Testdriven utveckling

Per Strandberg, Maj 2013

Idag kommer vi lära oss att…

- TDD är en bra utvecklingsmetod

- Grundmetoden är
  - **Red** – skriv testerna först
  - **Green** – skriv kod
  - **Refactor** – snygga till och förbättre kod

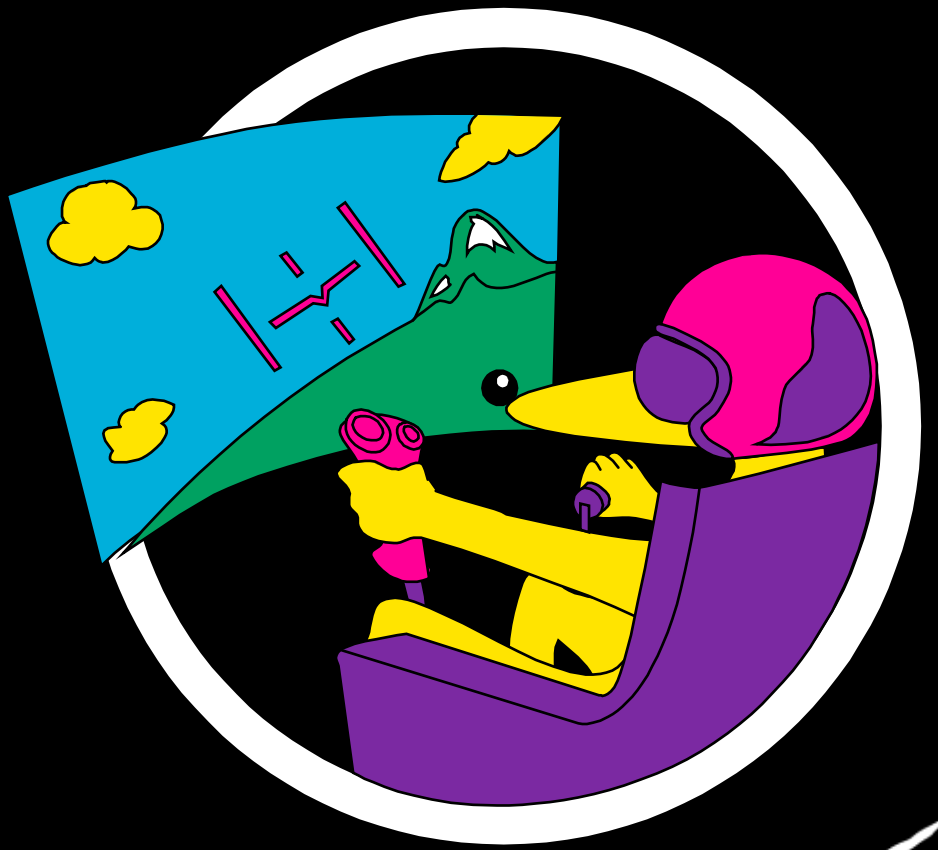
- TDD bör kompletteras med
  - Ramverk för enhetstester
  - Statisk kodanalys
  - Satstäckning

Per Strandberg

- Jobbar mer eller mindre med test sedan 2006
- Jobbat även med krav, granskning och utveckling i Python, .NET och embedded
- Bästa testminne var när jag…

Åhörarna

- Ålder/Erfarenhet
- Roller?
- (Hur) Gör ni enhetstester nu?
- Vilka andra tester gör ni?



1 – Lite om Test i Allmänhet och utvecklingsprocesser

Mål med testning?

Förebygga fel

Hitta fel eller risk

Underlätta och ge
stöd vid utveckling

Mål med testning?

Ge information om systemet

Ge förtroende för ”kvalitet”

Uppfyller vi krav?

Mäta ”kvalitet”

Hur fungerar test?

Det går inte att testa allt

Test kan hitta fel

Tidig testning lönar sig

Hur fungerar test?
Sju testprinciper

Test beror på sammanhang.

Ansamling av fel (där det finns ett fel finns det ofta fler)

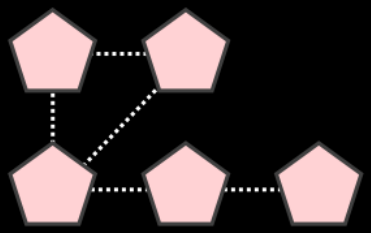
Det kan finnas fel trots att tester går igenom.

Att bara göra en typ av test kan göra systemet immunt.

