Testning och Debugging

Mikael Svahnberg*

2023-12-08

1 Introduktion

- Thomas & Hunt: **Kapitel 7: While you are Coding**
- Thomas & Hunt: Kapitel 3; topic 20: Debugging
- Thomas & Hunt: Kapitel 9; topic 51: Pragmatic Starter Kit

2 Städa din Arbetsplats

- I en fabrik händer ofta olyckor och man gör misstag när
  - saker står i vägen,
  - det är skräp på golvet,
  - verktyg står inte där de skall göra, osv.
- Arbete tar längre tid när man först måste
  - rensa upp en arbetsyta
  - leta efter verktygen
  - leta efter materiel som skall användas
  - osv.

3 5S

I Japan introducerades begreppet **5S**

Sort Gå igenom allt materiel och alla verktyg och plocka bort det som inte används

- Lättare att få en överblick över det som är kvar
- Logiskt ordnat
- Lättare att se vad som saknas, behöver beställas mer av, mm.
- Mer effektiv användning av arbetsyta

*Mikael.Svahnberg@bth.se

- Säkrare eftersom det finns färre hinder.

Set in Order Placera det som är kvar så att de lätt kan hittas när de behöver användas

- Lätt att hitta
- Lätt att nå
- Lätt att nå på ett säkert sätt

Shine Städa arbetsplatsen regelbundet

- Säkrare arbetsplats
- Trevligare arbetsplats
- Städa bort det som är trasigt och förbrukat
- Lätt att se när något inte är som det skall.

Standardize Etablera rutiner för när och hur man Sort, Set in Order, och Shine.

- Etablera en arbetskultur
- Alla vet vilka ansvar de har
- Även i en nödsituation

Sustain Gör allt det ovanstående till en vana.

- "Do without being told"

4 5S och Mjukvara

Sort Sortera det som används, plocka bort det som inte används

- Vilka verktyg använder du faktiskt?
- Vilka ramverk/plugins behövs faktiskt?
- Vilka `#includes` använder du egentligen?
- Vilka delar av programvaran används inte längre av kunderna?
- Vilka delar av programvaran har du levererat till vilken kund?

Set in Order Placera det som är kvar så att det är lätt att hitta

- Script för att sätta upp en ny utvecklingsmiljö
- Överlever ändringar en reboot?
- Konfigurationshantera *allt*, inklusive konfigurationen i sig själv.
- Fokus på *flow*. Ett enda kommando för en ny server eller databas, rollback av databasen eller arbetsplatsen.
- Snygga till koden så den är lätt att förstå

Shine Städa arbetsplatsen regelbundet

- Ny feature → ny branch + rollback av databas och arbetsplats.

- Snygga till koden så den är lätt att förstå

Standardize Etablera rutiner

- Processdokumentation
  - ny utvecklingsiteration
  - rollback av arbetsplatsen
  - driftsättning av applikationen
  - **Regler för vem som får göra vad**

Sustain Gör det som en vana

- Continuous Build / Integration / Deployment
  - Verktyg som testar automatiskt när du gör en commit.
- Skapar trygghet för utvecklare: Testerna fångar (förhoppningsvis) dina misstag.
- Lättläst kod == lättare att se buggarna.
- Skapa tester som en del av det normala arbetsflödet
  - Del av utveckling
  - Del av felsökning
  - *Konfigurationshantera testkoden!*
- Script och makron för att skapa nya filer (kommer rätt `#includes` med? I samma ordning?)

5 Shine: Refactoring

Refactoring handlar om att man förbättrar programkoden *utan att förändra funktionaliteten*

- Läsbarhet (t.ex. döp om variabler)
- Underhållbarhet (t.ex. strukturera om designen)

☐ You leave the code cleaner than you found it.

