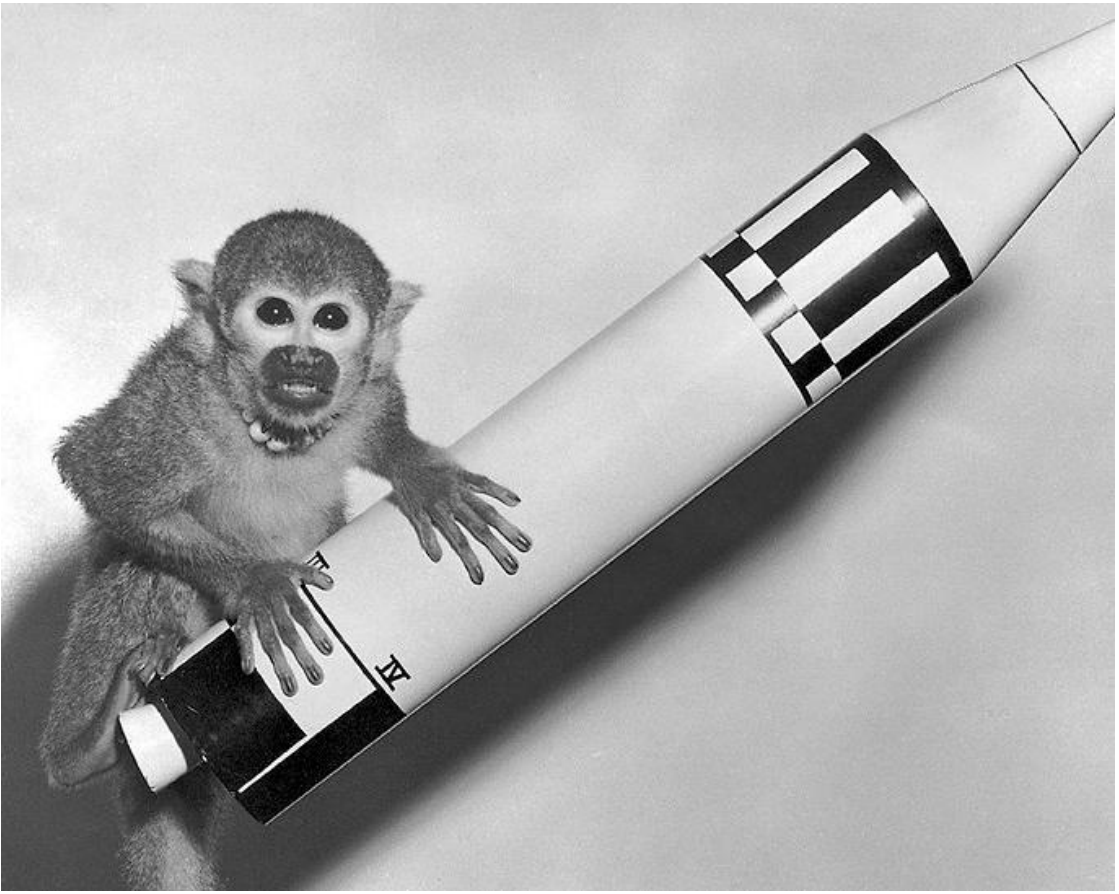
Software Testing
”Hur svårt kan det vara!?!”



Per Erik Strandberg

Våren 2013

Miss Baker… was a squirrel monkey who became… one of the first two animals launched into space by the United States and recovered alive… --Wikipedia

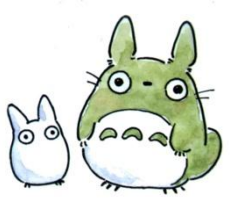
Vad ska vi prata om idag?

- Vad är test?
- Historisk Översikt
- Motivering & Kostnad
- Testindelningar med Exempel på Testuppdrag
- Komponenttester
- Statisk Testning
- Dynamisk Testning
- Testa Tidigt
- Agil testning

- Cirka 2,5 timmar

Per Erik Strandberg

- Civilingenjör och Matematiker
- Fru, 2 barn, hus
- Gillar:
  - Postapokalyptisk retro sci-fi
  - Totoro*
  - ”Datamusik”
  - Windows & GNU/Linux
  - Visual Studio & Emacs
- Jobbat med:
  - Krav, Test, Granskning, Python, .NET, embedded
- Jobbat med test:
  - mer eller mindre sedan 2006
  - i princip uteslutande sedan 2009
- Jobbar på HiQ sedan 2011
- Certifierad Testare 2012
- www.pererikstrandberg.se*



Vad är test?

Vad är test?

- Mål med testning:
  - Hitta fel.
  - Förebygga fel.
  - (Inte lösa fel).
  - Att få förtroende för kvalitetsnivån.
  - Ge stakeholders information om systemet
  - Underlätta utveckling och ge möjlighet att styra utveckling
- Hitta och förebygga fel/risk
- Förtroende för kvalitetsnivån.
  - Uppfyller krav
  - Fungerar som förväntat
  - Uppfyller intressenternas (stakeholders) önskan
- Testar med hjälp av:
  - Statiska metoder och Granskning
  - Dynamiska metoder
- Testa genom hela systemets livscykel
  - Tidigare: efter utveckling
  - Nu: Fokus på tidig testning

Validering & Verifiering

- Validering
  - Bygger vi rätt system?
  - Bekräftelse genom undersökning och genom framläggande av sakliga bevis för att kraven för en specifikt avsedd användning eller specifik tillämpning har uppfyllts. *
  - Med enbart validering riskerar man att bygga precis det som våra stakeholdes vill ha – utan att det fungerar.
- Verifiering
  - Bygger vi systemet rätt?
  - Bekräftelse genom undersökning och genom framläggande av sakliga bevis för specificerade krav har uppfyllts. *
  - Med enbart verifiering riskerar man att bygga ett helt oanvändbart men felfritt system.

* SSTB, Svensk Ordlista v 2.1-1

Sju Testprinciper

1. Test visar att det finns fel (inte att det inte finns fel).
2. Uttömmande testning är omöjlig
3. Tidig testning lönar sig
4. Ansamlingar av fel – där det finns ett fel finns det ofta fler.
5. Immunitets- paradoxen – att bara ha en typ av test gör efter ett tag varken till eller från.
6. Test beror på sammanhang/kontext – att bromstestet i labbet gick bra behöver inte betyda att bromstestet på stambanan kommer gå bra.
7. Frånvaro-av-fel-fallgropen. Om testerna inte hittar fel kan det vara brister i testerna och systemet.

Historisk Översikt

Historisk Översikt – 1800 - 1957

- 1843 – Ada Lovelace ”**possible source of error**” i Charles Babbages Analytical Machine
- 1878 – Thomas Edison benämner generella ingenjörsproblem som bug.
- 1931 – Baffle Ball ”**No bugs in this game**”
- 1946 – Grace Hopper hittar insekter (mal) i reläer. Populariserar termen bug.
- 1957 – Sputnik I, II och Laika

Historisk Översikt 1958-1977

Project Mercury was the first human spaceflight program of the United States. It ran from 1959 through 1963 with two goals: putting a human in orbit around the Earth, and doing it before the Soviet Union...

-- Wikipedia

- 1958 – Första teamet för mjukvarutester (**Project Mercury**)
- 1969 – Edsger Dijkstra ”testing show the presence, not the absence of bugs”
- 1970 – Waterfall
- 1975 – Microsoft
- 1976 – Apple
- 1976/77 – Kodkomplexitet, Mätetal och Design Patterns (för hus)

Historisk Översikt 1980-tal

- 1983 – **TPS Reports**!
  (IEEE 829 Software Test Documentation)
- 1985 – Första kommersiella verktyget för test
- 1986 – V-modellen
- 1987 – Use Case, Software Design Patterns
- 1988 – Undersökande tester (exploratory testing), första Issue Trackern, Spiral Model

Historisk Översikt 1990-tal

Welcome Visitors

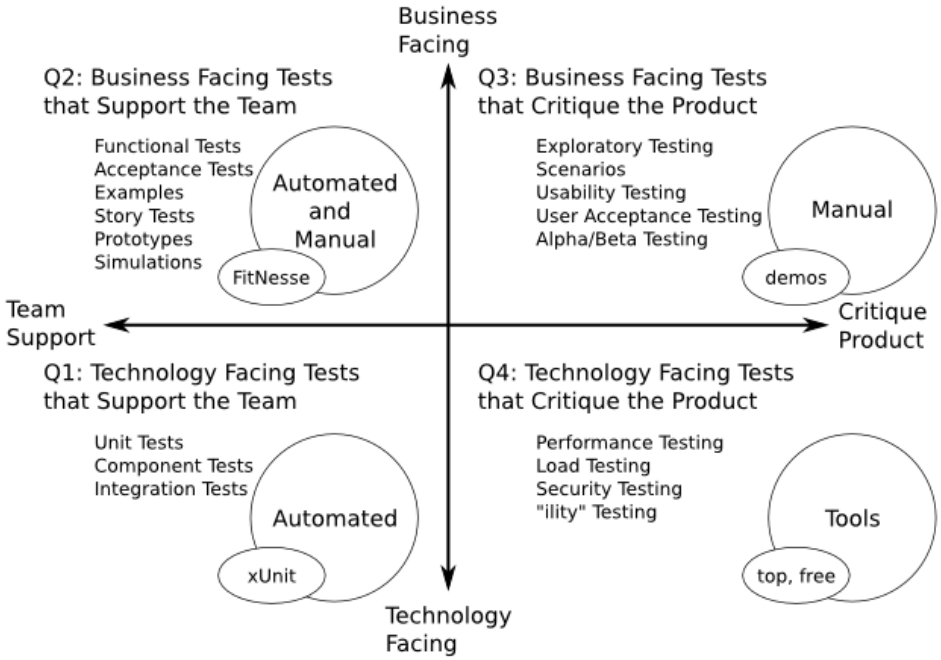
Welcome to WikiWikiWeb, also known as WardsWiki or just Wiki. A lot of people had their first wiki experience here. This community has been around since 1995 and consists of many people. We always accept newcomers with valuable contributions. If you haven't used a wiki before, be prepared for a bit of CultureShock. The usefulness of Wiki is in the freedom, simplicity, and power it offers.

This site's primary focus is PeopleProjectsAndPatterns in SoftwareDevelopment. However, it is more than just an InformalHistoryOfProgrammingIdeas. It started there, but the theme has created a culture and DramaticIdentity all its own. All Wiki content is WorkInProgress. Most of all, this is a forum where people share ideas! It changes as people come and go. Much of the information here is subjective. If you are looking for a dedicated reference site, try WikiPedia; WikiIsNotWikipedia!

- 1993 – Scrum
- 1994 – **WikiWikiWeb***, Ward Cunningham
- 1996 – Google Online
- 1996 – Extreme Programming

* 1994 kom verktyget – 1995 kom första wikin. Bilden ovan kommer från ”the C2 wiki”. (Cunningham & Cunningham: http://c2.com/cgi/wiki?WelcomeVisitors)

Historisk Översikt 2000-tal



- 2000 – Continuous Integration (Fowler)
- 2002 – TDD (Test-Driven Development) och ISTQB (International Software Testing Qualifications Board)
- 2003 – **Agila Testkvadranter**
- 2004 – Selenium
- 2007 – iPhone
- 2008 – HTC Dream (Första Androidtelefonen)

Historisk Översikt - Idag

- **Mobile application testing?**
- Crowd sourced testing?
- Testing-as-a-Service?
- Cross cloud testing?
- Test data generation and management?
- Business intelligence testing?

Motivering & Kostnad

Angripa från olika håll

1. Varför testa?
2. Skräckexempel
3. I vilken fas hittar vi problem?
4. Vad kostar en sen eller tidig bugg?
5. I vilken funktionalitet hittar vi fel?

