Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

3

Testen geïntegreerd in het curriculum Software Engineering.

Pieter van den Homberh - Fontys Hogeschool Techniek en Logistiek

Fred van Odenhoven - Fontys Hogeschool Techniek en Logistiek

SAMENVATTING

Testen van software is een speerpunt in onze opleiding Software Engineering. In de propedeusefase wordt de testgedreven softwareontwikkeling geoefend.
De student wordt aangeleerd software met testen af te leveren. Als onderdeel van de toetsing werd een performance-assessment ontwikkeld, dat de mogelijkheid biedt modelleren, programmeren en testen integraal te toetsen. Studenten blijken deze nieuwe toetsvorm positief te waarderen. In het kader van competentiegericht onderwijs is dit performance-assessment een waardevolle toevoeging.

INLEIDING

Studenten die leren programmeren, leren een nieuwe taal, maar ook een nieuwe manier om problemen op te lossen met de computer. Als de programmeeropgaven wat ingewikkelder worden, neemt de programmeervreugde af als fouten niet snel gevonden worden.
De volgende aanpak is illustratief: de code wordt net zolang opnieuw gecompileerd totdat de compiler tevreden is. Vervolgens wordt het programma net zo vaak uitgevoerd dat er uiteindelijk een uitvoerbaar programma ontstaat, dat doet wat de student op dat moment van het programma eist. Daarbij worden na elke run de fouten, die volgens de uitvoer nog aanwezig zijn, gecorrigeerd. Het programma bepaalt in feite welke fouten de student moet corrigeren. Dit lijkt op testen maar is het niet!
Deze inefficiënte en onvolledige werkwijze kan op professionele wijze verbeterd worden door het inzetten van unittesten. Naast unittesten is het leren omgaan met een versiebeheersysteem van grote waarde voor de praktijk van de software engineer. Dat blijkt ook wel uit het feit dat de meeste IDE's zoals Eclipse en Netbeans diverse varianten van versiebeheer standaard ondersteunen.

CURRICULUM

De vakken in het eerste jaar, die relevant zijn voor dit verhaal, zijn in het eerste semester:

- PRO1 programmeren in Java deel 1.

En in het tweede semester:

- PRO2 programmeren in Java deel 2,
- SEN1 software engineering deel 1,
- MOD1 objectgeoriënteerd modelleren met UML deel 1,
- PRJ2 software project: ontwerpen en realiseren van een webapplicatie met Java en een database.

De module SEN1 behoeft nadere toelichting: het hoofdthema is objectgeoriënteerd testen en testbaarheid, evenals het beheren van een softwareproject met een versiebeheersysteem. Verder worden algemene principes van de praktijk van softwareontwikkeling bijgebracht. Als leerboek werd een Duitstalig boek door ons in het Nederlands vertaald, Vigenschow [1].

Een hoofddoel van de module is studenten de gewoonte aan te leren software altijd met de bijbehorende (unit)testen af te leveren. Dit is een van de praktijkgerichte doelstellingen van het vak.
In het project PRJ2 kunnen studenten in een projectgroep de diverse vakkennis in de praktijk toepassen. Zoals het werken met een versiebeheersysteem (momenteel Subversion) en het systematisch testen van de software.

PROGRAMMEREN

In de module PRO1 wordt Java onderwezen en maken de studenten voor het praktisch werk gebruik van BlueJ. In het eerste semester maken de studenten al kennis met unittesten, omdat BlueJ het werken met JUnit ondersteunt. Systematisch testen schrijven bij het ontwikkelen van een Javaprogramma wordt echter niet gedaan. De studenten zijn vooral bezig met het leren van de taal en maken nog geen ingewikkelde programma's.
In de module PRO2 wordt de Javakennis verder uitgebreid en wordt aandacht besteed aan webprogrammering, in dit geval werken met JSP en JDBC. Deze onderdelen zijn noodzakelijk voor het project PRJ2. Hierin wordt een webapplicatie gerealiseerd met Java.

TESTEN

Aangezien BlueJ het werken met JUnit ondersteunt, maken studenten in het eerste semester al kennis met unittesten. In het tweede semester wordt Netbeans gebruikt voor het programmeerwerk. De aard van het project PRJ2 heeft tot deze keuze geleid.