☐ There is no new functionality added together with a refactoring

☐ All existing tests still pass after refactoring

6 Ren kod

☐ Obvious to other programmers

- Reasonable size of each component/class/method
- Good names for classes attributes, methods, variables

☐ Does not contain duplicated code

☐ Does not contain unnecessary classes

☐ Passes all tests

→ cheaper to maintain

7 Smutsig Kod – Teknisk Skuld

(The term *Technical Debt* is coined by Ward Cunningham)

- Quick fixes that we promise to ourselves that we are going to fix later.
- Goldplating that has no use right now but which we still need to maintain.
- Business Pressure
  - Ship it! We'll fix the details later...
  - Botch it so it more or less works. Hide the parts that are "under construction"
- Lack of Understanding of Consequences of Technical Debt
  - Technical Debt cost in time and resources for *all* development.
- Lack of adherence to original design decisions
  - maybe forgotten?
  - architecture decay
  - brittle system
  - changes in one component/class affect other classes
  - → time to trace and understand effect of changes
- Lack of documentation
  - How can you adhere to orignal design decisions if they are not documented?
  - Training of new employees
- Lack of interaction between team members
  - Shared understanding of design decisions means stricter adherence and less brittle systems
- Lack of tests
  - (Automated) tests bring direct feedback on what works or not.

8 Code Smells

- Originally by Martin Fowler.
- Can be summarised as "You know all the bad things your OO design teacher told you not to do? Well, don't!"

Examples

Bloaters Code that grows too long → low cohesion

- long methods

- long parameter lists
- large classes
- using primitives instead of small objects

Object Orientation Abusers incorrect use of object oriented programming principles

- Classes that do the same things but with different method names
- Classes that only partially implement the interface from a superclass
- Long and elaborate switch statements rather than relying on polymorphism

Change Preventers Responsibilities are scattered through code so you need to change several places at once

- When many methods need to be edited for a single change (e.g. adding a new product type)
- Parallel inheritance hierarchies: Adding a class in one hierarchy means you also need to add a class in another hierarchy

Dispensables Pointless code or text that does not contribute anything

- Too many comments
- Duplicate code
- Dead Code
- Classes that no longer do anything meaningful
- Adding classes or inheritance hierarchies for future needs

Couplers Things that tie classes too closely together

- Using data in other classes (more than your own data)
- Message chains `myFancyObject->getFrobnicator()->createFluxCapacitor()->initiate()`

9 När skall man strukturera om

Tre steg:

1. Första gången, bara se till att det funkar
2. Andra gången du gör något liknande, känn igen att det är likt men gör det ändå
3. Tredje gången – Strukturera om!

När du lägger till en feature

- Refactor under tiden du läser den existerande koden som en del av att förstå den

När du fixar en bug

- Städa upp koden under tiden du letar efter buggen

Kodgranskningar

- Regelbunden aktivitet med syfte att städa upp koden

Arbeta lite med omstrukturering hela tiden

10 Refactoring Techniques OVERVIEW

(A selection that has a design impact: There are many more techniques for how to write clearer code *within* methods)

- Break out code into new methods to simplify the code
- Move methods and attributes to the class that should be responsible for them
- Remove classes that do not have any responsibilities
- Hide delegates to avoid method chains. If you are just "object hopping" to reach the right object, then you know too much about the design.
- Use wrapper classes to add functionality to libraries.
- Use getters and setters to access data
- Keep code from different layers/components separate. Duplicate data that should pass between components.
- Introduce classes to maintain collections (xxxManager, xxxContainer, ...)
- Use Design Patterns instead of if-then-else chains.
- Create methods for complex if-then-else statements: `if condition() then trueCondition() else falseCondition()`
- Always return a meaningful object (e.g. a Null Object)
- Rename classes/methods/parameters/attributes/variables to meaningful and readable names
- Separate queries from modifiers → avoid side effects in code
- Parameterise method (from `frobnicateA(), frobnicateB(), \dots` to `frobnicate(type)`
- Replace complex constructor with a factory method.
- Apply Design Patterns
- Apply Fundamental Object Oriented Design Principles