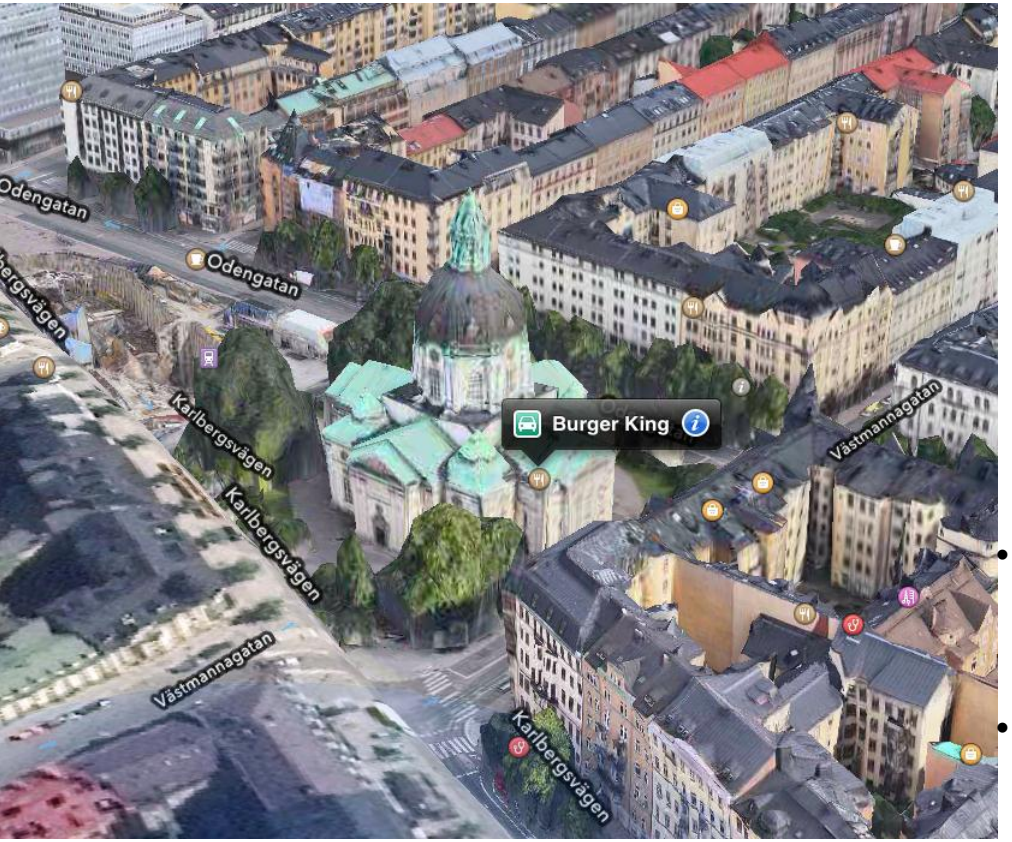
Varför Testa?

- Mjukvara finns överallt
- Folk gör fel
  - Tidspress
  - Komplex kod
  - Komplex infrastruktur
  - Förändrad teknologi
  - Många system
- Test av system och dokumentation identifierar/reducerar risk och kan bidra till kvalitet
- Test gör det möjligt att mäta kvalitet

- “Unit tests are the only way of even remotely convincing your customers and friends your code doesn't completely suck... “

–Martin Aspeli

Kostnad

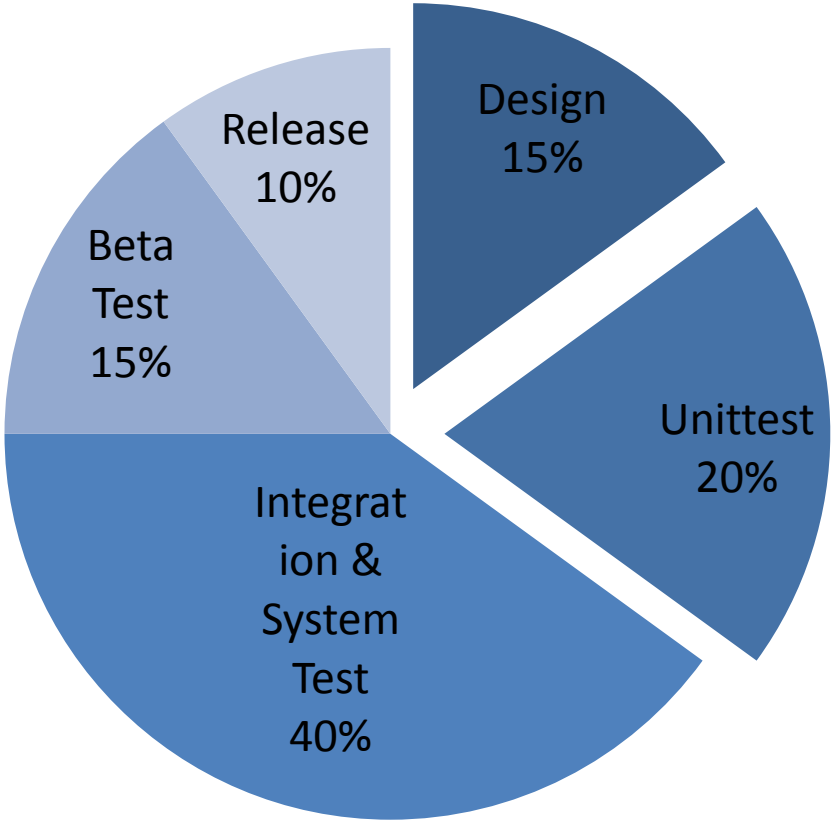


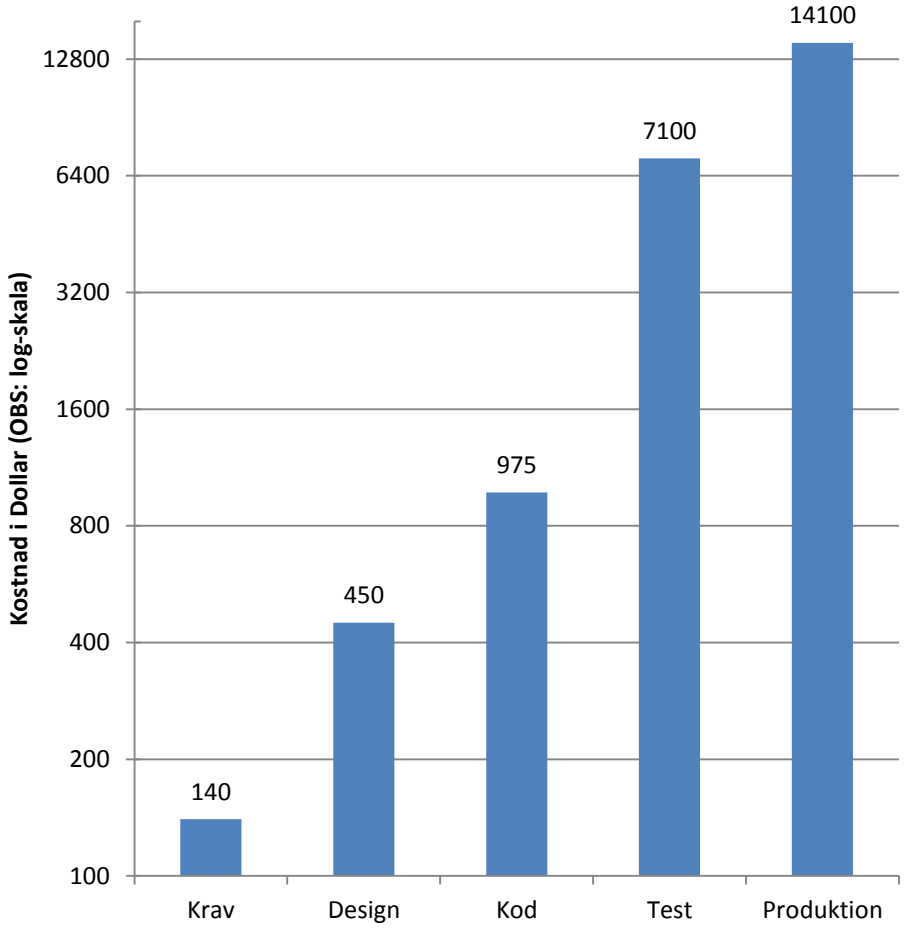
Gustav Vasa Kyrka invigdes 1906 och rymmer 1200 sittplatser – men inga hamburgare.

- Fältbuggar kan vara dyra.
  - Knight Capital Groups robothandel skenade aug 2012 till en kostnad av 400 miljoner dollar
  - Två av NASAs mars-landare hade problem (det ena orsakades av imperial/metric problem) till en kostnad av 357 miljoner dollar
  - Ariane 5 fick problem med ett kompensationssystem (konvertering från 64 bitar till 16): 500 miljoner dollar.
  - Misslyckad moln-tjänst för Amerikanska armen: 2.7 miljarder dollar
  - Y2K: uppskattningsvis 200 miljarder dollar
  - Skabbiga kartor i Apple Maps: 30 miljarder dollar i förlorat börsvärde
- Att fixa en bugg som hittas i designfasen kostar cirka 1% av motsvarande kostnad när den hittas i fält.
- Testning kan och bör göras i hela mjukvarans livscykel!
- Tumregel: ”Så länge testarna sparar/tjänar pengar åt projektet ska man testa.”

Var hittas defekter och vad kostar de?

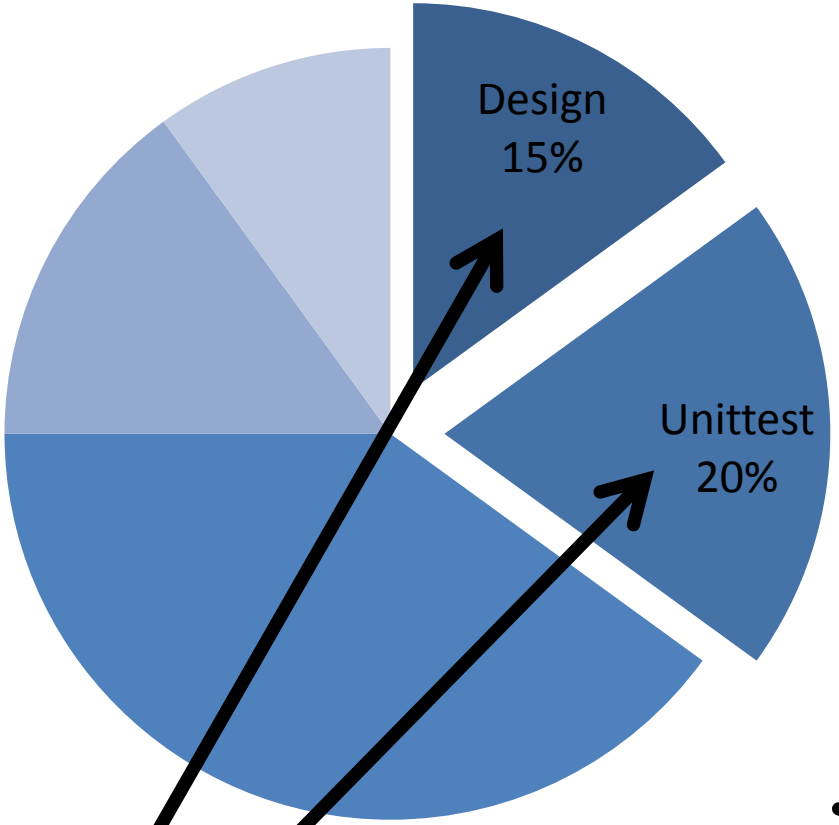
(en sammanställning från open source projekt av source ninja)

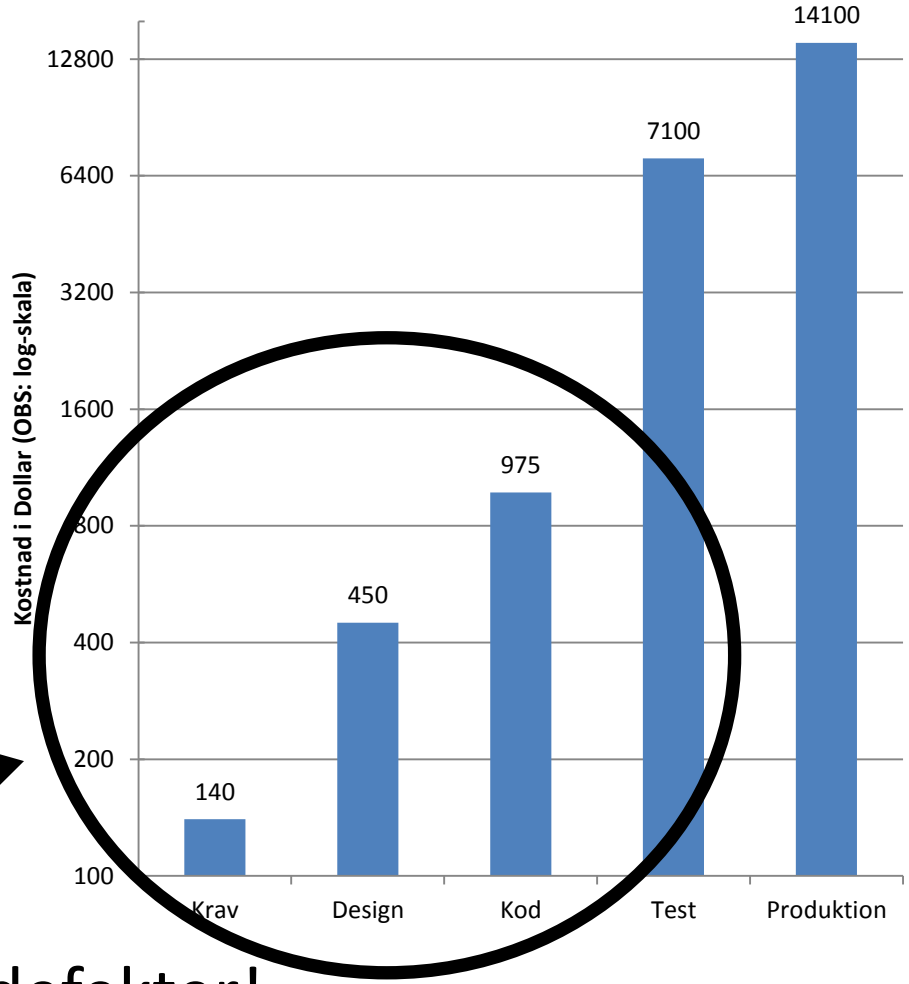




Var hittas defekter och vad kostar de?