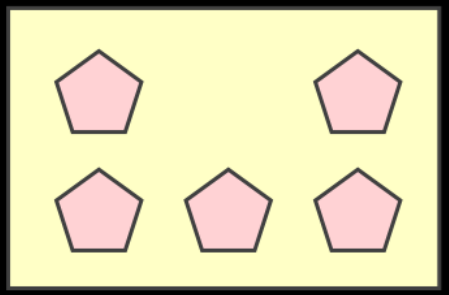
Testnivåer

Komponenttest

Integrationstest

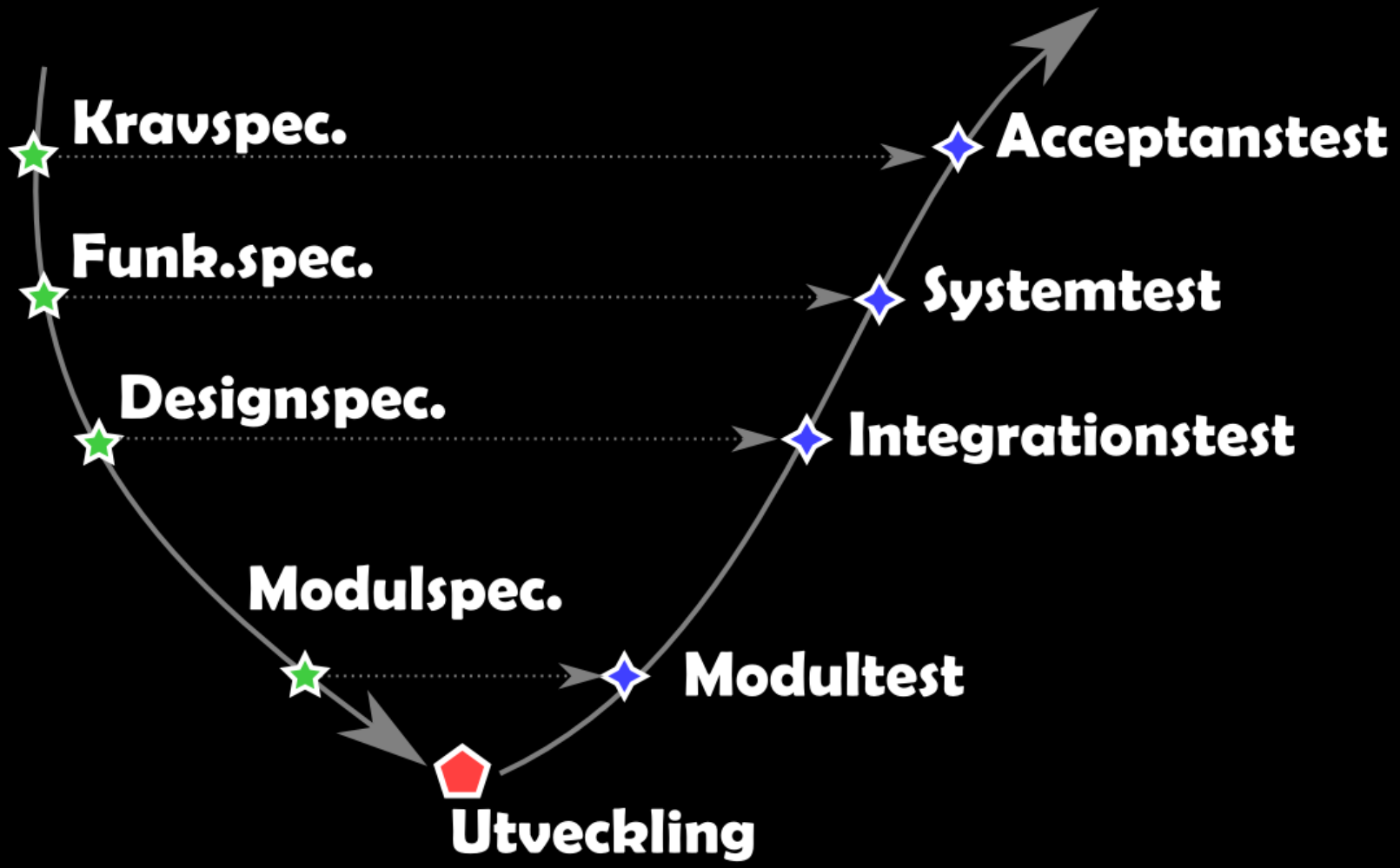


Systemtest

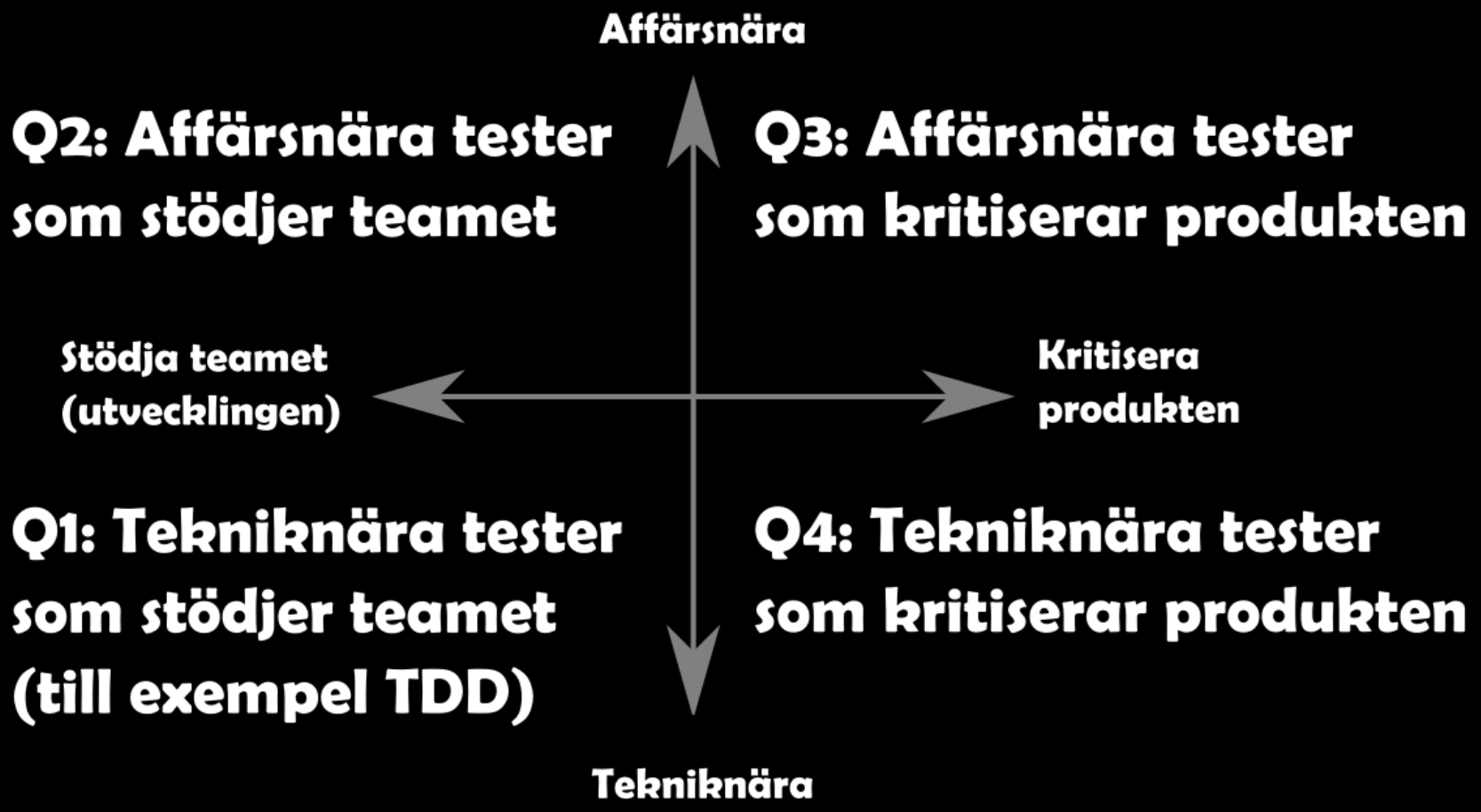


Acceptanstest

Utvecklingsmodell – V-modellen



Testprocess – Agila Testkvadranter



2 – Testdriven utveckling

Är testdriven utveckling bra?

Typiskt 2 testfall
per ”100 rader C”

Tar bort kanske
50% av felen

Bad fix injection: 3%

Är testdriven utveckling bra?

Referens: Economics of Software Quality, Addison Wesley Educational, Capers Jones & Olivier Bonsignour, ISBN: 9780132582209

Typiskt 2 testfall
per ”100 rader C”

Jämför med 3 testfall per ”100 rader C” för vanliga enhetstester

Tar bort kanske
50% av felen

Jämför med 35% för vanliga enhetstester

Bad fix injection: 3%

Jämför med 4% för vanliga enhetstester

Är testdriven utveckling bra?

Referens: Economics of Software Quality, Addison Wesley Educational, Capers Jones & Olivier Bonsignour, ISBN: 9780132582209

Typiskt 2 testfall
per ”100 rader C”

Jämför med 3 testfall per ”100 rader C” för vanliga enhetstester

Tar bort kanske
50% av felen

Jämför med 35% för vanliga enhetstester

Bad fix injection: 3%

Jämför med 4% för vanliga enhetstester

Är testdriven utveckling bra?

Bättre satstäckning
än liknande metoder

Bättre på att hitta buggar än
jämförbara metoder.

Referens: Economics of Software Quality, Addison Wesley Educational, Capers Jones & Olivier Bonsignour, ISBN: 9780132582209

Typiskt 2 testfall
per ”100 rader C”

Jämför med 3 testfall per ”100 rader C” för vanliga enhetstester

Tar bort kanske
50% av felen

Jämför med 35% för vanliga enhetstester

Bad fix injection: 3%

Jämför med 4% för vanliga enhetstester

Är testdriven utveckling bra?

Lägre komplexitet

Bättre modularitet

Bättre satstäckning
än liknande metoder

Bättre på att hitta buggar än
jämförbara metoder.

Mindre död kod

Mindre debuggning – snabbare
lokalisering av fel

Referens: Economics of Software Quality, Addison Wesley Educational, Capers Jones & Olivier Bonsignour, ISBN: 9780132582209

Är testdriven utveckling dålig?