In SEN1 wordt met kleinere programmeeropdrachten het testgedreven ontwikkelen en het gebruik van Subversion geoefend. Ook hiervoor is Netbeans een goede keuze.

Voor het vak SEN1 werd een performance-assessment ontwikkeld, dat het mogelijk maakt de kennis en kunde van de student in een zo goed als mogelijke praktijkgetrouwe omgeving te toetsen. Daarnaast wordt met een schriftelijke toets de theoretische kennis getoetst. De resultaten van het begeleidend practicum worden ook meegenomen in de eindbeoordeling van de student. In het eindcijfer weegt het performance-assessment het zwaarst mee. Dit assessment toetst niet alleen het programmeren van de testen, maar ook de teststrategie en de kwaliteit van de testen, zoals onder andere de test-coverage en niet in de laatste plaats de keuze van de testdata op basis van equivalentieklassen.

VERSIEBEHEER

Het beheren van broncode wordt door professionele programmeurs gedaan met een versiebeheersysteem. De open source varianten Cvs, Subversion (svn) en het recentere Mercury of Git worden veel toegepast. Als wij professionele software-engineers willen opleiden, dan dienen onze studenten ten minste een van deze tools te leren gebruiken.

PERFORMANCE-ASSESSMENT

Ten behoeve van het performance-assessment m.b.t. testgedreven programmeren is een speciale op Linux gebaseerde toetsomgeving opgezet. Deze keuze geeft ons volledige controle over wat er wel en niet gedaan kan worden met de machines zonder afhankelijk te zijn van derde partijen. In feite implementeren we ons eigen policysysteem. De omgeving bestaat uit een speciaal daartoe ingerichte 'examserver' met clients, die een ingeperkte omgeving (rbash) aan de studenten aanbieden.

We hebben de volgende eisen aan de testomgeving gesteld:

- Het assessment moet een praktijktest zijn; de student moet getoetst worden op zijn/haar vaardigheden met betrekking tot ontwerpen en implementeren van de testen met gebruikmaking van de tools.

Twee van de ruim 300 bezoekers van het NIOC 2009

- Elke student moet de toets individueel ondergaan.
- De omgeving moet door ons gecontroleerd worden en voor elke deelnemer gelijk zijn.
- De omgeving moet beschikbaar zijn voor alle studenten voordat de toets wordt gehouden. Dus: de studenten moeten ervaring kunnen opdoen met betrekking tot de toetsomgeving.
- De omgeving moet zo realistisch mogelijk zijn en dus vergelijkbaar met de normale werkomgeving gedurende projectwerk en programmeerwerk.
- De studenten hebben de beschikking over een volledig functionele IDE (Netbeans of Eclipse).
- De studenten moeten een begrensde toegang hebben tot elektronische documentatie, zoals API-documentaties: van de applicatie, van het Junit-framework en van Java zelf.
- De studenten moeten niet hun werk kunnen delen tijdens de test: alle netwerktoegang behalve naar de examenserver moet afgesloten zijn.
- Er vindt geen disksharing plaats.
- De omgeving moet zoveel als mogelijk fraude voorkomen.
- De studenten leveren hun werk in via een versiebeheersysteem. Subversion is op dit moment de keus.

Inrichting of implementatie van de omgeving:

- De student krijgt een speciaal account voor de duur van de toets met verse wachtwoorden. De homedirectory van deze gebruiker is op de lokale harde schijf en wordt tijdens inloggen aangemaakt. Deze directories worden kort na de toets weer verwijderd. Deze home-dirs bevatten een setup voor de IDE en een browser (Firefox) met bookmarks naar de documentatie.
- Toegestane internetverkeer instelbaar tot op hostniveau en/of poortniveau.
- Zoals bekend kunnen Unixachtige machines alleen commando's uitvoeren die te vinden zijn op het pad, (dat wij bepalen) of die gespecificeerd zijn met
  behulp van een relatief of absoluut pad (bijvoorbeeld met / in het pad).
- We gebruiken rbash als shell, die het veranderen van directory en pad verhindert en commando's met een / er in onmogelijk maakt. Dit geeft ons volledige controle over wat de student wel en niet kan doen.