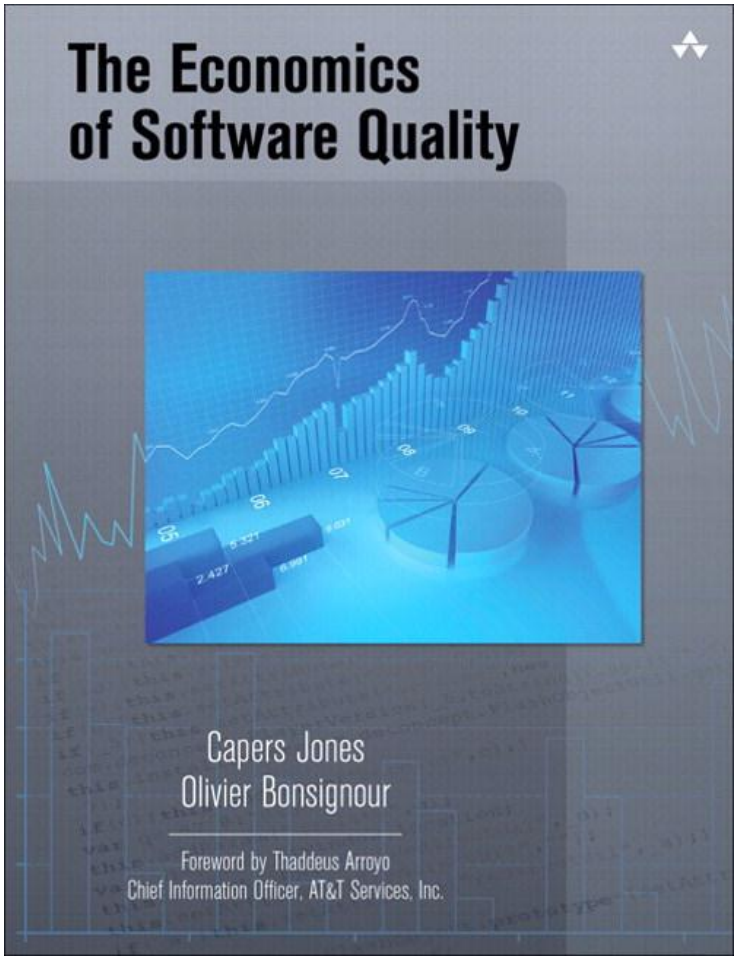
(en sammanställning från open source projekt av source ninja)





Här är det lämpligt att hitta defekter!

Economics of Software Quality, Intro



- **Capers Jones** & Olivier Bonsignour har studerat
- 13000 mjukvaruprojekt
- Från 1973 till 2011
- Kravmetoder
- Testmetoder
- Utvecklingsmetoder
- Gurun Rex Black rekommenderar den om man vill ha siffror

ESQ, vinst med kvalitet

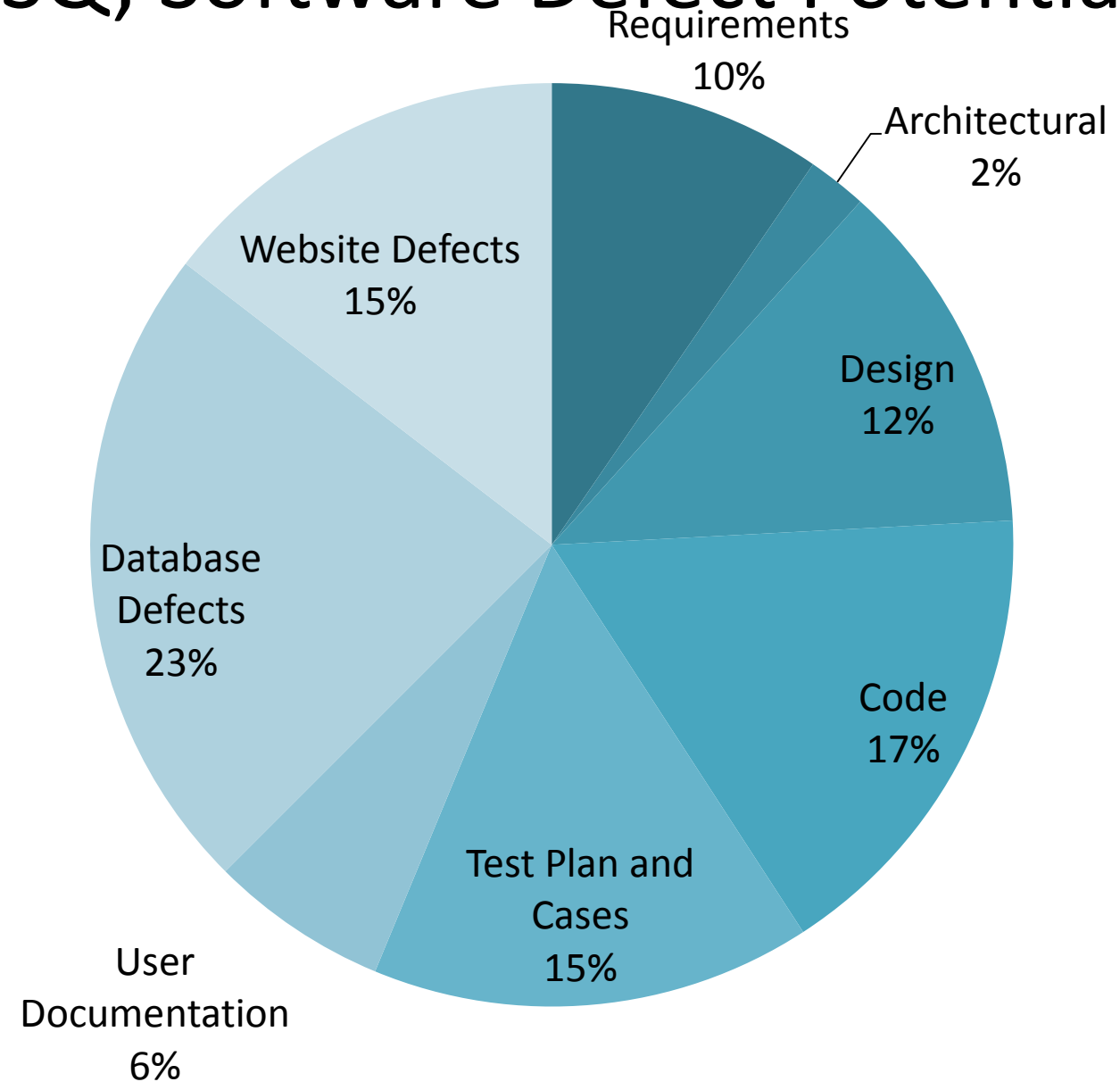
Storlek	Låg kvalitet	Medel kvalitet	Hög kvalitet
Litet (10 FP)	6,875 $	6,250 $ (91%)	5,938 $ (86%)
Stort (1 kFP)	1,039,889 $	920,256 $ (88%)	846,636 $ (81%)
Enormt (100 kFP)	507,767,782 $	433,989,557 $ (85%)	381,910,810 $ (75%)

- Hög kvalitet lönar sig oavsett storlek på projekt
- Hög kvalitet lönar sig mer ju större projekt man har
  - Ett litet projekt med hög kvalitet kostar 86% av motsvarande med lågt
  - Ett enormt projekt med hög kvalitet kostar 75% av motsvarande medlågt

Delar av tabell 1.1 som visar typisk kostnad för ett mjukvaruprojekt i olika storlek med olika kvalitet.

FP = Function Point, ett slags mått på komplexitet.

ESQ, Software Defect Potential



ESQ, 12 bästa faktorerna för mjukvarukvalitet

- Low defect potential
- Effective defect prevention methods
- High defect detection methods
- High defect removal efficiency
- **Use of pretest inspections**
- **Use of pretest static analysis**
- **Use of formal test case design**
- Good ease of learning
- Good ease of use
- Good technical support
- High user satisfaction
- Good Warranty

ESQ, sämsta faktorerna för mjukvarukvalitet

- Executice indifference to high quality
- Management indifference to high quality
- Team indifference to high quality
- Customer indifference to high quality
- **Cost-per-defect quality metrics (OBS!)**
- CMMI < 2
- Lines of code quality measures
- Litigation for poor quality (consequential, contractual, financial loss, safety, medical)
- No warranty expressed or implied

Return On Investment - Räkneexempel

- Detektera bugg med team av testare
  - Kostade 400 k$
  - Hittade 1500 buggar
  - Eller: 267 $/bugg
- Fixa en bugg som fallerat internt
  - Kostade 1250 k$
  - Fixade 1500 bugg
  - Eller: 833 $/bugg
- Fixa bugg som fallerat externt
  - Kostade 1500 k$
  - Fixade 500 bugg
  - Eller: 3000 $/bugg

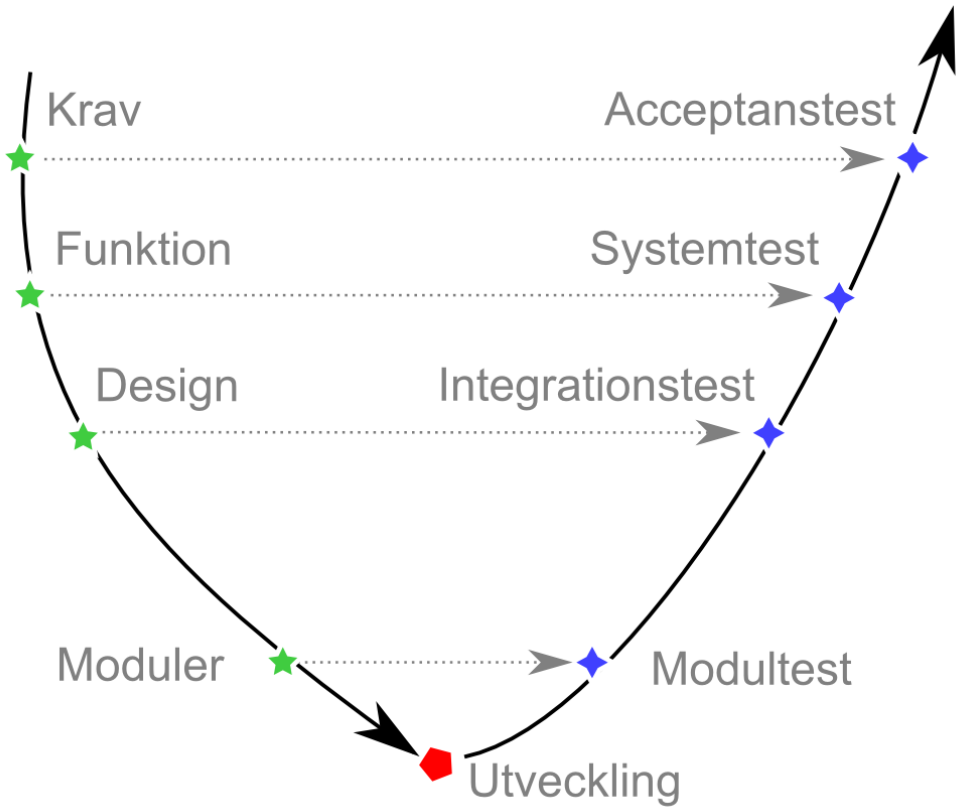
- Faktiska utgifter: 400 k$ + 1250 k$ + 1500 k$ = 3.15 M$
- Motsvarande utgifter helt utan testare: 3000 $/bugg * (1500 + 500) bugg = 6 M$
- Besparing: 6 M$ - 3.15 M$ = 2.85 M$
- Return On Investment = besparing/detektionskostnad = 2.85 M$ / 400 k$ =

713 %

Källa: Rex Black Consulting Services, Inc., *Free Tool for Calculating Software Testing ROI*
http://www.rbcs-us.com/blog/2010/09/09/free-tool-for-calculating-software-testing-roi/

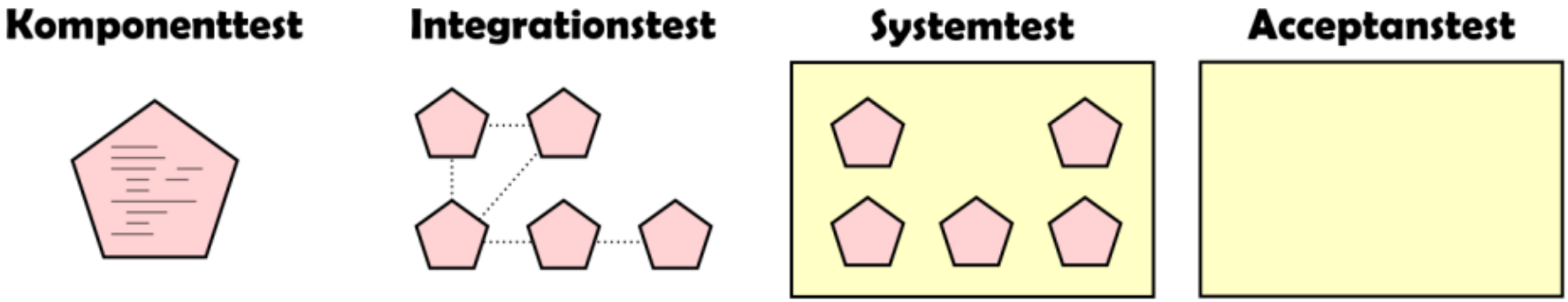
Olika sätt att dela in tester

V-modellen



- Klassiker (1986)
- Första processen som involverade testare tidigt
- Först olika nivåer av designspec och samtidig testspec
- Utveckling
- Sen nivåer av test