11 5S och Testning

En stor del i att få 5S att fungera är att man har bra rutiner för **testning**

- Testning måste vara en kontinuerlig vana
- Vältestad kod ger trygghet
- Trygghet ger lättrörlighet

12 Vart är vi på väg?

- Cheshire-katten i Alice i Underlandet: *Om du inte vet vart du är på väg så spelar det ingen roll vilket håll du går!*
- Testning som ett sätt att förstå kraven
- Testning som ett sätt att bestämma gränssnittet för en komponent/klass/metod

13 Manuell Testning

Görs hela tiden, delvis undermedvetet och delvis konkret:

1. Vi bestämmer oss för att skriva en metod, t.ex. `Date calculateShippingDate(Packet thePacket)`
2. Vi funderar på:
   - Vilka parametrar har metoden, vad betyder de?
   - Vad kan gå fel med parametrarna? Hur upptäcker vi det?
   - Vad skall metoden egentligen göra?
   - Vad skall metoden returnera?
   - Hur hanterar vi om det blir fel?
3. Vi skriver en första version av metoden.
4. Vi testar genom att anropa metoden med lite olika inparametrar.
   - Vi skriver ut lite resultat med `console.log()` och inspekterar manuellt att det verkar stämma.
   - Stämmer våra antaganden i steg 2? Annars gå tillbaka till steg (2) och uppdatera.
5. Är vi klara? Om inte, skriv mer av metoden och gå tillbaka till steg (4)

Utmaningar

- Vi minns inte vad vi redan testat
- Vi har ingen övergripande strategi för vilka värden på parametrar vi vill testa
- Vi har ingen övergripande strategi för vilka returvärden vi vill testa

14 Testramverk

- På den här nivån är det "enkelt" att skriva testkod. *Unit-Testing*
- **Utmaningen är att spara testerna, strukturera, och dokumentera dem.**
- Till detta finns speciella testramverk

  - Ofta integrerade i utvecklingsmiljöerna
  - Gör det lika enkelt att testa som att kompilera
  - Kan hjälpa till att generera data (property-based testing)
- Testdriven utveckling (TDD – Test-Driven Development)

```
TEST(SceneTests, listAvailableElementsRigtNumberOfElements) {
  Scene bilbliotkek;
  GameObject testObject("testObject");


  std::vector<GameObject> elements =bilbliotkek.listAvailableElements();

  ASSERT_EQ(elements.size(),1);
  ASSERT_EQ(elements[0]==testObject,1);
}
```

```
Feature: Is it Friday yet?
  Everybody wants to know when it's Friday

  Scenario: Sunday isn't Friday
    Given today is Sunday
    When I ask whether it's Friday yet
    Then I should be told "Nope"
```

15 A Software Tester walks into a bar...

A Software Tester walks into a bar and orders a beer.

- Then they orders -1 beers
- orders 999999999 beers
- orders a duck
- orders 0 beers
- orders a sdlkfjhkjsdhgfk
- orders `null`

... Warming up. Let's try some edge cases:

- Orders 3 friends to come over for some fun.
- Unhooks the tap and orders a beer.
- Breaks all the glassware and orders a beer.
- Sets the bar on fire and orders a beer.
- Orders someone else a beer.
- Has everyone order a beer.

- Orders in russian.
- Orders a beer for later.
- Orders every beer.
- Walks into the bar backwards.
- Runs into the bar.
- Sits at the bar overnight doing nothing to see what happens.
- Tries to sell a beer.
- Quickly orders a second beer before the first is served.
- Interrupts the order midway and walks out. (`^C`)
- Orders a beer in IE6.
- Orders 1 ; select * from liquors; — beers.
- Orders an apostrophe and walks out without paying the bill.
- Waits for someone else to order, stands between them and the bartender, takes the drink (Man In The Middle Attack)

. . . then sends all the beers back.

