EXAMENSARBETE VID CSC, KTH

Svensk titel

Engelsk titel

Jonas Frogvall
frjo02@kth.se

Exjobb i: exjobbsämne

Handledare: Stefan Arnborg

Examinator: Stefan Arnborg

Uppdragsgivare: Sigma

Svensk titel

Sammanfattning

Svensk sammanfattning.

Sammanfattningen på det språk rapporten är skriven på placeras först.

Engelsk titel

Abstract

Engelsk sammanfattning.

Om båda sammanfattningarna får plats på en sida, placerar du dem på samma sida. Sätt annars den andra på en ny sida.

Förord

Förordet sätter du på ny sida. Du behöver inte ha förord.

Innehållsförteckning

Problembeskrivning

Historia

Mutation testing

MuJava

Lösningsförslag 2, etc

Dynamisk klassladdning och Java Webstart

Dynamisk laddning av klasser i java är inget problem. Vet man vad klassen heter och vart den ligger så är man bara en rad kod bort.

```
ClassLoader loader = new URLClassLoader(new URL[]{}, Glitch.class.getClassLoader());
glitchList.add((Glitch) (loader.loadClass("com.sigma.qsab.glitches.customglitches." +
                glitchName)).newInstance()));
```

Vill man ladda in alla klasser i ett paket, vilket vi vill om vi skall få den dynamiska inladdningen av buggar att fungera, så går det också utan större bekymmer genom att ta reda på programmets *classpath*.

```
String path = System.getProperty("java.class.path");
```

Sedan går man igenom mappen där klasserna ligger och laddar in dem en efter en.

```
File[] files = (new File(path + "/com/sigma/qsab/glitches/customglitches")).listFiles();
for (File f : files) {
    if (f.getName().endsWith(".class")) {
        glitchList.add((Glitch) (loader.loadClass("com.sigma.qsab.glitches.customglitches." +
                        f.getName().substring(0, f.getName().length() - 6)).newInstance()));
    }
}
```

Detta funkar alldeles utmärkt lokalt, men när man distribuerar koden via Java Webstart och kör det så upptäcker man att klasserna inte har laddats.

Det är i sig kanske inte så konstigt, då Java Webstart kräver att man har hela programmet sparat i jar-filer. Då får man istället tänka om och börja ladda klasserna ur en jar-fil.

```
JarFile jar = new JarFile(path);
Enumeration<JarEntry> entries = jar.entries();
while (entries.hasMoreElements()) {
    JarEntry entry = entries.nextElement();
    if (entry.isDirectory()) continue;
    if (!entry.getName().startsWith("com.sigma.qsab.glitches.customglitches.")) continue;
    glitchList.add((Glitch) (loader.loadClass(entry.getName()).newInstance()));
}
```

Provar man nu att köra programmet lokalt från en jar-fil, så funkar det alldeles utmärkt, men när man kör det via JWS så fungerar det fortfarande inte.

Problemet inser man om man tittar på vad JWS använder för *classpath*.

```
C:\\Program Files (x86)\\Java\\jre6\\lib\\deploy.jar
```

Classpath är inte en sökväg till programmet, utan istället Javas *deploy.jar*, där JWS finns och körs från.

Hur löser vi då problemet? Hur skall man kunna hitta den jar-fil som klasserna ligger i? Svaret är tyvärr att det gör man inte.

Hur har vi då löst problemet?

[alt 1]

Vi kom fram till att man kommer bli tvungen att kompilera om hela projektet oavsett när man skapat en ny bugg-klass, för att den skall hamna i webstart-jaren, så då skrev vi ett eget maven-plugin som sökte igenom mappen med klassfiler och gjorde i ordning en property-fil som innehöll namnet på alla bugg-klasser. När vi känner till namnet på alla klasserna vid kompilering, så är det sedan en enkel sak att ladda in dem i runtime.

```
Properties glitchMap = new Properties();
glitchMap.load(GlitchLoader.class.getResourceAsStream("/glitches.properties"));
ArrayList<Glitch> glitchList = new ArrayList<Glitch>();
ClassLoader loader = new URLClassLoader(new URL[]{}, Glitch.class.getClassLoader());
for (int i = 0; i < Integer.valueOf(glitchMap.getProperty("nrofglitches", "0")).intValue();
     i++) {
    glitchList.add((Glitch) (loader.loadClass("com.sigma.qsab.glitches.customglitches." +
                   glitchMap.getProperty("" + i)).newInstance()));
}
```

[alt 2]

Skriv något om att ha klasserna i en separat jar-fil som man lägger en länk till i jnlp-filen. Ej testat ännu.