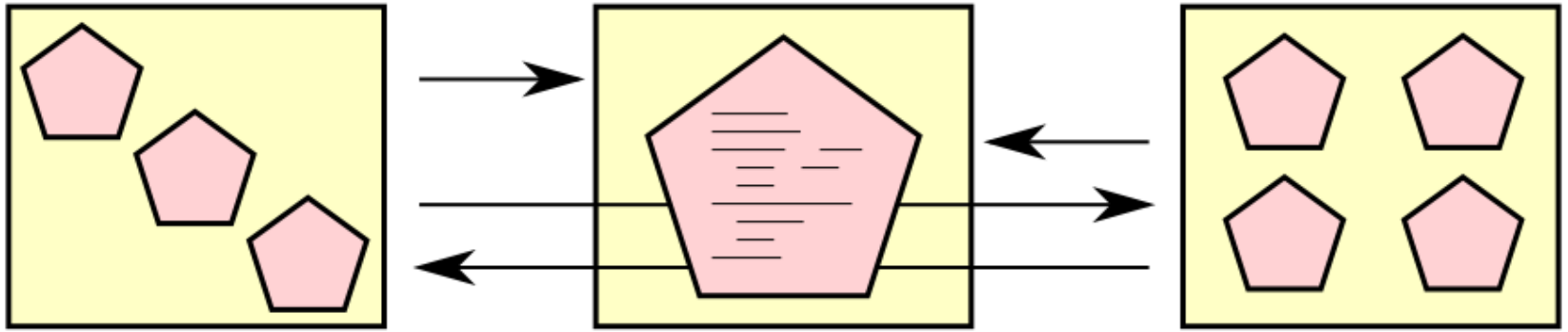
Testnivåer



- Komponenttester
  - Testa en komponent i taget
  - Tänk xUnit
  - Kan hitta 10-50% av fel
- Integrationstester
  - Testa två eller fler komponenter med varandra
  - Undvik big bang integration

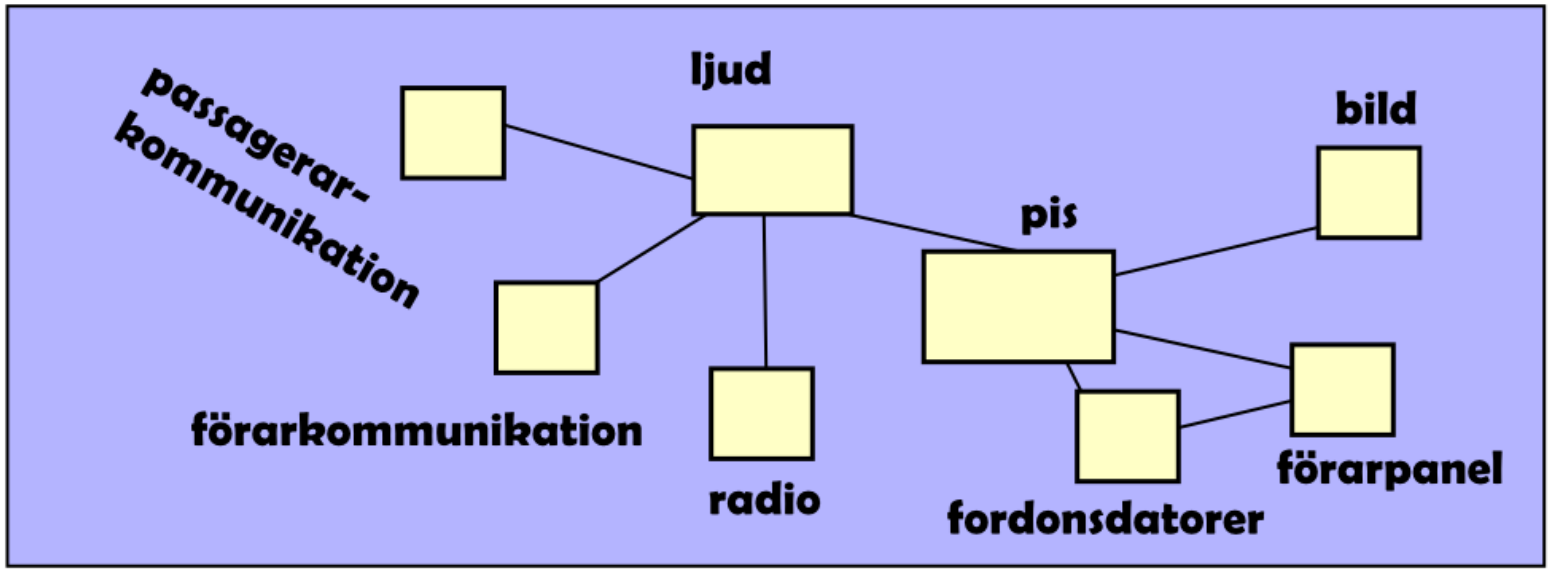
- Systemtester
  - Testa hela systemet
  - Efterlikna målmiljön
  - Kan hitta upp till 85%
- Acceptanstester
  - Testar inte ”för att hitta fel”
  - Målet är att kunden ska få tilltro till systemet
  - Utförs ofta av kunden
- System av System…

Industriell Kommunikation



- Produkten
  - Embeddedlösning
  - Har ett OS och ett shell
  - Industriell Kommunikation
    - Ethernet, Seriellt
    - Brandvägg, VLAN, PoE, …
    - Och allt annat
- Tester
  - Svårt eller omöjligt att testa i utvecklingsmiljö
  - Testar på slutprodukter
  - Aktiverar en funktion i taget (om det går) eller så lite som möjligt
  - Målet är komponenttester, men får inslag av integration och systemtest

Passagerarinformation i Fordon



- Produkten
  - Består av System av System
  - Ljudsystemet (med passagerarkommunikation) hade mer än 20 enheter med processor och mjukvara och ett antal ”dumma” enheter.
  - Hela fordonet består av allt från små enkla system med knapp, mikrofon, högtalare och dioder till enorma och säkerhetskritiska system (högspänning, motor, broms, dörrar, etc)
  - Mängder av olika protokoll mellan system (från signalkabel till olika mer eller mindre standardiserade protokoll)
- Tester för Passagerarinformation
  - Det mesta är omöjligt att testa i utvecklingsmiljö
  - Svårt att testa på slutprodukt (fordonet är stor och finns på en annan kontinent)
  - Simulatorer och delsystem i labb
  - Tidsbokning och ”trängsel” i testlabbet
  - Mycket manuella tester och undersökande tester
  - Integrations-, system- och acceptanstester

Andra indelningar av tester

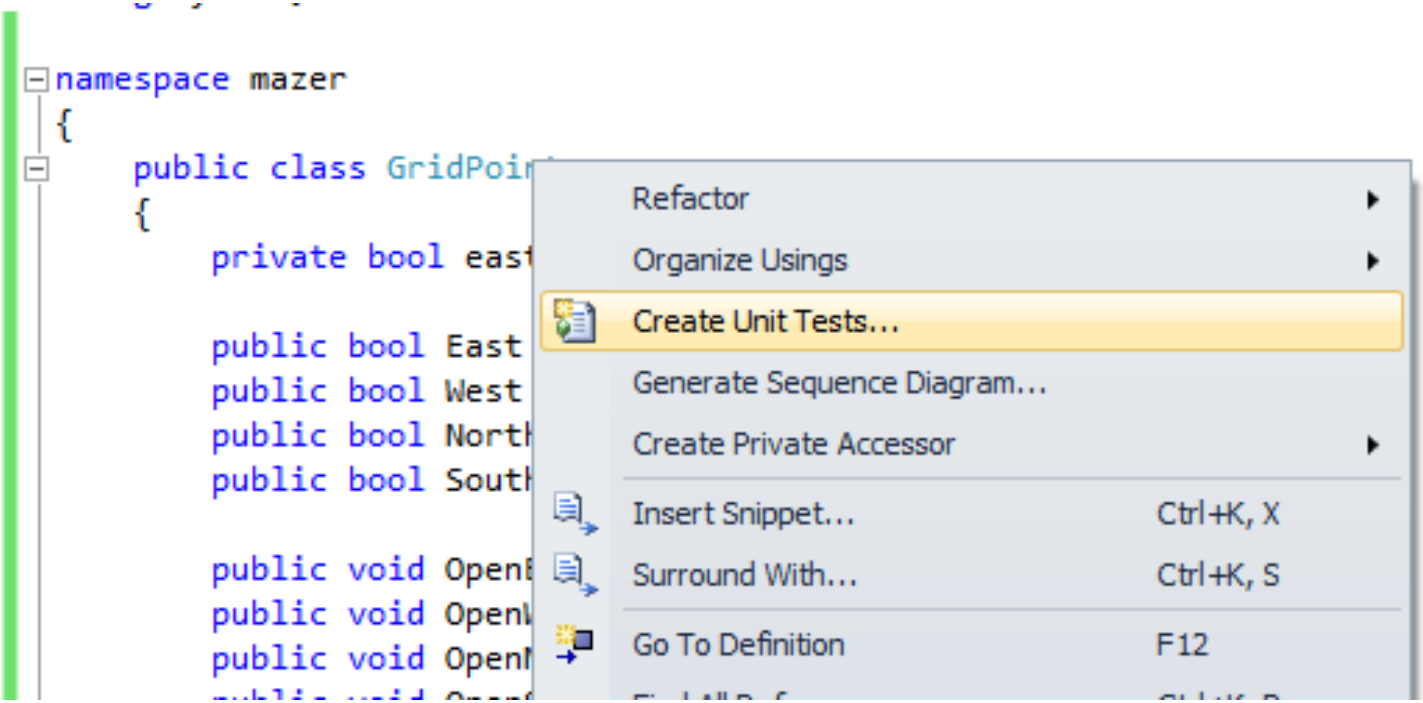
- Vilka kravtyper de matchar
  - Funktionell, etc
- Statiska/Dynamiska
  - Exekveras mjukvaran?
- Säkert fler sätt

- Funktionella tester: Vad systemet gör aka **black box**
  - Raketen träffar målet och exploderar.
- Icke-funktionella tester: Prestanda, last, användbarhet, …
  - Raketen är strömlinjeformat och har mindre än X % friktion vid Y km/h.
- Strukturella tester: till exempel kodtäckning och -komplexitet aka **whitebox**
  - Ingen modul får ha en cyklomatisk komplexitet över 15.
  - Ingen metod får ha över 50 rader kod
  - Enhetstesterna ska ge 100% satstäckning
- Regressionstester (omtester)
  - ”Felen vi såg vid de första provskjutningarna testas regelbundet och vi har inte sett dem igen”.