16 Lita inte på användare / Lita inte på dig själv!

- "I'm looking at a message on my screen: *No stupid idiot would ever get to this point*!"
- "Where did they find 10 stupid users?"
- `https://github.com/kuronpawel/big-list-of-naughty-strings`
- `https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/`

Räkna med:

- felinmatningar
- rätt inmatning som du missförstått
- medvetna försök att hacka ditt system
- att du behöver skydda: indata, utdata, debuginformation, konfigurationsfiler, header-data, . . .

17 Felsökning

Olika sorters fel

Kompileringsfel programmet kan inte kompileras

Körfel programmet krashar eller kastar ett felmeddelande

Designfel programmet ger fel svar

18 Strategier för felsökning

0. **Läs felmeddelandet.**
1. Återskapa felet.
2. Vad är minsta möjliga antal steg för att återskapa felet?
3. Skriv ett automatiserat test som återskapar felet.

19 Debugging

- Förstå ett körande program
- Inspektera detaljer
  - *Vad är värdet på den här variabeln?*
  - *När anropas den här metoden?*
  - *Vilka värden har parametrarna?*
  - *Vad händer?*
  - *Kommer jag ens såhär långt?*

Verktyg:

- Papper & Penna
- Blandade utskrifter: `console.log()` `printf()` `System.out.println()`
- Utskrifter via ett debug- eller log-ramverk (kanske med olika log-nivåer, kanske med mer information)
- Stegvis exekvering: *Debugger*

20 Sammanfattning

Håll koden ren och **Låt det bli en vana att städa**

- Det är lättare att se vad som är fel om ingenting är ivägen
- Refactoring: Förbättra koden utan att förändra funktionaliteten.

Använd automatiserade tester

- Analysverktyg för att förstå vad koden skall göra
- Felsökningsverktyg för att utforska buggar
- Minne för att se till att buggarna inte dyker upp igen

Bli bra på felsökning

- Läsa felmeddelanden
- Debugging

21 Nästa Föreläsning

- Dokumentera din kod
- Självdokumenterande kod
- Struktur som dokumentation
- Dokumentera din arbetsprocess
- `https://www.archbee.com/blog/how-to-write-code-documentation`

22 Övning: Test och Debug PRACTICE

23 Introduktion till SorterTool

- `https://codeberg.org/mickesv/SorterTool.git`
- *SorterTool* implementerar och testar ett par olika sorteringsalgoritmer.
- Det finns många olika sätt att sortera listor på som är olika snabba.
  - Sök på "Sorting out Sorting" för en gammal film (30 min) från 1980 som vi tvingades titta på i Datastruktur-kursen.
  - "Big-O" notation beskriver komplexiteten hos algoritmer.
- Exempel:
  - Insertion Sort: `O(n²)` Flytta element ur vägen och stoppa in elementet på rätt plats
  - Selection Sort: `O(n²)` Hitta det minsta värdet och stoppa in det först; börja om på element 2
  - Merge Sort: `O(n*log n)` Se till att varje par är ordnade, kombinera par `n` med `n+1`; upprepa.
  - QuickSort: `O(n*log n)`
    1. Välj ett element i mitten
    2. se till att alla till vänster är mindre och alla till höger större
    3. upprepa för vänster och höger.
  - Bubble Sort: `O(n²)` jämför varje element med alla andra och byt plats på alla element som inte redan är ordnade.

24 Kom igång med övningen

1. Klona SorterTool till din dator: `https://codeberg.org/mickesv/SorterTool.git`
2. Öppna projektet i din IDE. Studera följande filer:
   - `src/Main.java` för att snabbt kunna köra programmet
   - `src/Sorter.java` implementerar de olika sorteringsalgoritmerna.

- `Tests/SorterTest.java` testar `Sorter`.