Testmoduler och cirkulära beroenden

Tanken var att skriva en separat modul med ett testbibliotek för att hantera GUI-testning. Denna modul var menad att vara den enda modul som har kontakt med FEST-biblioteket och abstrahera bort kännedom om FEST från den som skriver testkod. En annan fördel detta skulle medföra är att man kan byta testmiljö utan att testen behöver skrivas om, eller ens känna till förändringen.

Det uppstod dock ett problem med denna abstrahering. FEST kräver att den känner till den Frame den skall jobba mot; och är således beroende av implementationsmodulen. Eftersom testen i sig är beroende av testbiblioteksmodulen och är en del av implementationsmodulen, så uppstår ett cirkulärt beroende och koden går inte att kompilera.

Ett antal lösningsförslag diskuterades.

Man kan ta bort kravet om abstraktion och låta testen känna till och sköta initiering av FEST-biblioteket och därmed ta bort testbibliotekets beroende till implementationsmodulen.

En annan, bättre, lösning är att lägga till ytterligare ett abstraktionslager och låta integrationstesten ligga i en egen modul, vilket resulterar i att implementationen inte längre är beroende av testbiblioteket.

Oavsett vilken väg man väljer att gå så har man löst problemet med cirkulära beroenden.

TDD

Att skriva kod testdrivet är att skriva testen först och sedan med minsta möjliga möda anpassa koden så att testen går igenom. Har man till exempel, som utgångspunkt, ett test som skickar in en sträng i en funktion och förväntar sig att strängen antingen godkänns (funktionen returnerar sant) eller inte godkänns (funktionen returnerar falskt), så är det enklaste sättet att skriva en funktion som skall klara testet genom att bara returnera det förväntade värdet. Lägger man sedan till ett till test som förväntar sig ett annat värde så får man skriva om funktionen på ett sådant sätt att den på enklaste sätt klarar av det både det gamla och det nya testet. Sedan skriver man ett nytt test och modifierar koden så att testet går igenom, o.s.v.

Fördelar

Det blir rätt från början. Blir det inte rätt så går inte testet igenom och man upptäcker genast att något är galet.

Ändrar man eller optimerar i koden vid något tillfälle så upptäcker man direkt om det helt plötsligt finns ett test som inte går igenom. Hade man inte haft några test så hade man kanske inte hittat buggen alls och den hade nått release.

Nackdelar

Det tar längre tid att skriva funktioner som är väldigt simpla.

Det kan hända att man missar att skriva ett test för något som borde ha testats och i och med att man implementerar funktionerna allt eftersom testen inte går igenom, så kanske man missar ett fall man hade täckt om man hade skrivit funktionen rätt från början.

Jag har nu skrivit funktioner utifrån test, men jag har även provat motsatsen. Jag har skrivit funktioner som jag sedan skrivit test för (och modifierat kod när den inte gått igenom).

Fördelen med att skriva koden innan testet märker man väl framförallt på väldigt enkla funktioner, där det är lätt att göra fel från början och inte allt för mycket meck att rätta till om man upptäcker ett fel. En funktion som jämför två strängar kanske man inte behöver lägga ned en massa tid och energi på att skriva ”fel” från början för att sedan rätta till.

Samtidigt gäller motsatsen också. Om en funktion är, eller riskerar bli, mer avancerad, så kan det vara bra att ta små steg i rätt riktning hela tiden, istället för att skriva en avancerad metod där det sedan visat sig att man inte tänkt på allt och man måste skriva om stora partier och kanske till och med behöva skriva om hela funktionen helt och hållet (och då kunde man ju gjort så redan från början).

Någon slags diskussion

Implementation

Slutsats

Litteraturlista

ANDERSSON, P. 1999. Testing algorithms. *Journal of algorithms*, Vol. 4, 1999, s. 217–223.

ANGEL E. 2011. *Computer graphics, a top down approach.* Prentice Hall. ISBN 123456789.

Modern och välkänd lärobok i datorgrafik som används vid KTHs kurser.

MORKES J. OCH NIELSEN J. 1997. *Consise, scannable, and objective: how to write for the web.* http://www.useit.com/papers/webwriting/writing.html

Artikel av bl.a. webbgurun Jakob Nielsen om hur webbsidor bör skrivas. Beskriver två experiment med läsning av webbsidor.

PETTERSSON, A. 2000. *Att skriva vetenskapliga rapporter.* Studentlitteratur. ISBN 123456789

Detta är en fejkad bok som inte finns.