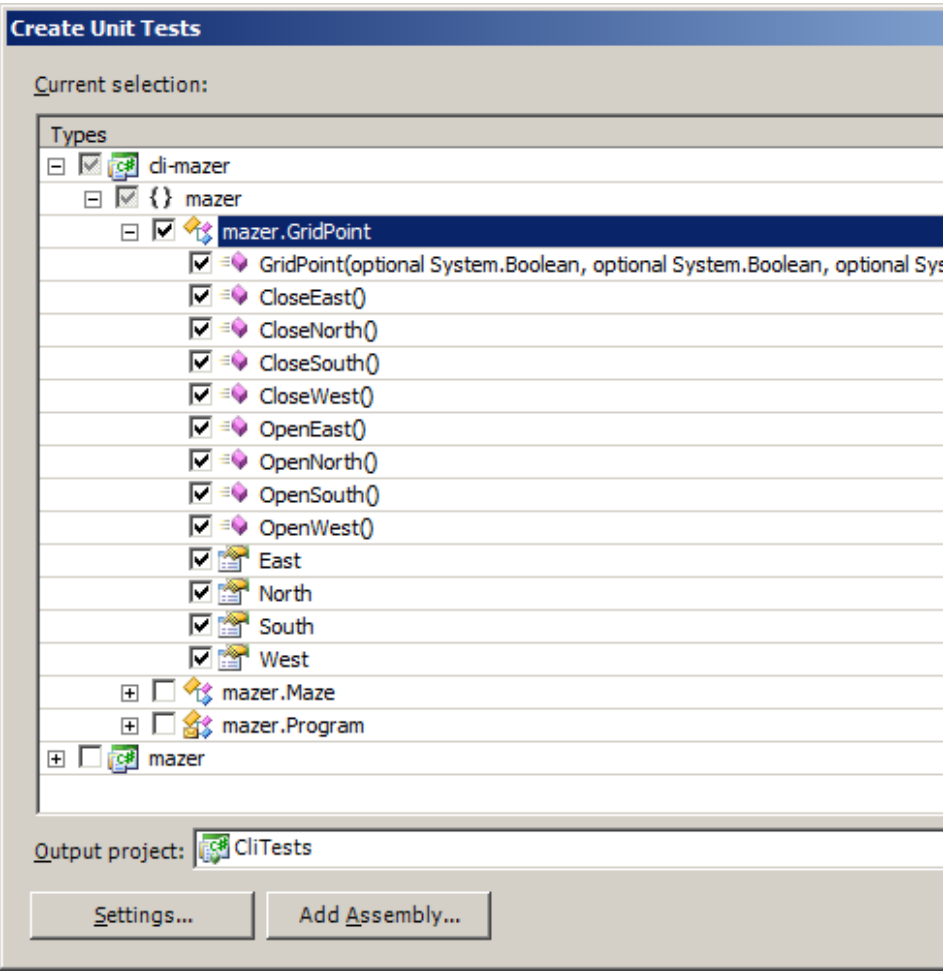
Mer om Komponenttester
(eller Unit tests)

Komponenttester i Visual Studio

1. Högerklicka



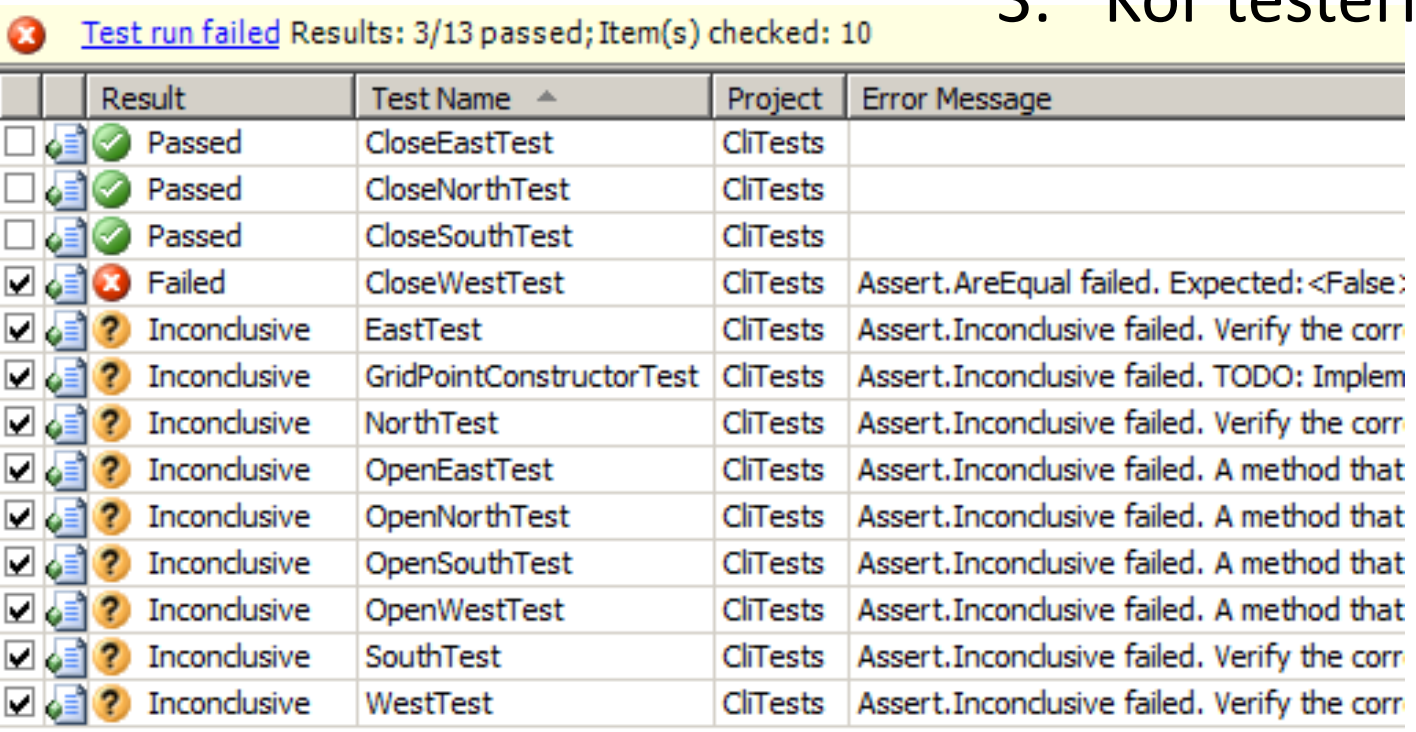
Komponenttester i Visual Studio



1. Högerklicka
2. Välj Metoder

Komponenttester i Visual Studio

1. Högerklicka
2. Välj Metoder
3. Kör testerna



Statiska Tester

Statiska Tester

- Viktigt och billigt.
- Granskning
  - Till exempel verifiera en designspecifikation mot en industristandard
- Statisk Analys med Verktyg
  - Till exempel att kompilera och ta hand om alla varningar

Statiska Tester - Granskning

- Asbilligt!

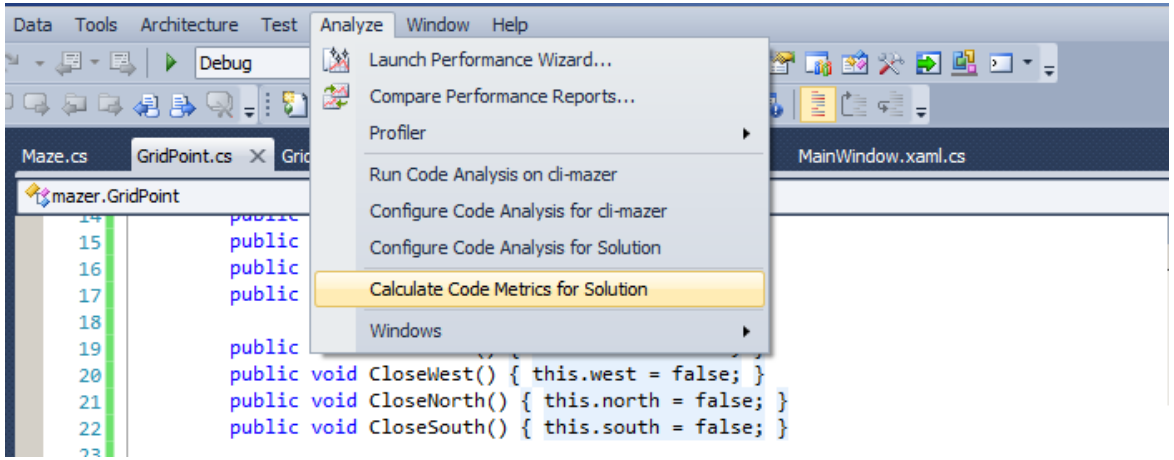
- IEEE Standard for Software Reviews (IEEE 1028)
- Från formell till informell
- Formellt 8 steg
  1. Uppfyller vi startkriterierna?
  2. Planering
  3. Start
  4. Individuella förberedelser
  5. Granskningsmöte
  6. Rättning/Omarbete
  7. Uppföljning
  8. Uppfyller vi avslutskriterierna?

Statisk Analys med Verktyg

- Asbilligt!

- Kodanalys
  - Använd kompilatorn
  - Använd Lint, StyleCop, etc
- Kodmetrik
  - Vanligt att titta på Komplexitet, till exempel Cyklomatisk Komplexitet
  - Storlek (tumregel: för varje 1000 rader kod över 1000 i en fil får man offra ett finger till den onde guden Malloc)
- I Visual Studio (Ultimate och kanske fler) är det inbyggt!

Visual Studio Code Metrics



- Hitta komplexitet
- Rikta refaktorering

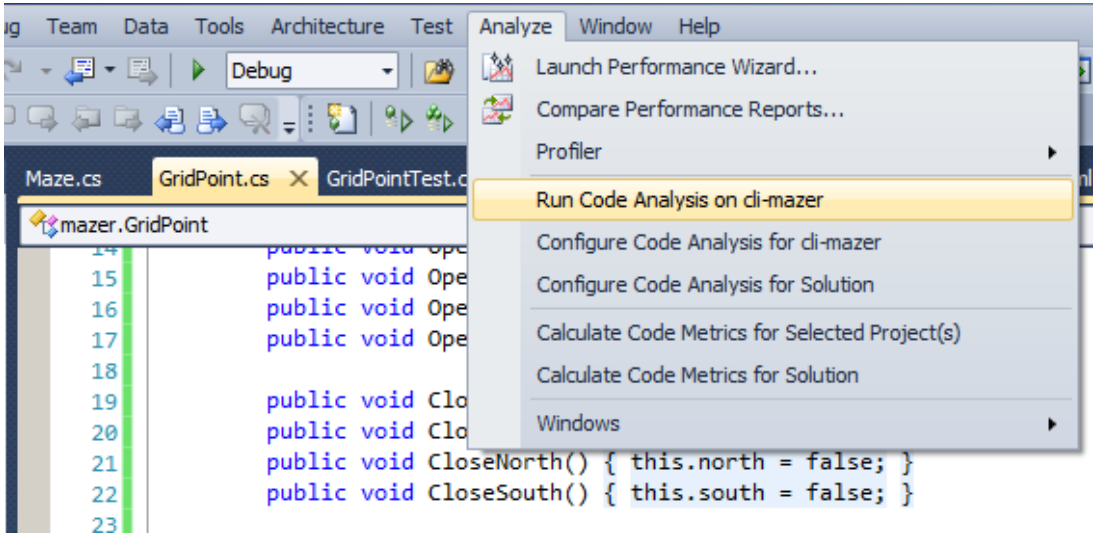
- Asbilligt!

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

Visual Studio Code Analysis



- Hittar Dataflödesproblem
  - Oanvända variabler
  - Använda men odefinierade variabler
- Kan hitta dålig kodstil
  - string foobar
  - int tmp
  - double d
- Kan hitta mycket mer

- Asbilligt!

		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

En gång till så att ni inte glömmer det:

Asbilligt!

Dynamiska Tester

Dynamiska Tester

- Skamtest (Smoke Test)
- Specifikationsbaserad (Black Box)
  - Ekvivalensklassindelning
- Strukturbaserad (White Box)
  - Kodtäckning
- Erfarenhetsbaserad (Exploratory/Utforskande)

Ekvivalensklasser - Intro

(Ekvivalensklasser är ett exempel på en **testdesignteknik**)

- Testa allt går inte
- Måste välja **rätt** tester
  - Till exempel enhetstester
- Dela upp variabler i klasser där systemet beter sig lika (valid och invalid)
- Kombinera olika indata
  - Alla valid x Alla valid
  - En invalid i taget

Ekvivalensklasser - Exempel

```
def calc_bonus(baseprice, member_years):
    """Calculate the bonus."""

    # check input
    if not (check_val(member_years, 0, 50)
        raise new MemberException("Bad member years!")
    elif not check_val(baseprice, 0, 10000):
        raise new PriceException("Impossible price!")

    # high price bonus
    if baseprice > 3000:
        bonus = 7.5
    elif baseprice > 2000:
        bonus = 5
    elif baseprice > 1000:
        bonus = 2.5
    else:
        bonus = 0

    # member bonus
    if member_years > 5:
        bonus += 5
    elif member_years > 1:
        bonus += 2.5

    # never more than 10%
    max_bonus = 10
    if bonus > max_bonus:
        bonus = max_bonus

    # return bonus
    return baseprice * bonus
```

- baseprice
  - Ok för:
    - 0-1000
    - 1001-2000
    - 2001-3000
    - 3001-10000
  - Inte Ok för:
    - -Inf - -1
    - 10001 - +Inf
- member_years
  - OK för:
    - 0-1
    - 2-5
    - 5-50
  - Inte OK för:
    - -Inf – 0
    - 51- + Inf

Ekvivalensklasser – Resultat

1 typfall

baseprice	member_years
500	2

4 otillåtna (invalid) fall

baseprice	member_years
-5	2
11500	2
500	-7
500	77

12 valid + 4 invalid
= 16 testfall

Totalt 12 tillåtna (valid) med kombationer

baseprice	member_years
500	0
500 (typfallet)	2 (typfallet)
500	7
1100	0
1100	2
1100	7
2350	0
2350	2
2350	7
8750	0
8750	2
8750	7

Testtäckning - Intro

- Tester testar koden
- Testtäckning testar testen
- Olika typer
  - Satstäckning*
  - Beslutstäckning
- Bra för att styra fortsatt insats av enhetstester
- Inbyggt i Visual Studio (Ultimate och säkert fler)

*nej – inte så som ni tänker

Testtäckning
Visual Studio Code Coverage

GridPoint	10	35,71 %
.ctor(bool,bool,boo…	0	0,00 %
CloseAll()	5	100,00 %
CloseEast()	0	0,00 %
CloseNorth()	0	0,00 %
CloseSouth()	0	0,00 %
CloseWest()	0	0,00 %
OpenAll()	5	100,00 %
OpenEast()	0	0,00 %
OpenNorth()	0	0,00 %
OpenSouth()	0	0,00 %
OpenWest()	0	0,00 %
get_East()	0	0,00 %
get_North()	0	0,00 %
get_South()	0	0,00 %
get_West()	0	0,00 %

```
public Maze(int rows = 3, int cols = 4, List<GridPoint> points = null,
    char corner = '*', char hor = '*', char ver = '*', char empty = ' ',
    int size = 2)
{
    // guard dimension
    if ((rows < 1) || (cols < 1))
    {
        string msg = String.Format("Bad maze dimension ({0}x{1})", rows, c
        throw new ArgumentException(msg);
    }
    this.rows = rows;
    this.cols = cols;
    this.size = size;

    // guard bad number of grid points
    if ((points != null) && (rows * cols != points.Count<GridPoint>()))
    {
        string msg = String.Format("Dimensional mismatch (maze of size "+
            "{0}x{1} will not fit {2} elements).",
            rows, cols, points.Count<GridPoint>());
        throw new ArgumentException(msg);
    }

    // ok - let's do it
    this.grid = new GridPoint[this.rows][];
    for (int i = 0; i < this.rows; i++)
    {
        this.grid[i] = new GridPoint[this.cols];
    }
```

Blått = Testat

Grönt = Delvis Testat

Rött = Inte testat

Kaffe?