Falsk trygghet

Övertestning

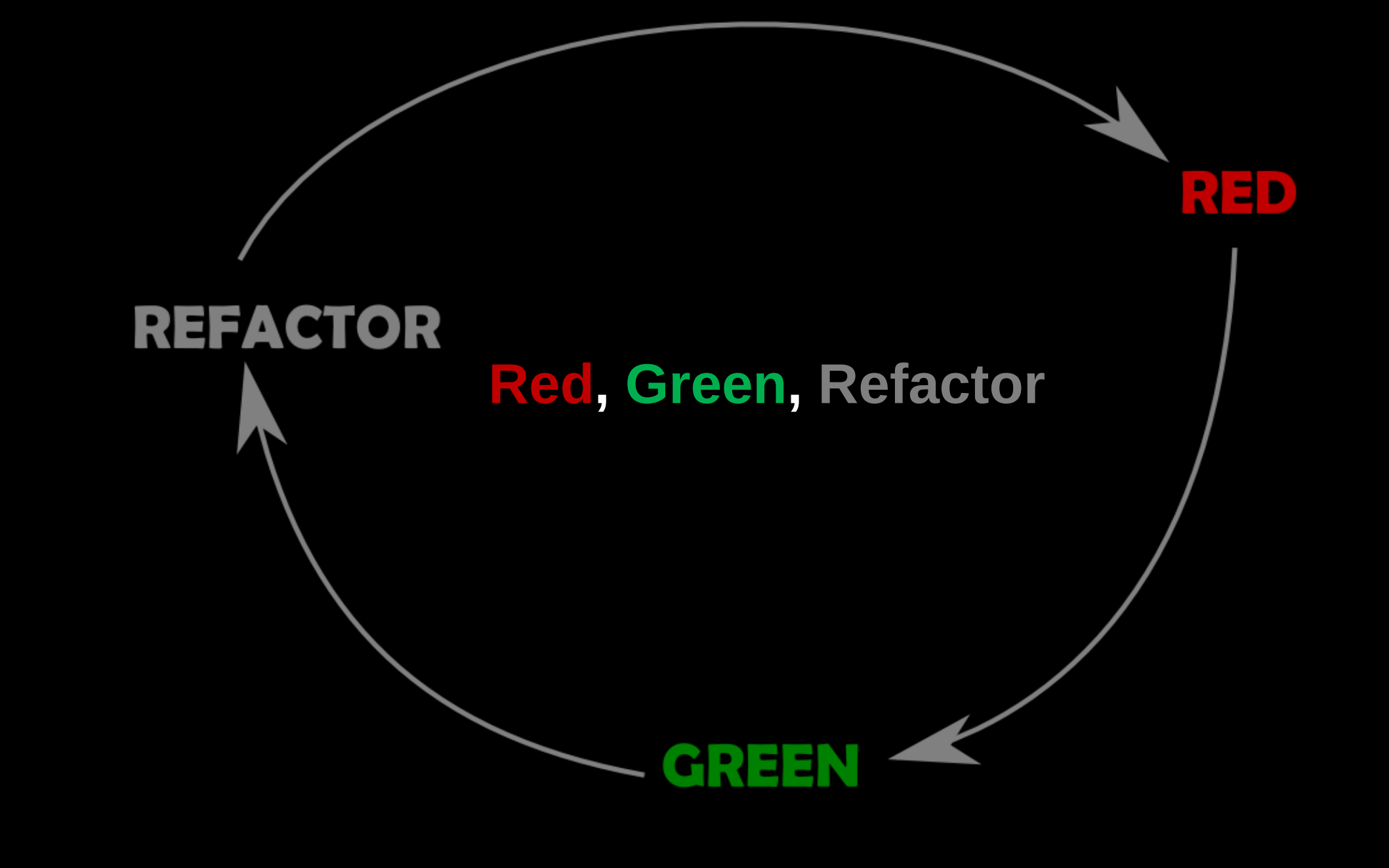
Är testdriven utveckling dålig?

Dubbelfel

Mindre fokus på annan testning

Test hamnar i produktion (#if DEBUG…)

Red, **Green**, Refactor

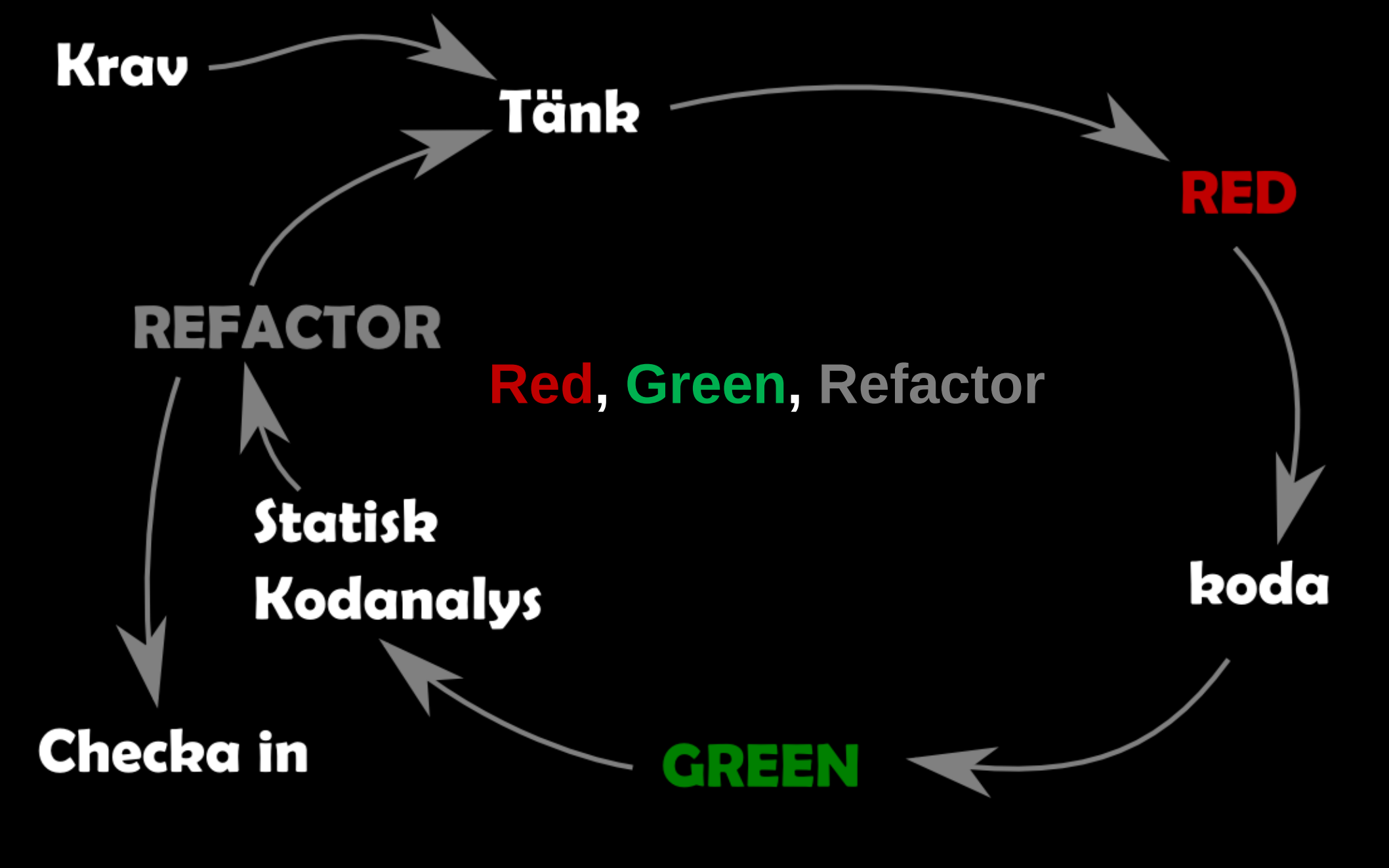

RED
GREEN
REFACTOR
Red, Green, Refactor

Red, **Green**, **Refactor**

Krav?

Kod?

Kvalitet?


Krav
Tänk
RED
koda
GREEN
Checka in
Statisk
Kodanalys
REFACTOR
Red, Green, Refactor

Kaffepaus?

3 – Exempel

(och satstäckning, statisk kodanalys och utvecklingsramverk)

Krav

Krav

”Kan du inte fixa lite kod som gör både tabeller och listor till mailen från dom där rapporterna från servern på ett vettigt sätt. Det måste klara mail i både html och text?”

