

Agile Testing

Automatiseret test
Af Poul Staal Vinje

Certified Scrum Master
Certified Scrum Practitioner

poul.vinje@gmail.com
www.agile-metoder.dk

Indhold

Indhold

Agile Testing	1
1. Indledning	2
2. Eksempel	2
3. xUnits.....	4
4. Behovet for automatiseret unittest	7
5. Prioritering af automatisering	7
6. Systemforvaltning	7
7. Andre former for automatisering	8
8. Anbefaling.....	8

1. Indledning

Dette er den tredje artikel om Agile Testing. Den fokuserer på automatisering.

Automatiseringen kan ske som unittest. Modul for modul, i takt med at de udvikles,

Alternativt kan det ske som en automatiseret test rettet mod brugergrænsefladen, og resultatet er en automatiseret system- eller accepttest.

I et objektorienteret udviklingsforløb kan man sige, at det bliver en unittest når automatiseringen bygges op om metoderne i forretningsklasserne, og en system- eller accepttest, når den rettes mod præsentationsklasserne, altså grænsefladen.

Det indledende eksempel fortsætter processen fra artikel 2. Det er en unittest, og det sker i en testdrevet udvikling hvor testcases skrives før koden.

2. Eksempel

I afsnittet 'Resultatet af forløbet' i artikel 2, vises en række testcases. I figur 1 i denne artikel er de første otte automatiseret.

```
require 'test/unit'
require 'ak1'

class Testregeltype < Test::Unit::TestCase
  def test_type
    assert_equal('1', regeltype(2,10000))
    assert_equal('2', regeltype(2,60000))
    assert_equal('3', regeltype(4,5000))
    assert_equal('4', regeltype(4,160000))
    assert_not_equal('1', regeltype(6,10000))
    assert_not_equal('2', regeltype(6,170000))
    assert_not_equal('3', regeltype(2,320000))
    assert_not_equal('4', regeltype(4,320000))
  end
end
```

Figur 1: Testcases fra artikel 2

Den første linie importerer Frameworket, Test/Unit.

Den anden linie importerer min kode.

I de resterende linier kaldes den metode i min kode, der skal testes. Testen stopper når der returneres et resultat der er forskelligt fra det jeg forventer. Eller, som det er tilfældet med de sidste fire testcases, hvis det ikke er forskelligt.

I eksemplet bruger jeg kun de første otte testcases fra artikel 2. For at holde det så simpelt som muligt. De første fire testcases adresserer de fire gyldige ækvivalensklasser. De sidste fire testcases adresserer ugyldige ækvivalensklasser.

Jeg kan nu køre dem, og se dem fejle, enkeltvis, indtil jeg får skrevet koden der sørger for at de passerer.

Figur 2 viser den metode jeg har skrevet. Det er det simpleste jeg har brug for som eksempel.

```
def regeltype (aar, kapital)
  if aar < 1 or aar > 5
    return '5'
  end

  if kapital < 1 or kapital > 300000
    return '6'
  end

  if aar >= 1 and aar <= 3 then
    if kapital <= 50000
      return '1'
    else
      return '2'
    end
  end

  if kapital <= 50000
    return '3'
  else
    return '4'
  end

end
```

Figur 2: Min Ruby kode

Jeg kører testen fra et prompt, som det ses i figur 3, men det kan gøres en automatisk gentaget regressionstest med faste mellemrum.

Unittest vil for de fleste være stedet at starte sin automatisering.

```
C:\ruby\freeride>aku
Loaded suite C:/ruby/freeride/aku
Started
.
Finished in 0.0 seconds.

1 tests, 8 assertions, 0 failures, 0 errors

C:\ruby\freeride>
```

Figur 3: Her passerer alle otte testcases.

3. xUnits

I eksemplet bruger jeg et Testing Framework der er skrevet til Ruby, fordi jeg skriver koden i Ruby. Tilsvarende vælges der andre Frameworks når der bruges et andet sprog. I Java hedder det mest anvendte JUnit. Til C# hedder det CSUnit eller NUnit. Disse Frameworks omtales generisk som xUnit. De er simple at lære. Et team er i gang med at bruge det, den samme formiddag.

Mange udviklingsmiljøer, de såkaldte IDE'er, giver mulighed for at integrere Frameworks. For eksempel kan CSUnit integreres i Microsofts. På trods af at Microsoft har undladt det i sin gratis Visual Express udgave, har jeg selv gjort det, med få minutter arbejde. JUnit kan integreres i de mest almindelige udviklingsmiljøer til Java. Oprettelse af testcases til koden er med andre ord en del af det konkrete udviklingsarbejde. Testcases kan oprettes og afvikles uden at forlade sit udviklingsmiljø.