Testa tidigt

Testa tidigt

- Oavsett om man jobbar med vattenfall, V-modellen eller agilt så brukar det vara ”faser”.
  - Krav
  - Design
  - Implementering
  - Test
- Ofta är det slutdatumet som är ”hårt”.

Testa tidigt

- ”Ibland” kan till exempel utvecklingen ta lite mer tid än man tänkt sig
- Slutdatumet är ”hårt”.
- Vi minskar mängden test

Testa tidigt

Så här borde vi göra:

- Testar vi istället tidigt så…
  - Hittar vi fel tidigt
  - Underlättar vi utveckling
  - Får vi färre fel i produktion
  - Testarna får ”ingen” startsträcka när testningen börjar
- Hur gör man?
  - Granska specifikationer
  - Skriv acceptanstester innan en implementation (eller riktigt tidigt i en iteration och skriv enhetstester under sprinten)
  - Skriv testfall innan du har något att testa på
  - Fokusera på områden med hög användning, hög prioritet eller hög risk.
  - Testmiljö. Automatisering?
  - Fler förslag?
  - Synlighet – vad är testat och hur går det?

Komma igång med test

- OK, det funkar – då skriver jag ett test så det inte pajar.
- OK, så här vill jag att det ska funka – då skriver jag ett test och när det går igenom funkar det.
- Vad har jag gjort?
- Hur kan jag paja det?
- Hur kan jag bryta ner testningen?
  - Moduler
  - Integration
  - System
  - Beta-tester
- Vad är viktigast?
- Keep It Simple

Övningsuppgift

Övningsuppgift

Ekvivalensklassindelning:

- Räkna testfall

Tillståndsmaskin:

- Hur många testfall behövs till?

Ekvivalensklassindelning

```
/// <summary>
/// Lock the file or optionally compete for priority of the file.
/// </summary>
/// <param name="leave">If true then ignore file if already in use.</param>
/// <param name="addLog">If true then log when attempting to lock file.</param>
/// <param name="priority">Priority of process.
/// Prio = 1, 2 or 3. A process with prio 3 can never lock the file.
/// </param>
/// <param name="process">Process ID, 1000 - 9999.</param>
/// <returns>True if the file is now locked by the process.</returns>
private bool LockFile(bool leave, bool addLog, int priority, int processID)
{
    if (!this.CheckInt(priority, 1, 3))
        throw new ArgumentException("Illegal Priority!");

    if (!this.CheckProcessID(processID))
        throw new ArgumentException("Illegal Process ID!");

    if (priority == 3)
        return false;

    if (this.LockPrio > priority)
        return false;

    if (addLog)
        this.Log(priority, processID);

    return this.Lock(priority, processID);
}
```

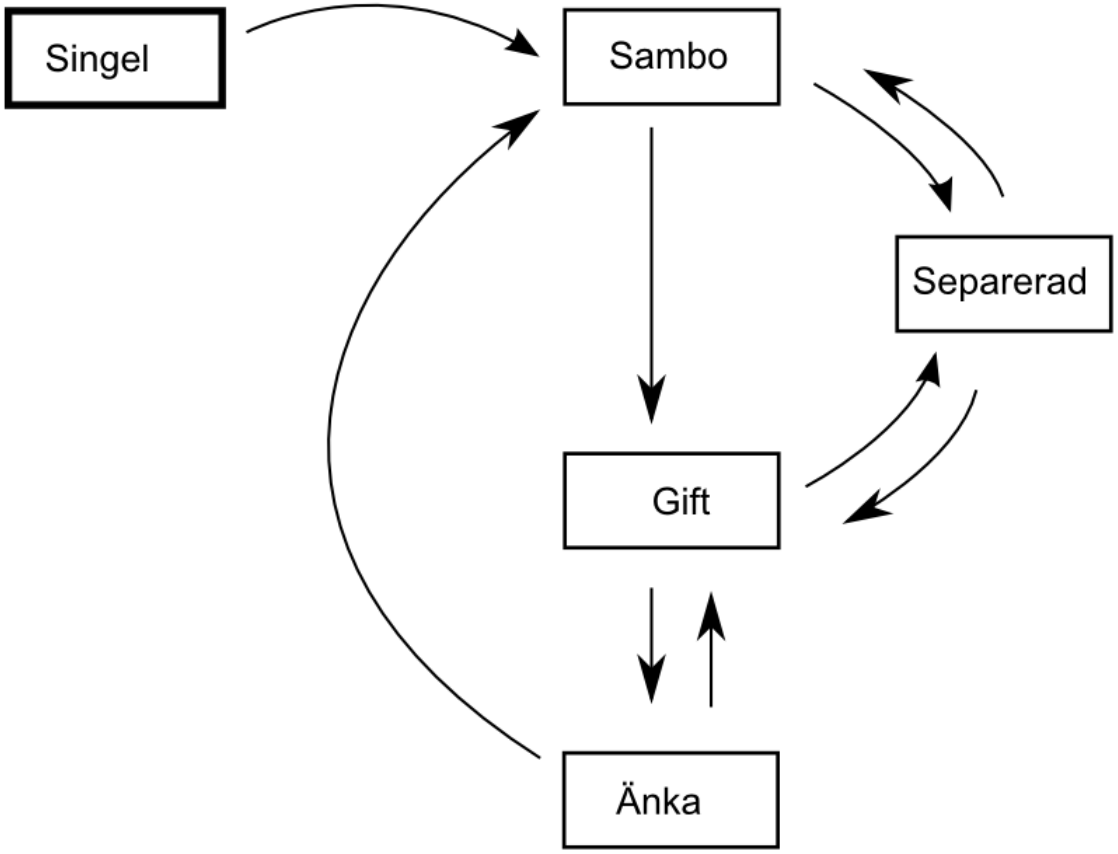
Ekvivalensklassindelning

leave	[True], [False]
addLog	[True], [False]
priority	]-Inf, 0], [1, 2], [3], [4, +Inf[
processID	]-Inf, 999], [1000, 9999], [10000, +Inf[