Tester

```
$ cat report.py
#!/usr/bin/env python

"""
Simple module for making tables and lists in both formatted text and html.

We can make a list in both html and txt
    >>> lst = ['Load: 80%', 'Idle: 15%', 'Unknown: 5%']
    >>> do_list(lst)
     * Load: 80%
     * Idle: 15%
     * Unknown: 5%
    >>> do_list(lst, html=True)
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
"""

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    print "Tests done."
```

Tester

```
$ cat report.py
#!/usr/bin/env python

"""
Simple module for making tables and lists in both formatted text and html.

We can make a list in both html and txt
    >>> lst = ['Load: 80%', 'Idle: 15%', 'Unknown: 5%']
    >>> do_list(lst)
     * Load: 80%
     * Idle: 15%
     * Unknown: 5%
    >>> do_list(lst, html=True)
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
"""

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    print "Tests done."
```

doctest: Dokumentationen innehåller testerna.

Tester

```
$ cat report.py
#!/usr/bin/env python

"""
Simple module for making tables and lists in both formatted text and html.

We can make a list in both html and txt
    >>> lst = ['Load: 80%', 'Idle: 15%', 'Unknown: 5%']
    >>> do_list(lst)
     * Load: 80%
     * Idle: 15%
     * Unknown: 5%
    >>> do_list(lst, html=True)
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
"""

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    print "Tests done."
```

doctest: Kör tester om man inte använder koden som modul (typ .dll, .so, …)

Tester

```
$ cat report.py
#!/usr/bin/env python

"""
Simple module for making tables and lists in both formatted text and html.

We can make a list in both html and txt
     >>> lst = ['Load: 80%', 'Idle: 15%', 'Unknown: 5%']
    >>> do_list(lst)
     * Load: 80%
     * Idle: 15%
     * Unknown: 5%
    >>> do_list(lst, html=True)
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
"""

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    print "Tests done."
```

Innehållet till listan.

Tester

```
$ cat report.py
#!/usr/bin/env python

"""
Simple module for making tables and lists in both formatted text and html.

We can make a list in both html and txt
    >>> lst = ['Load: 80%', 'Idle: 15%', 'Unknown: 5%']
      >>> do_list(lst)
       * Load: 80%
       * Idle: 15%
       * Unknown: 5%
    >>> do_list(lst, html=True)
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
"""

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    print "Tests done."
```

Test 1: till text

Tester

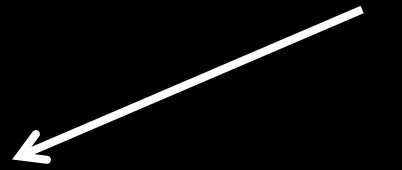
```
$ cat report.py
#!/usr/bin/env python

"""
Simple module for making tables and lists in both formatted text and html.

We can make a list in both html and txt
    >>> lst = ['Load: 80%', 'Idle: 15%', 'Unknown: 5%']
    >>> do_list(lst)
     * Load: 80%
     * Idle: 15%
     * Unknown: 5%
     >>> do_list(lst, html=True)
     <ul>
       <li>Load: 80%</li>
       <li>Idle: 15%</li>
       <li>Unknown: 5%</li>
     </ul>
"""

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    print "Tests done."
```

Test 2: till html



Tester

```
$ cat report.py
#!/usr/bin/env python

"""
Simple module for making tables and lists in both formatted text and html.

We can make a list in both html and txt
    >>> lst = ['Load: 80%', 'Idle: 15%', 'Unknown: 5%']
    >>> do_list(lst)
     * Load: 80%
     * Idle: 15%
     * Unknown: 5%
    >>> do_list(lst, html=True)
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
"""

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    print "Tests done."
```

All kod som finns. Så hur kommer det gå det när vi kör testerna?

Kör Testerna

```
$ python report.py
**********************************************************************
File "report.py", line 8, in __main__
Failed example:
    do_list(lst)
Exception raised:
    Traceback (most recent call last):
      File "/usr/lib/python2.6/doctest.py", line 1253, in __run
        compileflags, 1) in test.globs
      File "<doctest __main__[1]>", line 1, in <module>
        do_list(lst)
    NameError: name 'do_list' is not defined

[...]
```

Uppdatera Koden

```
$ cat report.py
#!/usr/bin/env python

[...]

def do_list(lst, html=False):
    None

[...]
```

Kör Testerna

```
File "report.py", line 8, in __main__
Failed example:
    do_list(lst)
Expected:
     * Load: 80%
     * Idle: 15%
     * Unknown: 5%
Got nothing
**********************************************************************
File "report.py", line 12, in __main__
Failed example:
    do_list(lst, html=True)
Expected:
    <ul>
      <li>Load: 80%</li>
      [...]
```

Uppdatera Koden

```
$ cat report.py
#!/usr/bin/env python

[...]

def do_list(lst, html=False):
    for item in lst:
        print " * %s" % item

[...]
```

Kör Testerna

```
$ python report.py
**********************************************************************
File "report.py", line 12, in __main__
Failed example:
    do_list(lst, html=True)
Expected:
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
Got:
     * Load: 80%
     * Idle: 15%
     * Unknown: 5%
**********************************************************************
[...]
```

Ett grönt test
(syns inte)

Uppdatera Koden

```
[...]

def do_html_list(lst):
    print "<ul>"
    for item in lst:
        print "<li>%s</li>" % item
    print "<ul>"


def do_list(lst, html=False):
    if html:
        do_html_list(lst)
        return
    for item in lst:
        print " * %s" % item

[...]
```

Kör Testerna

```
File "report.py", line 12, in __main__
Failed example:
    do_list(lst, html=True)
Expected:
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
Got:
    <ul>
    <li>Load: 80%</li>
    <li>Idle: 15%</li>
    <li>Unknown: 5%</li>
    <ul>
```

Uppdatera Koden

```
[...]

def do_html_list(lst):
    print "<ul>"
    for item in lst:
        print "  <li>%s</li>" % item
    print "</ul>"

[...]
```

Kör Testerna (verbose den här gången)

```
$ python report.py -v
[...]
Trying:
    do_list(lst, html=True)
Expecting:
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
ok
2 items had no tests:
    __main__.do_html_list
    __main__.do_list
1 items passed all tests:
   3 tests in __main__
3 tests in 3 items.
3 passed and 0 failed.
Test passed.
Tests done.
```

4 - Övning

Romersk Miniräknare

Övning: Romersk Miniräknare

En kund, en vadslagningsbyrå i Rom, behöver mjukvara för att **addera två romerska tal**

Till exempel:

- ”II” + ”II” = ”IV”

Input: två strängar bestående av

- I, V, X, L, C, D, M för 1, 5, 10, 50, 100, 500, 1000

Output: ett tal av samma typ

Övningen består i att skriva testfall

Övning: Romersk Miniräknare – Nya krav

En kund, en vadslagningsbyrå i Rom, behöver kunna **addera och subtrahera ett godtyckligt antal** romerska tal, med vissa medeltida utökningar.

Till exempel:

- ”F” + ”I” + ”Y” = ”CXCI”
- ”IV” – ”II” = ”II”

Input: två strängar bestående av

- I, V, X, L, C, D, M för 1, 5, 10, 50, 100, 500, 1000
- Och även: F, Y, E och Z för 40, 150, 250 och 2000.

Output: ett tal av samma typ

Tillbaka till Exemplet

”Kan du inte fixa lite kod som gör både tabeller och listor till mailen från dom där rapporterna från servern på ett vettigt sätt. Det måste klara mail i både html och text?”

Kör Testerna (verbose den här gången)

```
$ python report.py -v
[...]
Trying:
    do_list(lst, html=True)
Expecting:
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
ok
2 items had no tests:
    __main__.do_html_list
    __main__.do_list
1 items passed all tests:
   3 tests in __main__
3 tests in 3 items.
3 passed and 0 failed.
Test passed.
```

Alla tester OK!