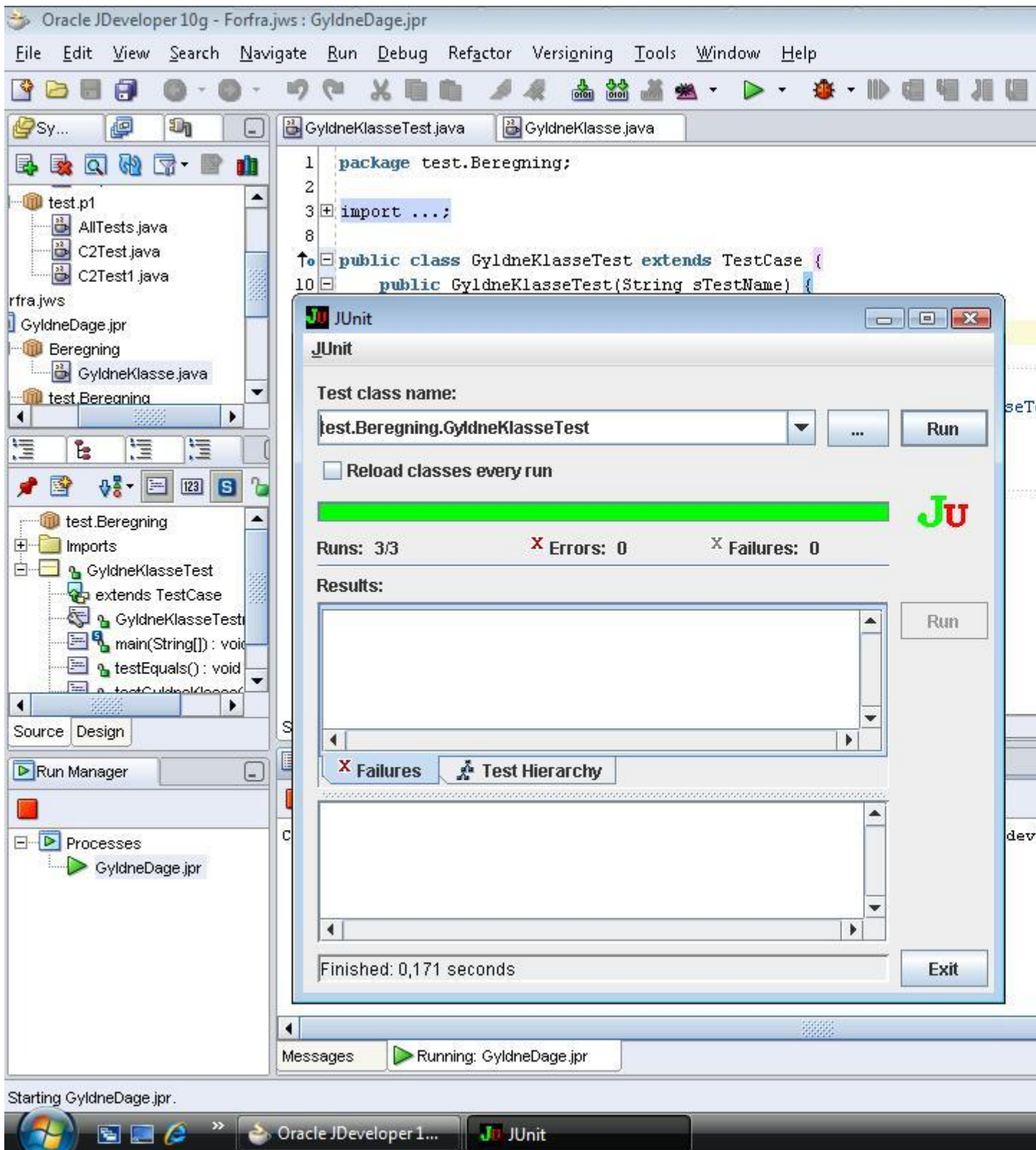
Figur 4 viser at en moderne IDE giver muligheden. Jeg har brugt JDeveloper til at give et eksempel på JUnit. Jeg har skrevet assertions og java, og kører den fra IDE'en.

xUnit har mange fordele, hvoraf jeg vil fremhæve to: De virker, og de er gratis. Jeg kunne fristes til at nævne en tredje, nemlig at supporten, ved hjælp af nettet, er omfattende og effektiv. Men der er sjældent, eller aldrig, brug for hjælp. Det duer.

Testen kræver ingen manuel verifikation. Jeg skal ikke selv kontrollere resultatet af en test.

xUnits giver ikke indgreb i koden; ingen code intrusion.

xUnits gør det muligt at udvikle pålidelig kode, i en tilfredsstillende testdrevet udviklingsproces, aldeles gratis. Og med fuld dokumentation af både kode og test, der i øvrigt bør betragtes som et hele.



