

# 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. döpa 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 original 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 sdlkfjhksdhgfk
- 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**

- Läs 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:  $O(n^2)$  Flytta element ur vägen och stoppa in elementet på rätt plats
  - Selection Sort:  $O(n^2)$  Hitta det minsta värdet och stoppa in det först; börja om på element 2
  - Merge Sort:  $O(n \cdot \log n)$  Se till att varje par är ordnade, kombinera par  $n$  med  $n+1$ ; upprepa.
  - QuickSort:  $O(n \cdot \log n)$ 
    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:  $O(n^2)$  jämför varje element med alla andra och byt plats på alla element som inte redan är ordnade.

## 24 Kom igång med övningen

1. Klon 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

Verktøy for å stega igjennom koden

Skriv enkel kod for å utvärdera saker, eller legge til variabler som skall övervakas

The screenshot shows the Visual Studio Code interface with the 'Debug' tab selected. The 'Threads & Variables' pane on the left shows the execution stack, with 'bubbleSort:49, SorterTest' selected. The 'Console' pane on the right shows the evaluation of expressions. The 'Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)' input field is highlighted with a red box. Below the input field, the results of the evaluation are displayed, including the values of 'this', 'out', 'sortedArray', 'testArray', and 'testArray.length'. Red arrows point from the text annotations to these elements: one to the 'Step Over' button, one to the input field, and one to the variable values.

Debug SorterTest.bubbleSort

Threads & Variables Console

main...NNING Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

bubbleSort:49, SorterTest

```
invokeVirtual:1, Lambda$4  
invoke:1, Lambda$Form$5  
invokeExact_MT:1, Invoke  
invokeImpl:153, DirectMet  
invoke:103, DirectMethod  
invoke:580, Method$java.  
invokeMethod:725, Reflect  
proceed:60, MethodInvo  
proceed:131, InvocationInt  
intercept:149, TimeoutExt  
interceptTestableMethod:1  
interceptTestMethod:84, Ti
```

```
this = (SorterTest@1868)  
out = (int[79]@1867) [981, 979, 972, 970, 955, 949, 944, 943, 927, 917, 905, 870, 864, 851, 816, 809, 807, 802, 802, 799, 798]  
out.Length = 79  
sortedArray = (int[79]@1871) [47, 68, 70, 74, 76, 83, 89, 90, 116, 117, 122, 131, 159, 165, 183, 223, 248, 254, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300]  
testArray = (int[79]@1870) [816, 682, 658, 757, 768, 688, 116, 917, 338, 757, 652, 433, 509, 576, 723, 47, 745, 851, 610, 909, 809, 807, 802, 802, 799, 798, 797, 796, 795, 794, 793, 792, 791, 790, 789, 788, 787, 786, 785, 784, 783, 782, 781, 780, 779, 778, 777, 776, 775, 774, 773, 772, 771, 770, 769, 768, 767, 766, 765, 764, 763, 762, 761, 760, 759, 758, 757, 756, 755, 754, 753, 752, 751, 750, 749, 748, 747, 746, 745, 744, 743, 742, 741, 740, 739, 738, 737, 736, 735, 734, 733, 732, 731, 730, 729, 728, 727, 726, 725, 724, 723, 722, 721, 720, 719, 718, 717, 716, 715, 714, 713, 712, 711, 710, 709, 708, 707, 706, 705, 704, 703, 702, 701, 700, 699, 698, 697, 696, 695, 694, 693, 692, 691, 690, 689, 688, 687, 686, 685, 684, 683, 682, 681, 680, 679, 678, 677, 676, 675, 674, 673, 672, 671, 670, 669, 668, 667, 666, 665, 664, 663, 662, 661, 660, 659, 658, 657, 656, 655, 654, 653, 652, 651, 650, 649, 648, 647, 646, 645, 644, 643, 642, 641, 640, 639, 638, 637, 636, 635, 634, 633, 632, 631, 630, 629, 628, 627, 626, 625, 624, 623, 622, 621, 620, 619, 618, 617, 616, 615, 614, 613, 612, 611, 610, 609, 608, 607, 606, 605, 604, 603, 602, 601, 600, 599, 598, 597, 596, 595, 594, 593, 592, 591, 590, 589, 588, 587, 586, 585, 584, 583, 582, 581, 580, 579, 578, 577, 576, 575, 574, 573, 572, 571, 570, 569, 568, 567, 566, 565, 564, 563, 562, 561, 560, 559, 558, 557, 556, 555, 554, 553, 552, 551, 550, 549, 548, 547, 546, 545, 544, 543, 542, 541, 540, 539, 538, 537, 536, 535, 534, 533, 532, 531, 530, 529, 528, 527, 526, 525, 524, 523, 522, 521, 520, 519, 518, 517, 516, 515, 514, 513, 512, 511, 510, 509, 508, 507, 506, 505, 504, 503, 502, 501, 500, 499, 498, 497, 496, 495, 494, 493, 492, 491, 490, 489, 488, 487, 486, 485, 484, 483, 482, 481, 480, 479, 478, 477, 476, 475, 474, 473, 472, 471, 470, 469, 468, 467, 466, 465, 464, 463, 462, 461, 460, 459, 458, 457, 456, 455, 454, 453, 452, 451, 450, 449, 448, 447, 446, 445, 444, 443, 442, 441, 440, 439, 438, 437, 436, 435, 434, 433, 432, 431, 430, 429, 428, 427, 426, 425, 424, 423, 422, 421, 420, 419, 418, 417, 416, 415, 414, 413, 412, 411, 410, 409, 408, 407, 406, 405, 404, 403, 402, 401, 400, 399, 398, 397, 396, 395, 394, 393, 392, 391, 390, 389, 388, 387, 386, 385, 384, 383, 382, 381, 380, 379, 378, 377, 376, 375, 374, 373, 372, 371, 370, 369, 368, 367, 366, 365, 364, 363, 362, 361, 360, 359, 358, 357, 356, 355, 354, 353, 352, 351, 350, 349, 348, 347, 346, 345, 344, 343, 342, 341, 340, 339, 338, 337, 336, 335, 334, 333, 332, 331, 330, 329, 328, 327, 326, 325, 324, 323, 322, 321, 320, 319, 318, 317, 316, 315, 314, 313, 312, 311, 310, 309, 308, 307, 306, 305, 304, 303, 302, 301, 300, 299, 298, 297, 296, 295, 294, 293, 292, 291, 290, 289, 288, 287, 286, 285, 284, 283, 282, 281, 280, 279, 278, 277, 276, 275, 274, 273, 272, 271, 270, 269, 268, 267, 266, 265, 264, 263, 262, 261, 260, 259, 258, 257, 256, 255, 254, 253, 252, 251, 250, 249, 248, 247, 246, 245, 244, 243, 242, 241, 240, 239, 238, 237, 236, 235, 234, 233, 232, 231, 230, 229, 228, 227, 226, 225, 224, 223, 222, 221, 220, 219, 218, 217, 216, 215, 214, 213, 212, 211, 210, 209, 208, 207, 206, 205, 204, 203, 202, 201, 200, 199, 198, 197, 196, 195, 194, 193, 192, 191, 190, 189, 188, 187, 186, 185,
```

- 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 från 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?