3. Kör alla tester. Vad händer?
4. Kör specifikt testet för `bubbleSort()`

25 Skaffa mer information

Läs meddelandet

Nej, allvarligt! **Läs meddelandet!**

- Vad kan `AssertionFailedError` betyda?
- I vilken fil och på vilken rad händer det?
- Vad står det på den raden?
  - Räcker detta för att förstå vad som gick fel?
  - Hur kan du ta reda på mer informaiton?

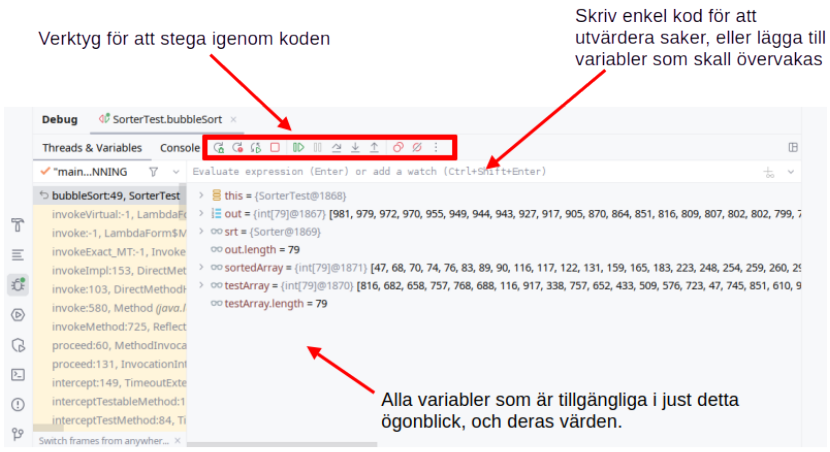
26 Använd Debuggern

1. I filen `SorterTest.java`, klicka till höger om radnummret 56 (`assertTrue(isOrdered(out))`); Det borde bytas till en liten stopp-skylt
2. Kör nu testet för `bubbleSort()` i "Debug"-läge.

Vad ser du nu i:

- kodfönstret?
- fönstret nedanför koden?

27 Debug-fönstret



- Här kan vi se varför testet misslyckas, men inte vad i koden som orsakade felet.
- *Ledtråd:* Titta på arrayen `out` och alla värden där. Kan du se något mönster?

28 Stega genom koden

1. Flytta break-punkten frånmn rad `56` till rad `54` (`int [] out = srt.bubbleSort(testArray)`).
2. Kör testet för `bubbleSort()` igen i Debug-läge (Starta om testet när du får frågan).

Viktiga verktyg för att stega genom koden:

Continue/Resume Kör på till nästa breakpoint

Step over Kör nästa instruktion, och stanna när du kommer tillbaka

Step in Följ med in i nästa instruktion (oftast in i en metod)

Step out Kör klart metoden du är i nu, och stanna när du kommer tillbaka.

Använd **Step in** nu för att stega in i anropet till `srt.bubbleSort()` .

- Notera att variabelfönstret ändras. Vad ser du där nu? Varför?

Stega ett par varv i den inre for-loopen.

- raderna med `for` och `if` körs i varje varv.
- När fortsätter koden in i if-satsen?
  - Vid vilka värden på `out[outer]` respektive `out[inner]` ?

Stämmer detta? (Det gör det inte. Vad borde hända?)
Rätta till felet och fortsätt debugga.

- Som du märker verkar inte ändringen fungera.
- Du måste *starta om* testet för att det skall ta effekt.

29 Kör om alla tester

- När du nu har fixat felet så borde alla tester bli gröna.
- Varför blir både `sort()` och `bubbleSort()` gröna? Du har ju bara fixat `bubbleSort()`...

30 Lek vidare på egen hand

- Kan du använda debuggern för att förstå hur sorteringsalgoritmerna fungerar?
- Skriv din egen sorteringsalgoritm och testa den.
- Just nu finns det ett test för varje metod. Kan du ha fler tester för varje metod?
  - Vilka fler tester kan vara relevanta?