We hebben de volgende tools ontwikkeld ten behoeve van de omgeving:

- Kleine aanpassingen aan de start-scripts van de display en windowmanager van de Linux X11 omgeving.
- De Windowmanager is IceWM, die is licht en gemakkelijk aan te passen.
- Kleine scripts om de SSH-verbinding op te zetten en de verbinding te testen.
- Voor de documentatielatex: een paar macro's helpen bij het maken van meertalige opgavenbeschrijvingen, zie Codesnippet 1.
- Antbuildscripts genereren de Javadoc, zipbestanden voor de studenten en een testomgeving om onze eigen oplossing te testen.
- Scripts om de accounts en de repositories 'on the fly' te creëren.
- Om de studenten te voorzien van code die up-to-date is, waaruit onze oplossing verwijderd is hebben we een codestripperanttask ontwikkeld. Zie Codesnippets 2, 3 en 4.

```
\begin{itemize}
 \item \large
  \EN{A few \LaTeX\ macros to help in making multilingual task descriptions.}
  \NL{Een paar \LaTeX\ macros helpen bij het maken van meertalige
            opgavenbeschrijvingen.}
  \DE{Einige \LaTeX\ Macros zum erstellen von mehrsprachigen
            Aufgaben-Beschreibungen.}
\end{itemize}
```

Snippet 1: Een paar Latex macro's helpen bij het maken van meertalige opgavenbeschrijvingen

```
<taskdef name = 'codestripper'
        classname = 'org.fontysvenlo.codestripper.CodeStripper' / >
<target name = 'strip'>
      <codestripper
        todir = 'out'
        dir = 'src'
        deletelines = 'true'
        includes = '**/*.java'
        dryRun = 'false'
        verbose = 'true'
        starttag = '^\s*/[*]{2}.*'
        endtag = '.*[*]/.*'
      />
</target>
```

Snippet 2: Deze taak stript C-stijlcommentaar uit alle Java-bronbestanden, maar kan van alles strippen uit op tekst gebaseerde bestanden. (Zie: http://javabits.fontysvenlo.org)

```
int getCredits(Score score, PlayLine line) {
  //Start Solution::replacewith:://TODO
  if (line == PlayLine.D1 || line == PlayLine.D2) {
    return 2*this.getCredits(score);
  }
  return getCredits(score);
  //End Solution::replacewith::return 0;
}
```

Snippet 3: Code in de repository

```
int getCredits(Score score){
                    //TODO
                    return 0;
}
```

Snippet 4: Code na stripping

DE APPLICATIE

We wilden een applicatie, die niet goed testbaar is door hem gewoon te gebruiken. Daarmee verhinderen we dat code via de applicatie wordt geschreven in plaats van via testen. Wij denken dat je dat kunt bereiken door iets in te brengen, dat niet door de student te controleren is: randomness. Van daaruit is het een kleine stap naar een spel. De slotmachine was geboren. Ons initiële ontwerp was functioneel maar saai (voor de test zelf was het achteraf gezien prima). Nu hebben we een swingapplicatie die ook geluiden produceert afhankelijk van het spelverloop.

Het is een goedogende applicatie, die een model bevat met goede testmogelijkheden. Een voorbeeld is de methode met taak: 'bepaal de score op een bepaalde speellijn'.

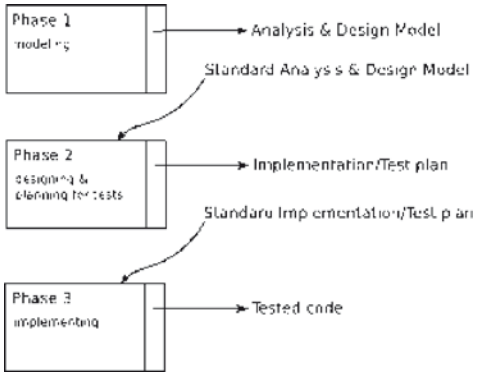
Van de slotmachine hebben we meerdere varianten op voorraad:

- iconslotmachine
- numberslotmachine
- yahtzeeslotmachine
- pokerslotmachine

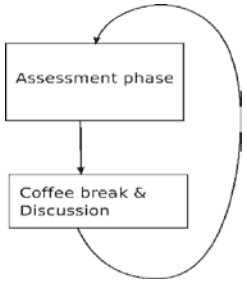
Zie voorbeelden in figuren 1 en 2.
(DEMO http://javabits.fontysvenlo.org/iconslotmachine/dist/launch.jnlp)

Figuur 1:
Icon slotmachine

Figuur 2:
Poker slotmachine



Figuur 3:
Assessment-fasen



Figuur 4:
Assessment-cyclus

DE OPDRACHT

Projectcode is compleet geïmplementeerd inclusief testsoftware (EYODF[1]). De code is voorzien van Javadoc-commentaar. Onze zorg is de foutkans daarin zo klein mogelijk te maken. De codestripperanttask in de antbuildfile voor het codestrippen en aldus maken van het examenproject voor de studenten. De opgavencode stemt zodoende altijd overeen met onze code, die getest is met onze code. Dit is ook het geval met de code in de overige documentatie die als PDF wordt aangeboden. De opgavenbeschrijving is een aparte PDF en de //TODO's in de code zijn natuurlijk ook een goed hulpmiddel. Alle code, configuratie en latex-bestanden bevinden zich in een docent-repository.

UITVOERING EN VERWERKING

Als we onze examenomgeving zelf getest hebben, kunnen we de omgeving openstellen voor de studenten. Het is mogelijk tijdens het assessment de voortgang te volgen door in de repositories van de studenten de commitopdrachten te monitoren. Weinig commits van een student zeggen niet altijd dat hij of zij achterloopt, het kan ook zijn dat die student te weinig commits uitvoert, wat tegen de aanbevelingen is. Er moet namelijk regelmatig gecommit worden.

Onze eigen tests kunnen daarna ook gebruikt worden om de code van de studenten te testen. Het is niet verstandig om alleen op basis van deze testen een cijfer vast te stellen. Een tweede testmogelijkheid is om de test van de student te laten uitvoeren op onze eigen docentcode. Verder kunnen ook coveragetesten worden toegepast, maar het is raadzaam om de code van de studenten ook met het oog te bekijken.

Uitbreidingen

Het performance-assessment werd tot nu toe gebruikt voor de module SEN1. Het is natuurlijk ook een assessment voor de programmeervaardigheid zelf en zou daarom ook mogelijk ook een deel van de toets van de module PRO2 kunnen vormen.

1 Eat Your Own Dog Food

Een wezenlijke uitbreiding is de combinatie van meerdere performance-assessments, zoals achtereenvolgens voor: modelleren, testontwerp en testuitvoering.

Na elke fase is een pauze wenselijk. Daarin kunnen studenten ook onderling over de afgelopen fase discussiëren. De volgende fase borduurt dan voort op hetzelfde thema, maar begint met eenzelfde beginsituatie voor alle deelnemers. Bijvoorbeeld: een modelleringopgave kan voorafgaan aan een implementatiefase. In de implementatiefase krijgt elke student een standaarduitwerking van de modelleringfase. Zie ook de figuren 3 en 4. Drie fasen zijn waarschijnlijk teveel voor een assessmentdag.

CONCLUSIES

Onze ervaring beperkt zich tot nu toe tot het toetsen van het programmeren en het testen, en die ervaring is ronduit positief te noemen. Het opzetten van de omgeving om het performance-assessment uit te kunnen voeren is niet bijzonder complex, maar kost wel de nodige inspanning en controles. De studenten ervaren het als een eerlijke en nuttige aanvulling op de schriftelijke toetsen en het groepswerk.

↑

De centrale hal van de Faculteit Natuur en Techniek van de HU

[1]
Uwe Vigenschow Objectgeorienteerd Testen, vertaald uit het Duits door Joop Wekking, Fontys Hogeschool voor Techniek en Bedrijfsmanagement 2007, ISBN 978-90-811523-1-02

Over de auteurs
Ir. Pieter van den Homberh en Dr. Ir. Ferd van Odenhoven zijn verbonden aan Fontys Hogeschool Techniek en Logistiek te Venlo, opleiding Software Engineering.