Kör Testerna (verbose den här gången)

```
$ python report.py -v
[...]
Trying:
    do_list(lst, html=True)
Expecting:
    <ul>
      <li>Load: 80%</li>
      <li>Idle: 15%</li>
      <li>Unknown: 5%</li>
    </ul>
ok
2 items had no tests:
    __main__.do_html_list
    __main__.do_list
1 items passed all tests:
   3 tests in __main__
3 tests in 3 items.
3 passed and 0 failed.
```

Men hur är det med täckningen!?!

Kolla Täckning

```
[...]

$ coverage run report.py
Tests done.

[...]

$ coverage report
Name      Stmts   Miss  Cover
---------------------------
report       16      0   100%

[...]
```

Satstäckning

(vi kommer snart till Refactoring)

”Since 1963”

Kan vara ett krav
(speciellt när det handlar om tillämpningar inom det militära eller flyg)

Satstäckning

Testtäckning testar testen

Tester testar koden

Finns inbyggt i till exempel Visual Studio (Ultimate och säkert i fler)

```
24          public Maze(int rows = 3, int cols = 4, List<GridPoint> points = null,
25              char corner = '*', char hor = '*', char ver = '*', char empty = ' ',
26              int size = 2)
27          {
28              // guard dimension
29              if ((rows < 1) || (cols < 1))
30              {
31                  string msg = String.Format("Bad maze dimension ({0}x{1})", rows, cols);
32                  throw new ArgumentException(msg);
33              }
34              this.rows = rows;
35              this.cols = cols;
36              this.size = size;
37
38              // guard bad number of grid points
39              if ((points != null) && (rows * cols != points.Count<GridPoint>()))
40              {
41                  string msg = String.Format("Dimensional mismatch (maze of size "+
42                      "{0}x{1} will not fit {2} elements).",
43                      rows, cols, points.Count<GridPoint>());
44                  throw new ArgumentException(msg);
45              }
46
47              // ok - let's do it
48              this.grid = new GridPoint[this.rows][];
49              for (int i = 0; i < this.rows; i++)
50              {
51                  this.grid[i] = new GridPoint[this.cols];
52              }
53
```

Statiska Tester
(testa utan att köra koden)

Billigt och Effektivt

Statiska Tester
(testa utan att köra koden)

Till exempel **granskning**

- Se till exempel IEEE 1028 för bra metod formell
- Rekord: fånga 97% av alla fel

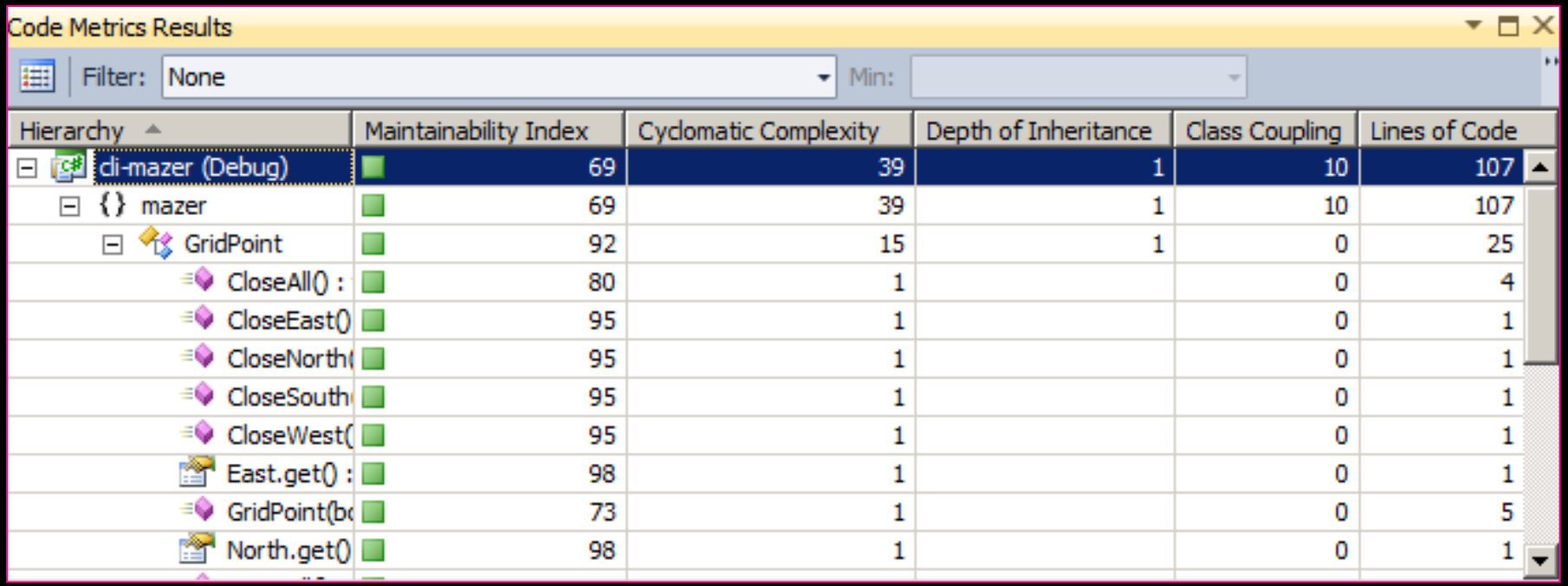
Till exempel **Statisk Kodanalys** med Verktyg

- kompilator, lint, resharper(?), etc
- Kodanalys
- Kodmetrik

Billigt och effektivt!!!

(testa utan att köra koden)

Till exempel **Statisk Kodanalys**


Code Metrics Results
Filter: None
Min:
Hierarchy
Maintainability Index
Cyclomatic Complexity
Depth of Inheritance
Class Coupling
Lines of Code
cli-mazer (Debug) 69 39 1 10 107
mazer 69 39 1 10 107
GridPoint 92 15 1 0 25
CloseAll() : 80 1 0 4
CloseEast() 95 1 0 1
CloseNorth(95 1 0 1
CloseSouth(95 1 0 1
CloseWest(95 1 0 1
East.get() : 98 1 0 1
GridPoint(bo 73 1 0 5
North.get() 98 1 0 1

		char, int)' is never used. Remove the parameter or use it in the method body.		
⚠	19	CA1801 : Microsoft.Usage : Parameter 'empty' of 'Maze.Maze(int, int, List<GridPoint>, char, char, char, char, int)' is never used. Remove the parameter or use it in the method body.	Maze.cs	10
⚠	20	CA1801 : Microsoft.Usage : Parameter 'hor' of 'Maze.Maze(int, int, List<GridPoint>, char, char, char, char, int)' is never used. Remove the parameter or use it in the method body.	Maze.cs	10
⚠	21	CA1801 : Microsoft.Usage : Parameter 'ver' of 'Maze.Maze(int, int, List<GridPoint>, char, char, char, char, int)' is never used. Remove the parameter or use it in the method body.	Maze.cs	10
⚠	13	CA1804 : Microsoft.Performance : 'GridPoint.GridPoint(bool, bool, bool, bool)' declares a variable, 'foo', of type 'string', which is never used or is only assigned to. Use this variable or remove it.	GridPoint.cs	32
⚠	24	CA1814 : Microsoft.Performance : 'Maze.grid' is a multidimensional array. Replace it with a jagged array if possible.	Maze.cs	17