Figur 4

4. Behovet for automatiseret unittest

Første gang er unittesten en del af en testdrevet udviklingsproces (TDD). Senere er det regressionstest. Med de nødvendige indslag af videreudvikling i en fornyet TDD proces.

Unittesten er uundværlig i alle Agile udviklingsforløb på grund af behovet for at gentage testforløbet når der er tilføjet ny funktionalitet. Testcases følger udviklingen i brugernes krav.

Hvis kravene ændrer sig, ændrer jeg testcases, og testen failer. Således som den skal, og som jeg ønsker at den skal.

Hvis grænsen på 50.000 i eksemplet flyttes, ændrer jeg testcases. Testen failer, således som den skal, og som jeg ønsker det.

I alle tilfælde hvor testen failer, kan jeg bringe koden i orden, gentage testen til den passer, og fortsætte udviklingen.

Automatiseret unittest er nyttig i mange forskellige sammenhænge. Her er nogle væsentlige:

- Videreudvikling i samme iteration (TestDrivenDevelopment)
- Videreudvikling på tværs af iterationer (Regressionstest)
- Refactoring af koden (Regressionstest)
- Ændringer og tilpasninger (TestDrivenDevelopment i systemforvaltning)
- Continuous Integration in the Small (Regressionstest)
- Continuous Integration in the Large (Regressionstest)
- Rettelser af fejl (TestDrivenDevelopment, Gentest og Regressionstest)

Automatiseret unittest kan også nyttiggøres i et klassisk forløb, hvor koden skrives først. Men det kræver større motivation at skrive testcases til færdig kode.

5. Prioritering af automatisering

Målet er fuld automatisering. Men mange vælger at prioritere således at det kun er kritiske og/eller komplekse metoder. Ikke simple 'get' og 'set'. Primært kode der er følsom over for tilstandsskift.

En kollega i Tyskland fortalte at de i snit automatiserede 20 % af klasserne og at det svarede til 30 % af koden. Valget var baseret på detaljerede risikoanalyser.

6. Systemforvaltning

De eksisterende systemer, der også skal eksistere i en overskuelig fremtid, er nemmest at automatisere løbende. Det kan gøres i fire situationer:

1. Automatisering i forbindelse med at rette en fejl
2. Automatisering i forbindelse med tilpasning af kode, når omverdenen ændrer sig
3. Automatisering i forbindelse med videreudvikling af en del af systemet
4. Automatisering i forbindelse med refactoring og sanering af en del af systemet

Skriv testcases og automatiser dem før indgrebet. Det er betryggende at se den automatiserede test gå glat igennem før koden ændres. Skriv derefter de nye testcases før den nye kode, altså en testdrevet proces.

Ved rettelse af en fejl er det også betryggende at starte med at skrive de testcases der genskaber fejlen. Når den er genskabt, ændres koden, og testen passerer.

I forbindelse med refactoring er det også bedst at have testen på plads, derefter forbedre koden, og se at testen stadig passerer.

7. Andre former for automatisering

Når unittest er på plads kan man gå videre med automatiseret accepttest, hvor testen af en hel applikation automatiseres. Det er mere omfattende, og kræver flere ressourcer. Gode eksempler på værktøjer er Selenium og Watir, begge til test af web applikationer.

En tredje mulighed for automatisering er systemtesten. Det er en lidt anden vej at gå, hvor man forenklet kan sige at det er unittest i stort format. Det er værktøjer som FIT/FITNesse. Igen er det en vej der kræver flere ressourcer, men vil for mange være et naturligt skridt efter at være i en god gænge med unittesten.

Accepttesten kan udvikles fra første dag. Og bruges som AcceptanceTestDrivenDevelopment (ATDD).

8. Anbefaling

Beskriv behovet for automatiseret test for hvert projekt der startes, og for hvert produkt der forvaltes.

Begge dele skal være Agile beskrivelser. Det betyder at de er kortfattede, typisk ved kun at omfatte tekst der svarer til en enkelt A4 side, og kan læses og forstås af målgruppen på 10-15 minutter.

Beskriv hvem der er involveret i automatiseringen, hvordan testdata til automatiseringen fremskaffes og værktøjerne der benyttes til det. Resultatet er en Agil og Lean proces der beskriver automatiseringen.

Lad generelt automatisering ligge i forlængelse af en TDD/ATDD proces. Både når det gælder nyudvikling og videreudvikling af produkterne i en aktiv systemforvaltningsproces.

Få automatisering af testen ind som en rutinemæssig del af arbejdet.