Valid Cases	2 * 2 * 2 * 1 = 8
Invalid Cases	0 + 0 + 2 + 2 = 4

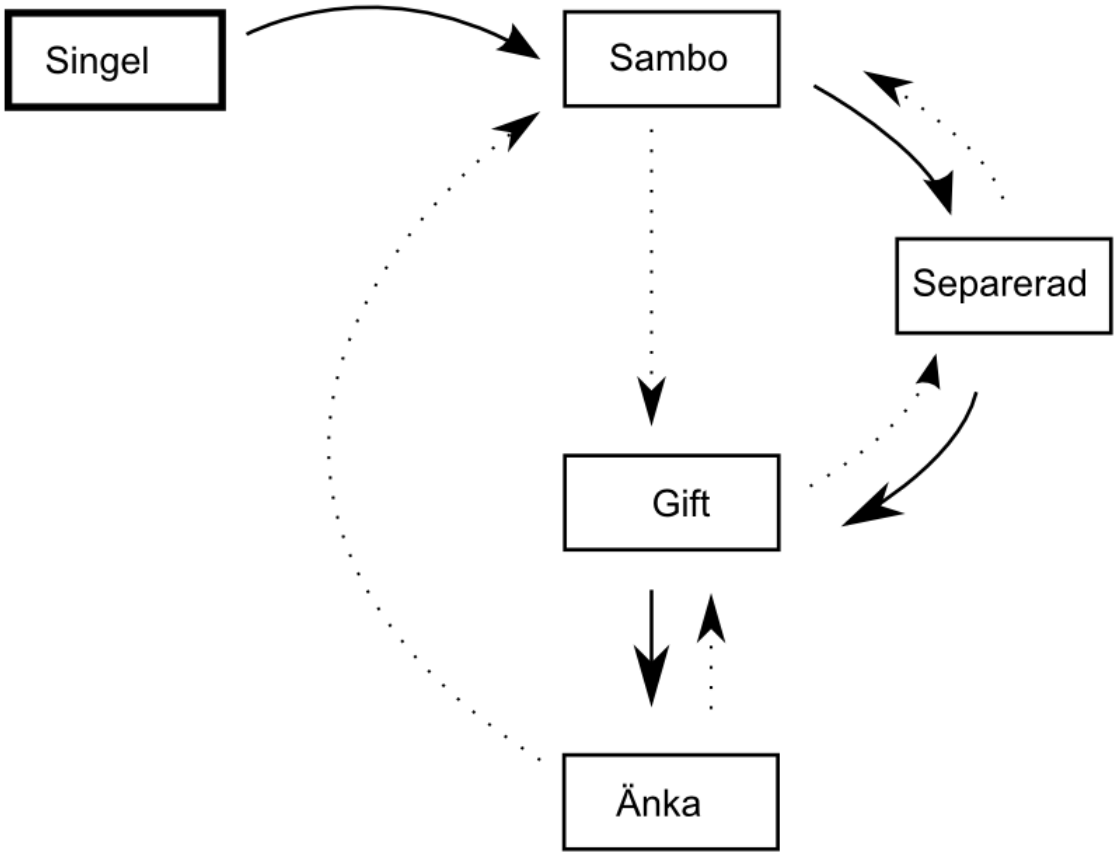
minsta testinsatsen för att

Testa Alla Tillstånd



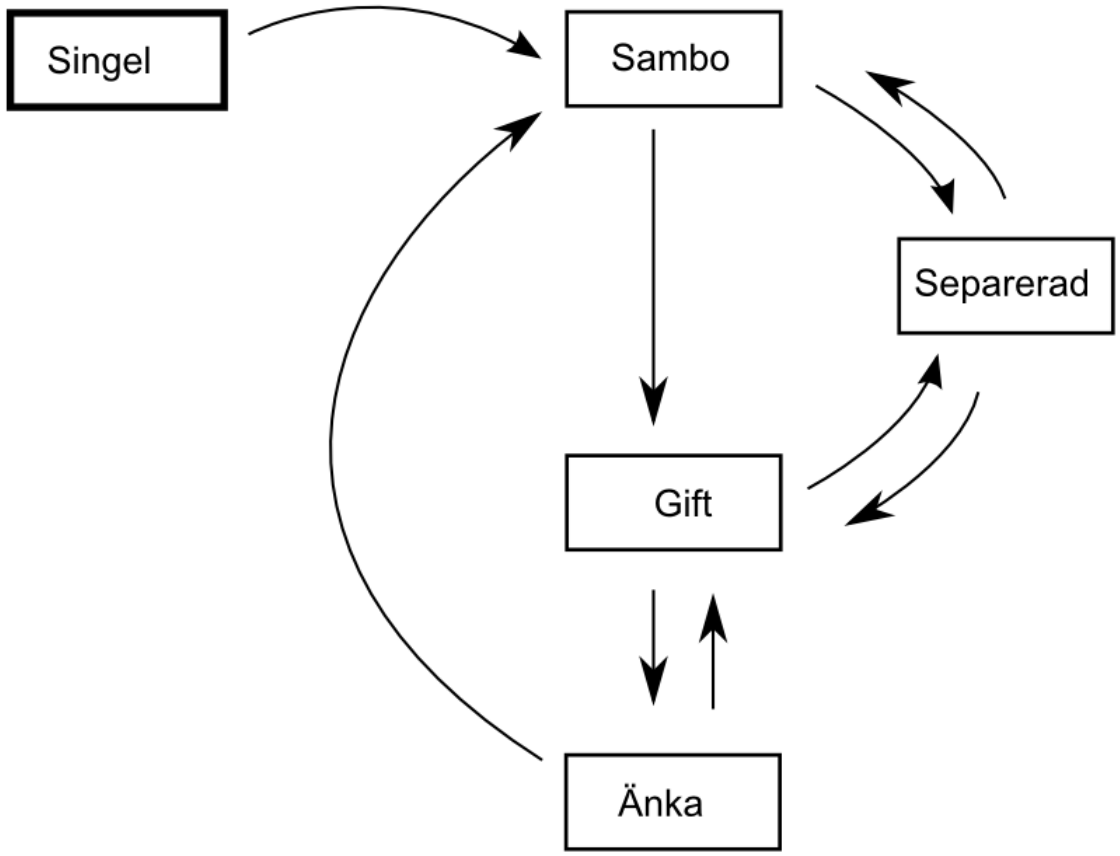
minsta testinsatsen för att

Testa Alla Tillstånd



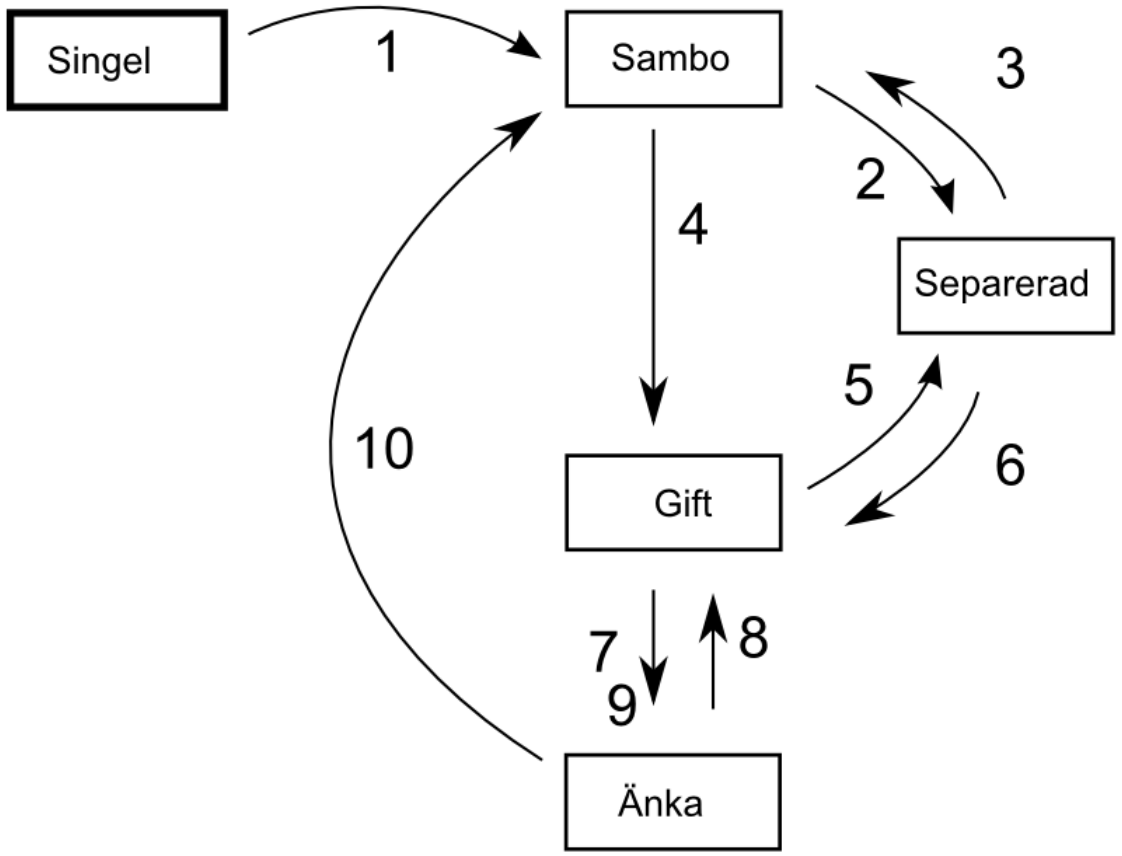
minsta testinsatsen för att

Testa Alla Övergångar



minsta testinsatsen för att

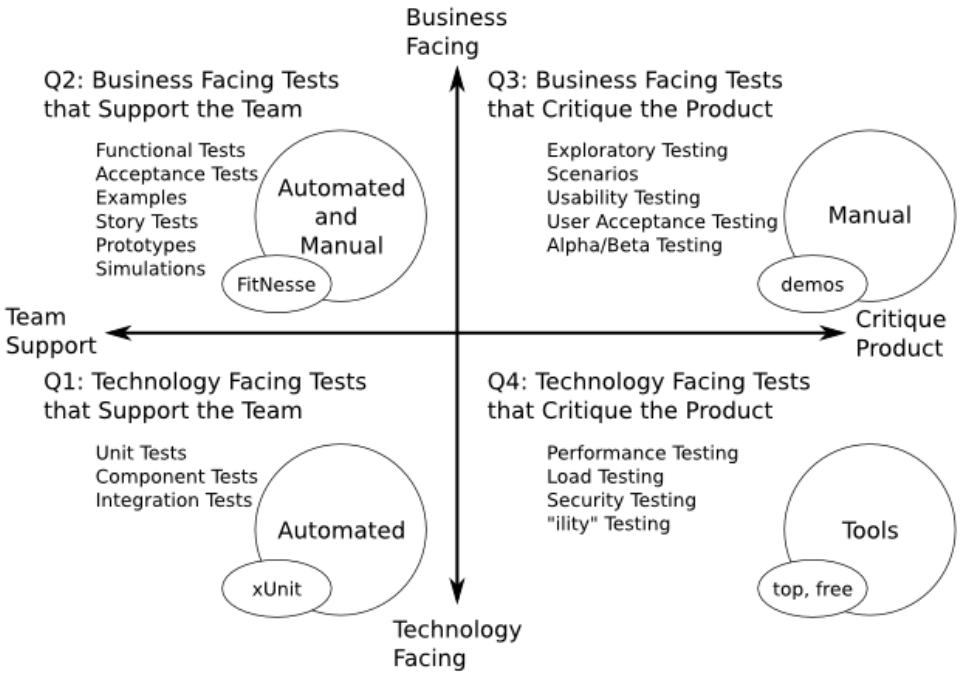
Testa Alla Övergångar



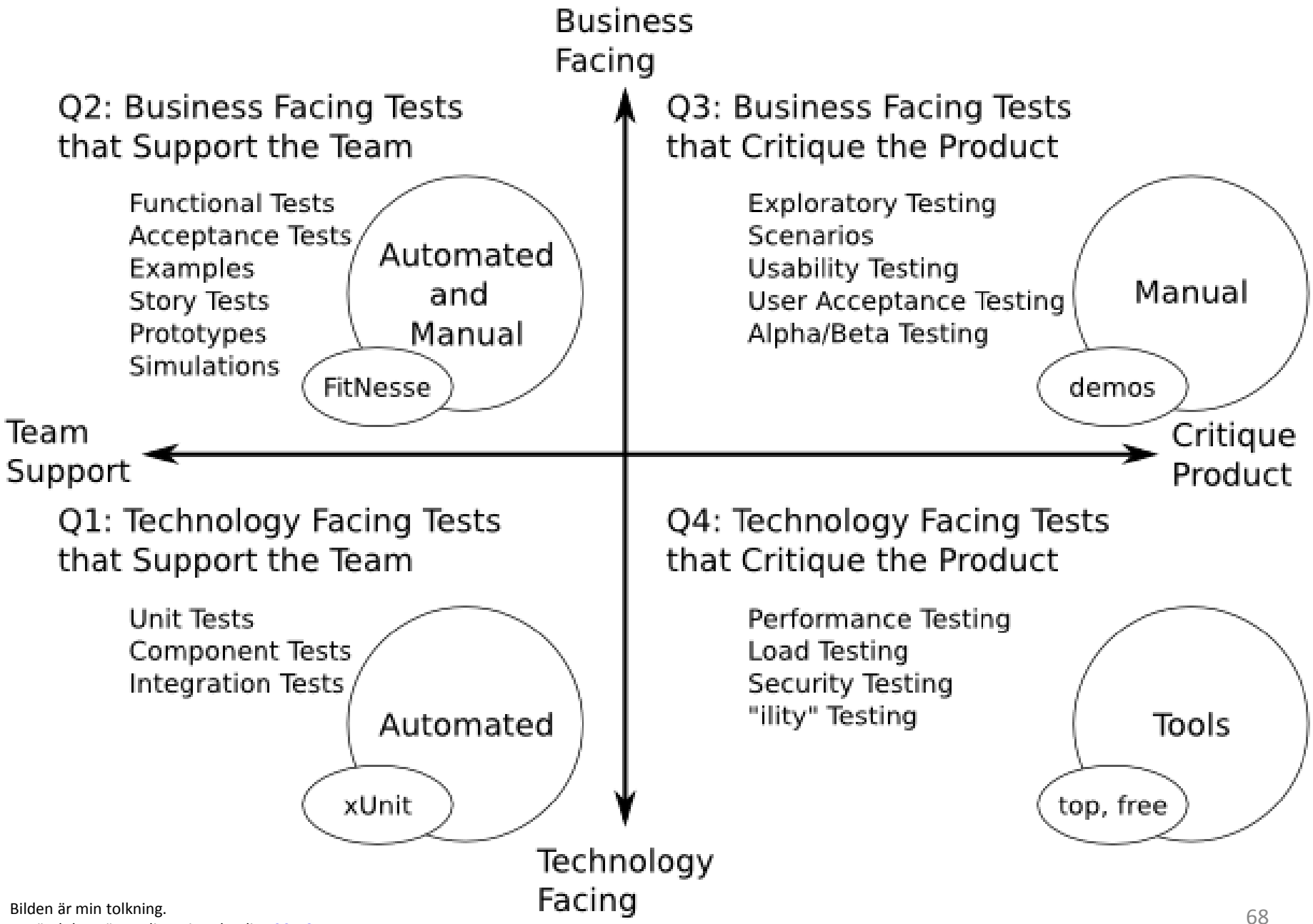
Sammanfattning Övningsuppgift

Agil och Riskbaserad Testning

Agila Testkvadranter



- Testkvadranterna är ett verktyg.
- Kolla att man i varje sprint har tester av olika typer och inte bara till exempel xUnit.
- Idén är från 2003, men populariserades i *Agile Testing* av Crispin & Gregory, 2009 (se sidan *Resurser*)



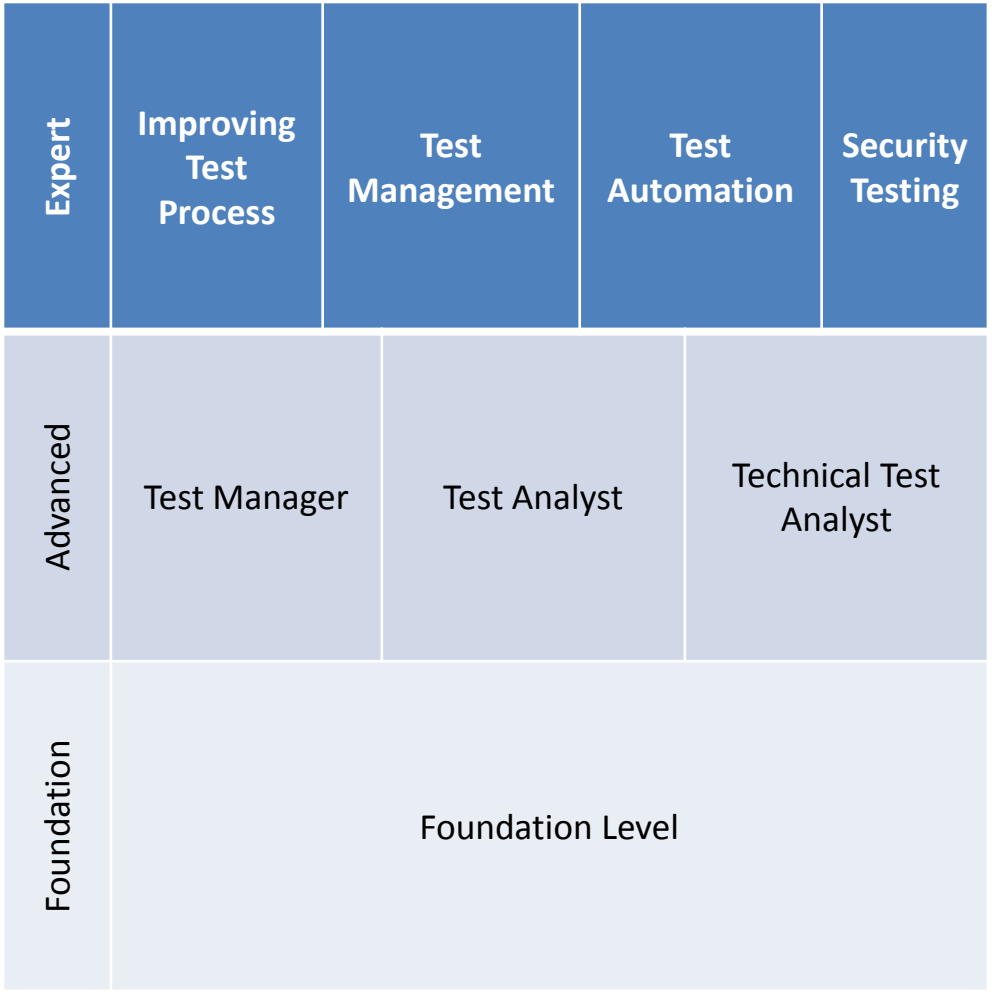
Riskbaserad Testning

(exempel på teststrategi)