Refactoring

Statisk Kodanalys
kan leda er

Titta i koden

Koden måste vara
förvaltningsbar

Be en kollega kika på koden

Refactoring

Använd bra namn på variabler

Titta på kompilatorn
varningar

Gör inte för mycket på en plats

Statisk Kodanalys
kan leda er

Titta i koden

Koden måste vara
förvaltningsbar

Be en kollega kika på koden

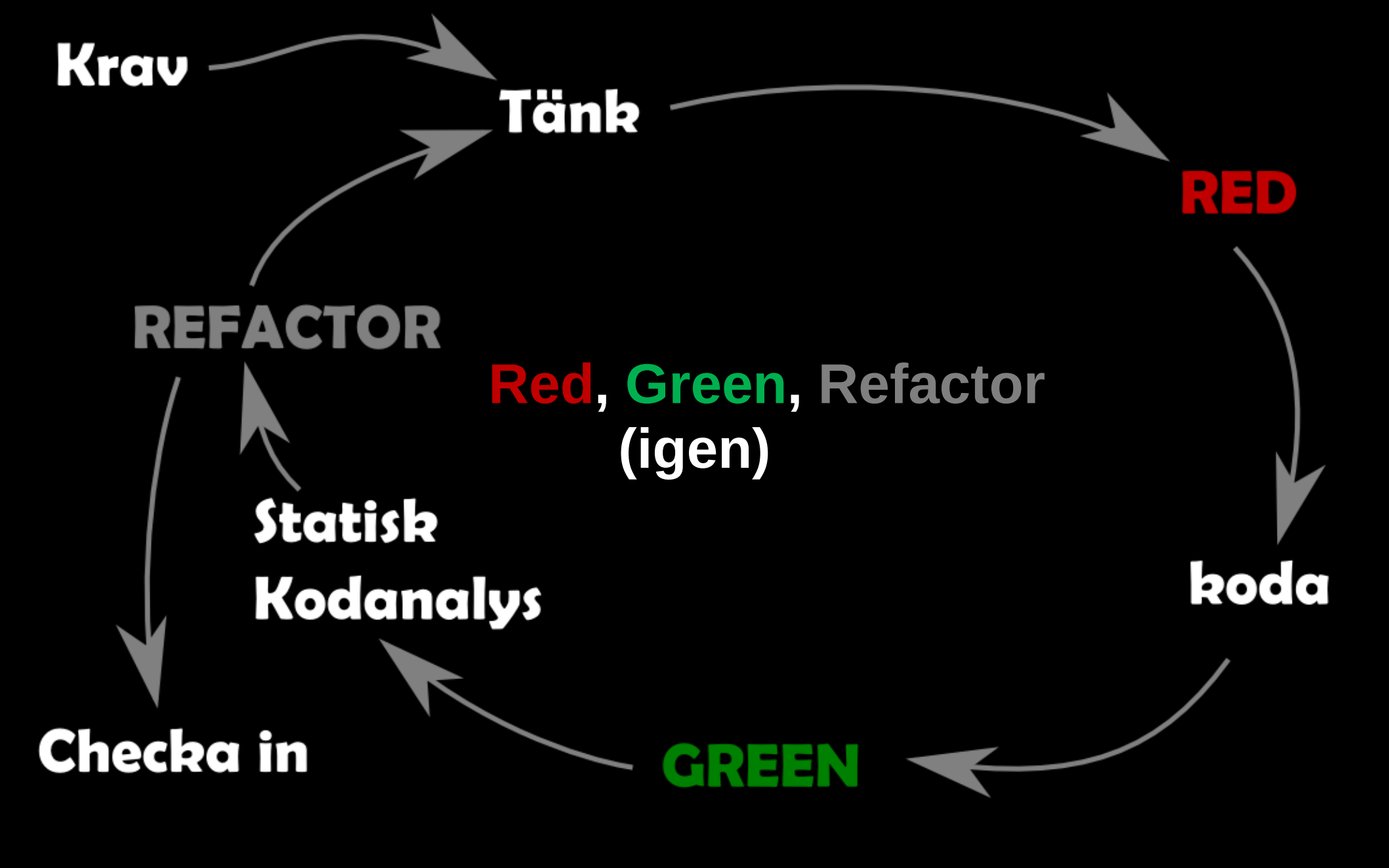
Refactoring

Använd bra namn på variabler

Titta på kompilatorn
varningar

Gör inte för mycket på en plats

Med bra tester vågar
man ändra koden


Krav
Tänk
RED
koda
GREEN
Checka in
Statisk
Kodanalys
REFACTOR
Red, Green, Refactor
(igen)

Exempel

Bryt ut, före

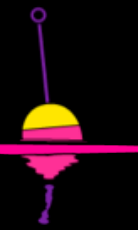
```
public class Customer
{
        private String name;
        private String workPhoneAreaCode;
        private String workPhoneNumber;
        private String homePhoneAreaCode;
        private String homePhoneNumber;
}
```

Bryt ut, efter

```
public class Phone
{
        private String areaCode;
        private String number;
}

public class Customer
{
        private String name;
        private Phone workPhone;
        private Phone homePhone;
}
```

Bryt ut, igen



```
public class Customer
{
        // 4000 lines of code
}
```

Bryt ut, igen – efter

```
public class Customer
{
    // 1000 lines of code
}

public class Phone
{
    // 500 lines of code
}
```

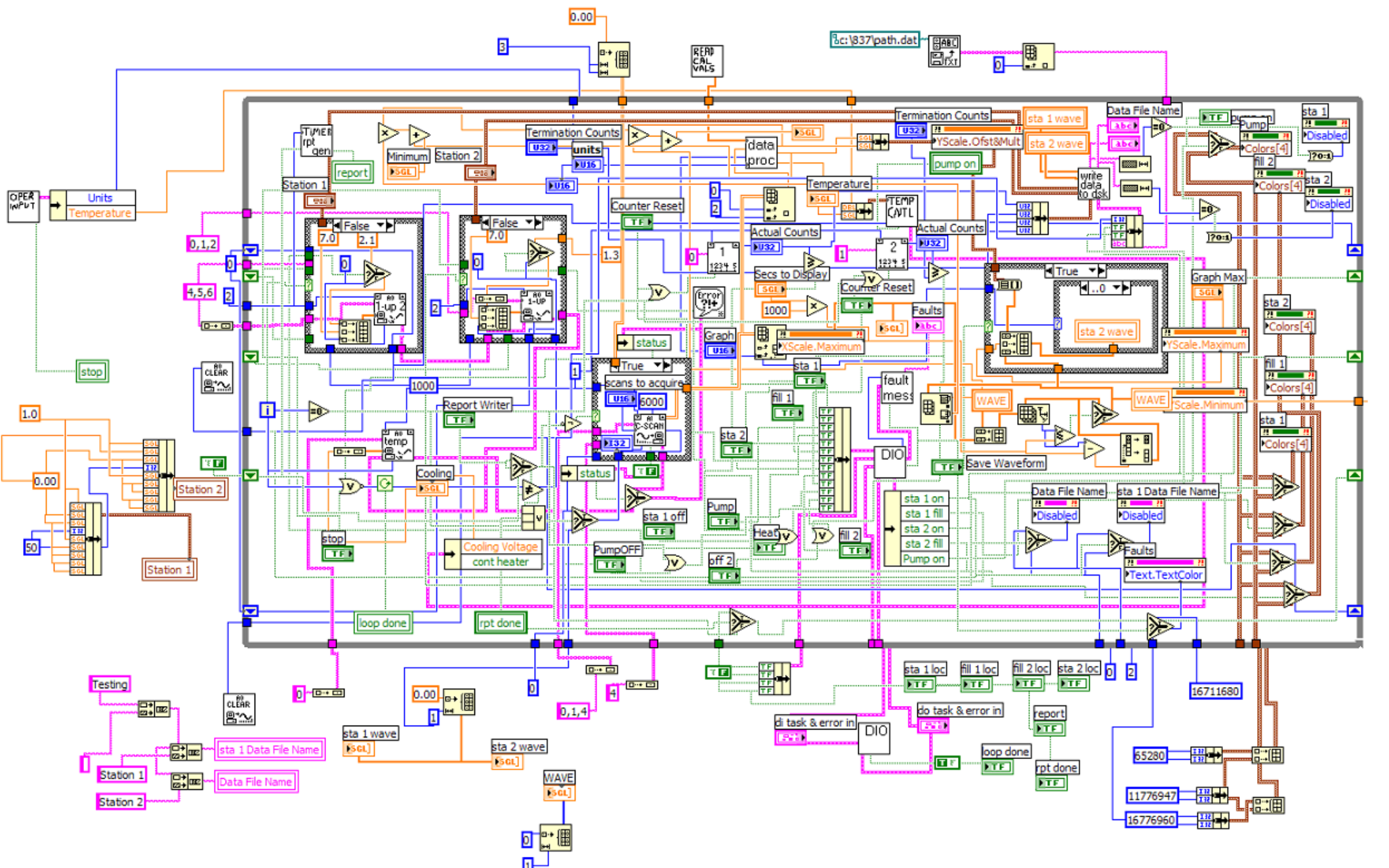
```
public class Purchase
{
    // 500 lines of code
}

public class Log
{
    // 1000 lines of code
}
```

```
// 1000 lines of duplicated code removed
```

