeeld ingegeven door de ISO 9126 standaard voor softwarekwaliteit, moet hiervoor onderdeel zijn van het inrichten van een ontwikkelstraat. Tools voor automatische code reviews kunnen vervolgens ingezet worden om deze te borgen.

## Toekomstmuziek

Er is in dit artikel gesproken over automatische reviews van Java software op een laag abstractieniveau, met name binnen een klasse. Een software architectuur beschrijft de structuur van de oplossing op een hoger abstractieniveau. De concrete implementatie van de oplossing door de software ontwikkelaars is binnen het kader van de architectuur specificatie in de meeste gevallen op meerdere manieren mogelijk. In de praktijk wordt echter nogal eens afgeweken van dit kader. Om te controleren of de implementatie zich conformeert aan de architectuur zou een tool die deze 'architectural constraints' afdwingt een krachtig instrument zijn. Helaas zijn er nog geen tools beschikbaar voor Java die hier met succes invulling aan geven. «

**Referenties**

FindBugs - http://findbugs.sourceforge.net
PMD - http://pmd.sourceforge.net
CheckStyle - http://checkstyle.sourceforge.net
Sun code conventies - http://java.sun.com/docs/codeconv/