- Testa allt går inte
- Måste välja **rätt** tester
- Rätt är här att minska risk (gärna under hela livscykeln)
- Sträva efter att
  - Hitta viktigaste defekterna
  - Samla information och värdera om risker
- Identifiera negativa händelser
- Kvantifiera effekt
- Kvantifiera sannolikhet
- Riskvärde = effekt * sannolikhet
- Angrip värsta riskerna först
- När tid och pengar tar slut är den minsta risken kvar

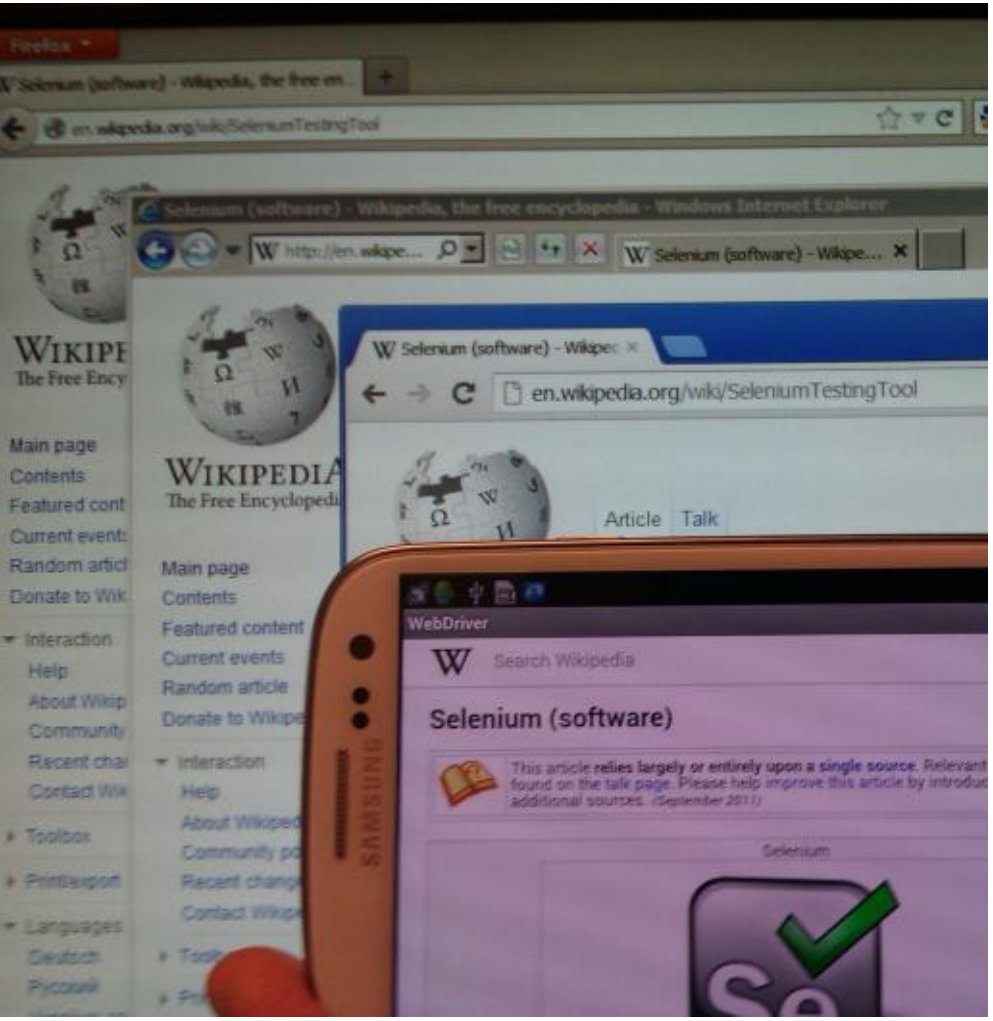
Avslutningsvis

ISTQBs Certifieringar



- Finns kursplan hos till exempel SSTB.
- Foundation ger bra bredd
- Jag lärde mig
  - Vokabulär (Vad heter use case på svenska?)
  - Mer om Granskning
  - Mer om Statisk Analys
  - Mer om Standarder
  - Mer om dokumentation och TPS-reports

Vad har vi inte pratat om?



- Testledning och Verktyg
  - Får man lite om man ska certifiera sig
  - (vi nämnde xUnit, Visual Studio och Selenium men det finns mycket mer)
- Usability testing
  - Och lasttester, säkerhetstester, etc
- Testautomatisering
- Testa mobil webb och applikationer
- Kompetensgrupp i Test 2012
- Kompetensgrupp i Krav och Test 2013

Jag testar webb i Firefox, Internet Explorer och Chrome på Windows samt i Androids webbläsare med verktyget Selenium och lite skript i Python.
http://www.pererikstrandberg.se/blog/index.cgi?page=SeleniumPlusAndroid

Resurser

Rear Admiral Grace Hopper (1906 – 1992) was an American computer scientist and United States Navy officer. She developed the first compiler for a computer programming language. She is credited with popularizing the term "debugging" for fixing computer glitches (motivated by an actual moth removed from the computer). The U.S. Navy destroyer USS Hopper is named for her.

--Wikipedia

- Material från SSTB
  - Swedish Software Testing Board
  - Har till exempel kursmaterial, ordlistor och översättning av termer
  - http://www.sstb.se/se/dokument-6109081
- Rex Black
  - Guru (ibland lite pompös men man vänjer sig) som skrivit många böcker och har en hel rad med webinars om till exempel ”Agile Testing Challenges” och slides
  - http://www.rbcs-us.com/software-testing-resources/library
  - http://www.youtube.com/user/RBCSINC
- Wikipedia:
  - Artikel om Software testing
  - Portal om Software Testing
  - Bok om Software_testing
- Världens första wiki:
  - http://c2.com/cgi/wiki
- Böcker jag kan rekommendera:
  - Software Testing Foundations, Rockynook, Andreas Spillner, Tilo Linz & Hans Schaefer, ISBN 9781933952789
  - Agile Testing , Addison-Wesley, Lisa Crispin & Janet Gregory, ISBN: 9780321534460.
  - Advanced Software Testing, Rockynook Computing, Rex Black, ISBN: 9781933952369
  - Economics of Software Quality, Addison Wesley Educational, Capers jones & Olivier Bonsignour, ISBN: 9780132582209

Sammanfattning

U.S. Navy destroyer USS Hopper

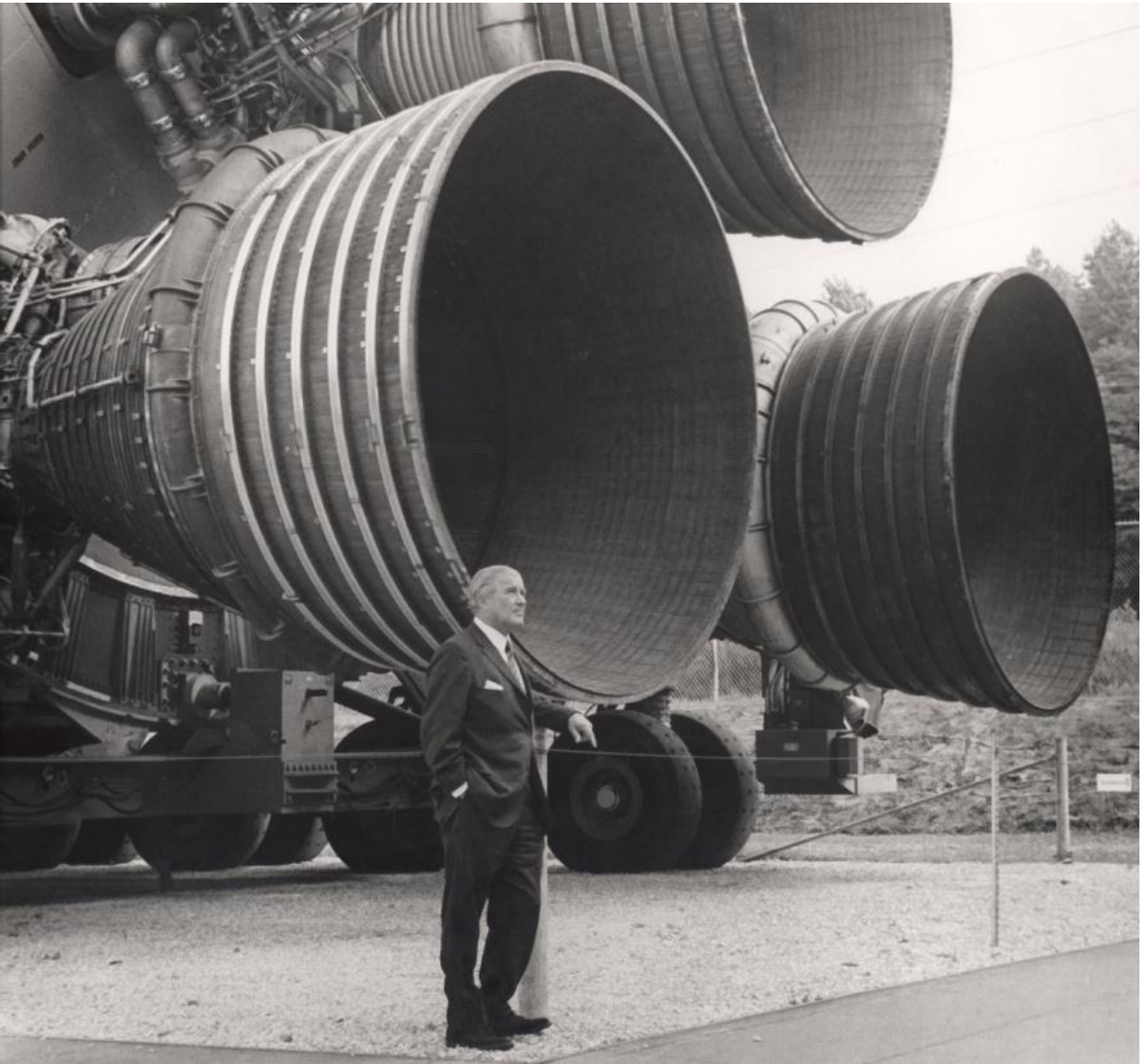
- Vad är test?
- Historisk Översikt
- Motivering & Kostnad
- Olika indelningar
- Komponenttester
- Statisk Testning
- Dynamisk Testning

Repetition

Repetition

- När kom första test-teamet?
- Vad är validering?
- Hur länge ska man testa?
- Hur är indelningen i de agila testkvadranterna?
- Vilka fyra testnivåer brukar man ha?

En annan V-modell



Wernher Von Braun

Byggde en annan V-modell (V-2 raketer – inte den här bilden) som bland annat "bombade" Paris.

Rekryterades till USA (Operation Paperclip).

På bilden syns von Braun med F-1 motorerna till Saturn V.

"To date, the Saturn V is the only launch vehicle to transport human beings beyond low Earth orbit. A total of 24 astronauts were launched to the Moon, three of them more than once, in the four years spanning December 1968 through December 1972." --Wikipedia

Referenser

- Interweb
  - SSTB: Swedish Software Testing Board http://www.sstb.se/
  - ISTQB: International Software Testing Qualifications Board http://istqb.org/
  - Wikipedia om Software Testing http://en.wikipedia.org/wiki/Software_testing
  - Boston Technology: http://www.boston-technology.com/blog/?p=1107
  - Testing References (Joris Meerts, Cap Gemini Nederländerna) http://www.testingreferences.com/testinghistory.php
  - CNN Money om buggar http://money.cnn.com/2012/08/09/technology/knight-expensive-computer-bug/index.html
  - Source Ninja om kostnad för buggar I Open Source projekt http://blog.sourceninja.com/wp-content/uploads/2011/10/sourcecodebug-sourceninja-101411-1.jpg
  - The Inquisitr och Forbes om Apples kartor: http://www.inquisitr.com/349629/apple-maps-fiasco-cost-company-30-billion-puts-ceo-tim-cooks-job-in-jeopardy/ http://www.guardian.co.uk/technology/blog/2012/oct/01/apple-30-billion-maps-mistake
  - RBCS Free Tool for Calculating Software Testing ROI http://www.rbcs-us.com/blog/2010/09/09/free-tool-for-calculating-software-testing-roi/
  - Min hemsida:
    - http://www.pererikstrandberg.se/blog/index.cgi?page=AgilaTestKvadranter
    - http://www.pererikstrandberg.se/blog/index.cgi?page=TestingInVisualStudioPart1
- Se även böcker och sidor listade på sidan Resurser

Bildkällor

- State Library and Archives of Florida
  http://www.flickr.com/photos/floridamemory/sets/

- Copenhagen Suborbitals (Danska Rymdprojektet)
  http://www.copenhagensuborbitals.com/
  https://picasaweb.google.com/114657365514543526210

- Wikipedia och Wikimedia Commons
  http://commons.wikimedia.org/wiki/Main_Page
  http://en.wikipedia.org/wiki/Grace_Hopper