Övning

Kollegan: ”kan du inte kolla igenom min kod, det är något som inte känns bra”


OPER INPUT
Units
Temperature
stop
1.0
0.00
50
Station 2
Station 1
Testing
Station 1
Station 2
sta 1 Data File Name
Data File Name
CLEAR
0,1,2
4,5,6
TIMER rpt gen
report
Minimum
Station 2
Station 1
False
7.0
2.1
1-up 2
1-up
Termination Counts
units
data proc
Counter Reset
Actual Counts
Secs to Display
1000
Temperature
TEMP CNTL
pump on
write data to disk
Data File Name
sta 1 wave
sta 2 wave
YScale.Ofst&Mult
Pump
Colors[4]
fill 2
sta 1
sta 2
Disabled
Graph Max
YScale.Maximum
Scale.Minimum
True
..0
status
scans to acquire
5000
C-SCAN
Report Writer
temp
Cooling
Cooling Voltage
cont heater
Graph
XScale.Maximum
Faults
fault mess
WAVE
DIO
Save Waveform
sta 1 on
sta 1 fill
sta 2 on
sta 2 fill
Pump on
sta 1 Data File Name
Text.TextColor
sta 1 off
PumpOFF
off 2
Heat
fill 1
fill 2
loop done
rpt done
sta 1 loc
fill 1 loc
fill 2 loc
sta 2 loc
di task & error in
do task & error in
16711680
65280
11776947
16776960
c:\837\path.dat
READ CAL VALS

```c
/*
 * get BMI
 */
double get(double a, double b)
{
    /* check illegal values */
    if (a < 1)
    {
        a = 71.0;
        b = 177.0;
    }

    if (b < 1)
    {
        a = 71.0;
        b = 177.0;
    }

    /* convert values */
    a *= 0.025;
    b *= 0.45;

    /* //print a and b
     * printf("a %lf\n", a);
     */

    return b/a/a;
}
```

```c
/*
 * simple test program
 */
int main(int argc, char * argv[])
{
    double a = 71.0;
    double b;

    double bmi = get(a, b);

    if ((20 < bmi) && (bmi < 30))
    {
        printf("OK\n");
    }
    else
    {
        printf("FAIL\n");
    }

    return 0;
}
```

Utvecklingsramverk

Enhetstester

Satstäckning

Utvecklingsramverk

(Rapportering)

Statisk Kodanalys

Enhetstester
- doctest

Satstäckning
- coverage.py

Utvecklingsramverk

(Rapportering)

Statisk Kodanalys
- pylint

Enhetstester
- doctest
- CppUnit?
- Unit++?

Satstäckning
- coverage.py
- gcov?

Utvecklingsramverk

(Rapportering)

Statisk Kodanalys
- pylint
- cpplint (by Google)?

Kaffepaus?

5 – Buggar

Buggrapport

Jag tror det är fel
på lorem

Jag får för många ord.

Buggrapport

Jag tror det är fel
på lorem

Jag får för många ord.

Buggrapport

DITT JÄVLA SKITSKRIPT
FYLLER DOM NYA
DISKAR MED SKIT – JAG
HAR JU FÖR FAN SAGT
ÅT DIG!!!

lorem

- En lorem-ipsum-generator för konsolen*
- Genererar ett antal
  - Ord
  - Tecken
  - Rader
- Använder flera olika citatlistor

- Det finns begränsat med tester

* Jag gjorde en (på riktigt, det är sant) 2007 som exemplet baseras på. Den ”lever” på http://code.google.com/p/lorem/

lorem, use cases

```
$ python lorem.py -n 3
lorem ipsum dolor

$ python lorem.py -c 10
lorem ipsu

$ python lorem.py -l 4
lorem ipsum dolor sit amet consetetur sadipscing elitr sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat sed diam
voluptua at vero eos et accusam et justo duo dolores et ea rebum stet
clita
kasd gubergren no sea takimata sanctus est lorem ipsum dolor sit amet
lorem
```

Det finns lite tester

```
$ coverage run test_lorem.py

$ coverage report
Name          Stmts   Miss  Cover
-------------------------------
lorem           117     71    39%
test_lorem        2      0   100%
-------------------------------
TOTAL           119     71    40%
```

Testerna går igenom

```
$ python test_lorem.py -v
Trying:
    import lorem
Expecting nothing
ok

[...]

1 items passed all tests:
   8 tests in lorem.txt
8 tests in 1 items.
8 passed and 0 failed.
Test passed.
```

1 Characters

We can make characters.

```
>>> import lorem
>>> chars = lorem.do_lorem(0, 0, 50, 0)
>>> len(chars)
50
```

Even long strings of characters

```
>>> chars = lorem.do_lorem(0, 0, 5000, 0)
>>> len(chars)
5000
```

2 Words

We can make words - the famous lorem ipsum for example

```
>>> print lorem.do_lorem(2, 0, 0, 0)
lorem ipsum
```

And also longer ones

```
>>> words = lorem.do_lorem(10, 0, 0, 0)
>>> len(words.split(' ')) == 10
True
```

Nytt testfall

```
And also longer ones with more splitting
    >>> words = lorem.do_lorem(200, 0, 0, 1)
    >>> words = words.replace('\n', ' ')
    >>> words = words.replace('  ', ' ')
    >>> len(words.split(' '))
    200
```

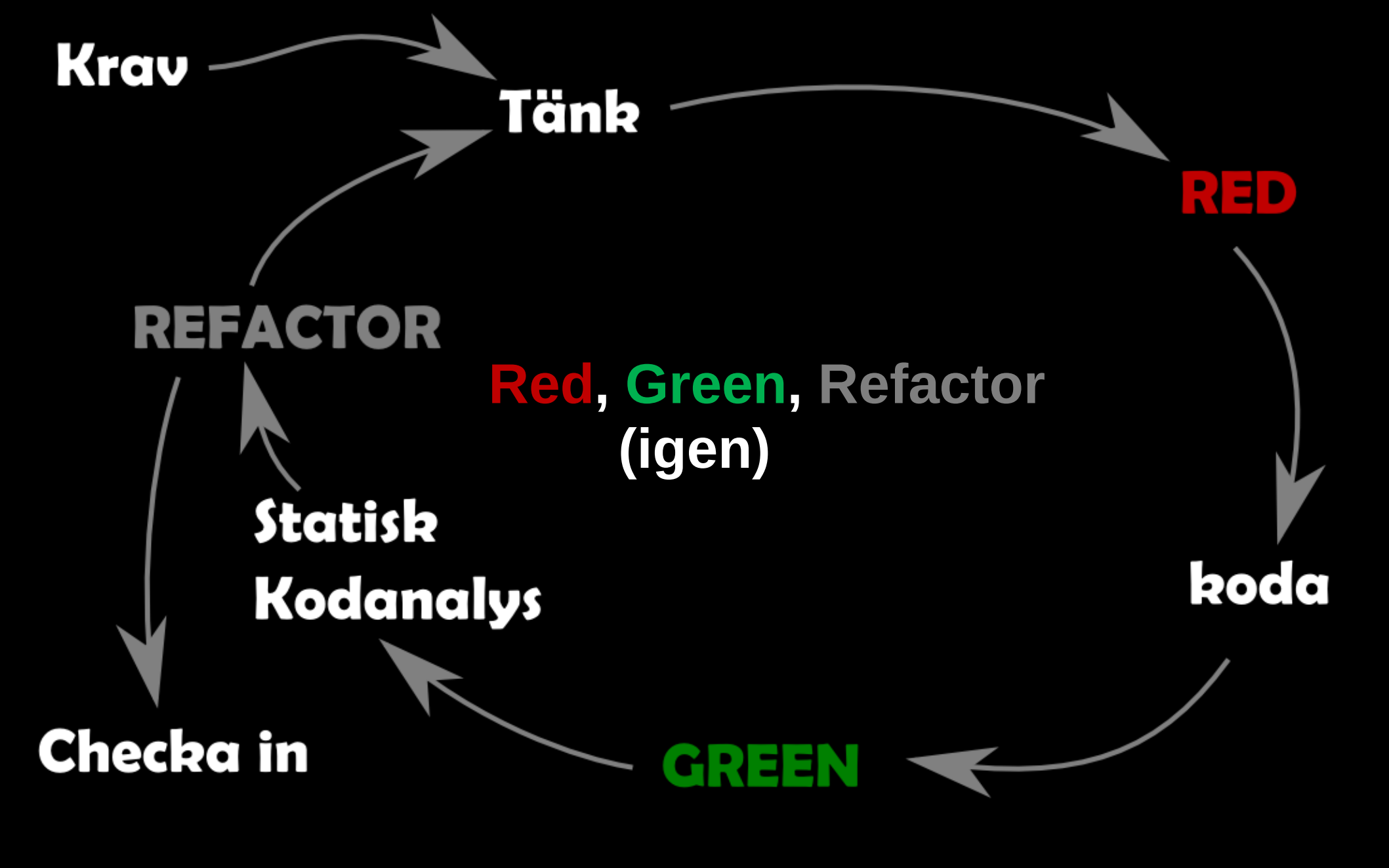
Nytt testfall

```
And also longer ones with more splitting
    >>> words = lorem.do_lorem(200, 0, 0, 1)
    >>> words = words.replace('\n', ' ')
    >>> words = words.replace('  ', ' ')
    >>> len(words.split(' '))
    200


Failed example:
    len(words.split(' '))
Expected:
    200
Got:
    228
```

Sammanfattning

- Vi fick en buggrapport.
- Testerna som fanns hade missat ett problem.
- Vi lade till nya tester som fångar problemet.
- Nu kan vi koda om.
- Sen kollar vi att testet går igenom.
- Och att alla gamla tester fortfarande är ok.
- Det nya testfallet läggs i regressionstesthögen.


Krav
Tänk
RED
koda
GREEN
Checka in
Statisk
Kodanalys
REFACTOR
Red, Green, Refactor
(igen)

6 – Legacy

Kod som är gammal

Kod någon annan skrev

Kod du är rädd för att ändra

Kod utan tester

Vad är Legacy?

Kod som är gammal

Kod någon annan skrev

Kod du är rädd för att ändra

Kod utan tester

Vad är Legacy?

Är det någon som vet vad koden gör?

Hen jobbar inte här längre

Sist vi rörde den koden tog det en vecka att fixa

Kod som är gammal

Kod någon annan skrev

Kod du är rädd för att ändra

Kod utan tester

Vad är Legacy?

”write only”

Är det någon som vet vad koden gör?

Hen jobbar inte här längre

Sist vi rörde den koden tog det en vecka att fixa

Testa Legacy Code – bygg en plattform

- Titta på koden och bli förvirrad
- Säkra upp områden kring problemet
  - Skriv ett test för att se om du fattar koden
  - Testet fallerar
  - Anpassa testet
  - Testet går igenom
- Skriv tillräckligt många tester så att området kring buggen känns säkrat.

Testa Legacy Code – angrip buggen

- När området kring buggen är säkrad angriper du buggen på samma sätt som en vanlig bugg:
  1. Skriv ett test som fallerar.
  2. Lokalisera problemet.
  3. Rätta koden.
  4. Kolla att testet är ok.
  5. Kolla att alla gamla tester fortfarande är ok.

Testa Legacy Code – några tips

- Det kommer ta tid.
- Dokumentera gärna det du lär dig om koden.
- Var beredd att kasta dina ändringar och få börja om.
- Skriv inte tester till övergiven kod (då kommer du inte våga kasta den).
- Kolla täckning.

7 - Övning

Game of Life

Kodkata: Game of life

Conway’s Game of Life

- I ett rutnät finns initialt liv här och där
- Livet överlever om det finns lagom många grannar (2-3 grannar)
- Liv dör om det är för få eller för många grannar (färre än 2 eller fler än 3)
- Liv uppkommer om det finns ett magiskt antal grannar (3 grannar)

Skriv bara tester – ingen kod.

Game of life, nya krav

Det ska vara ett **hexagonalt** nät

- Livet överlever om det finns lagom många grannar (2-4 grannar)
- Liv dör om det är för få eller för många grannar (färre än 2 eller fler än 4)
- Liv uppkommer om det finns ett magiskt antal grannar (3-4 grannar)

Det ska finnas **döende** – det vill säga en cell som lever men inte riktigt är död.

- Det ska ta ett konfigurerbart antal iterationer att för liv att dö
- Döende celler ska kunna väckas till liv lättare än döda celler.

Det ska finnas **fossil** – det vill säga en indikator på att liv en gång funnits.

Skriv bara tester – ingen kod.

Game of life, nya krav igen

Det ska finnas två eller fler **konkurrerande populationer**

- LivA ska lättare dö om det är granne med LivB och vice versa.

Livet ska leva i en **rymd** och inte en plan.

Skriv bara tester – ingen kod.

8 - Referenser

Litteratur

- ***Test Driven Development****: By Example*, av Kent Beck från Addison-Wesley 2002, ISBN: 9780321146533
- ***Refactoring*** *- Improving the Design of Existing Code,* av Martin Fowler, Kent Beck och John Brant från Addison-wesley 1999, ISBN: 9780201485677
- ***Agile Testing*** *- A Practical Guide for Testers and Agile Teams*, av Lisa Crispin och Janet Gregory från Addison-Wesley 2008, ISBN: 9780321534460
- ***Lean-Agile Acceptance Test-Driven Development****: Better Software Through Collaboration,* av Kenneth Pugh från Addison-Wesley 2010, ISBN: 9780321714084.

Referenser

- The History of Software Testing, av Joris Meerts och Dorothy Graham http://www.testingreferences.com/testinghistory.php
- Wikipedia om TDD: http://en.wikipedia.org/wiki/Test-driven_development
- Presentation av Kirrily Robert om TDD: http://www.slideshare.net/Skud/test-driven-development-tutorial
- James Shore om TDD och refactoring: http://www.jamesshore.com/Agile-Book/test_driven_development.html resp http://www.jamesshore.com/Agile-Book/refactoring.html
- Min blogg om test http://www.pererikstrandberg.se/blog/index.cgi?page=KategoriTest
- Kodkator från Coding Dojo: http://codingdojo.org/cgi-bin/wiki.pl?KataCatalogue

9 - Avslutningsvis

Vad tar ni mer er om TDD?

Idag har vi lärt oss…

- TDD är en bra utvecklingsmetod

- Grundmetoden är
  - **Red** – skriv testerna först
  - **Green** – skriv kod
  - **Refactor** – snygga till och förbättre kod

- TDD bör kompletteras med
  - Ramverk för enhetstester
  - Statisk kodanalys
  - Satstäckning