# Kvalitetssäkring i ett Scrumteam

Richard Kronfält, 29 september 2011

# Handuppräckning

> Hur många arbetar idag som ”Testare”?

> Hur många arbetar idag som ”Programmerare”?

> Hur många arbetar idag med projektledning eller team-ledning?

> Hur många arbetar, eller har arbetat, i ett Scrum (eller annat agilt) team?

Sådant som stärker kvaliteten.
”Testning” är bara *en* del av det.

Egentligen 1-4 team

# Kvalitetssäkring i ett Scrumteam

Praktiskt, processmässigt,
samarbetsmässigt, attityder, strategiskt, osv

Grupp människor som (sam)arbetar mot ett/flera gemensamt mål, som försöker följa de ”regler” som Scrum föreskriver

> Att med hjälp av Scrum gå från ”inget” till ”något”

> Mina personliga betraktelser, mina åsikter, mitt perspektiv

# Vem är Richard?

- Mjukvaruprojektledning sedan 2000, mestadels ”traditionella” projekt, off-shore/near-shore
- Tek. Mag. Programvaruteknik Karlskrona-Ronneby
- Agile, Lean och Scrum sedan 2007
- Introducerade Scrum @ PowerChallenge & Managerzone, 2007 - 2009
- Introducerade Scrum @ Axis, 2009 - pågående
- Linjechef sedan början av 2009 – **inte testingenjör**
- CSM, CSPO and CSP
- Scrumblogg (www.scrumftw.com), Scrum Practitioners South, m.m.
- richard.kronfalt@gmail.com / http://linkedin.com/in/kronfalt

## Traditionell ansats

Faser

Mjukvaruutveckling
& kvalitetssäkring

?

...Eh??

Iterativ &
Inkrementell

**Oordnad ansats**

**Agil ansats**

# Den traditionella

> ”Vattenfall”

> Indelad i tydliga faser

> Kraven låses tidigt

> Testspec som baseras på låsta krav

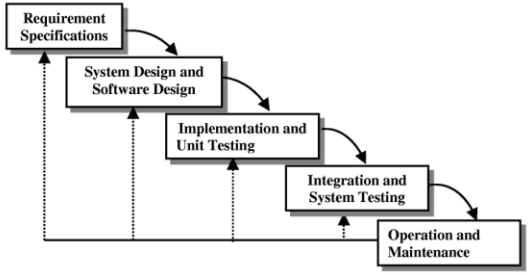
> Testfall förbereds under Implementationsfasen

> Överlämning till Testarna efter Implementationsfasen

10: > Testningen genomförs systematiskt

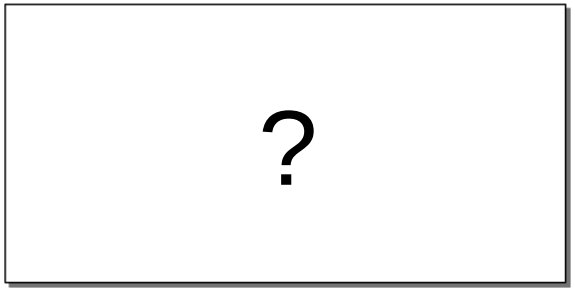
> Buggar noteras, spåras och de viktigaste åtgärdas

> Goto 10 tills redo för release – och/eller tiden är slut



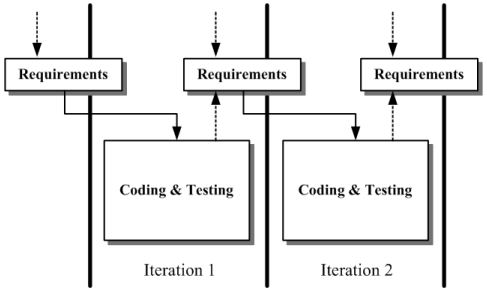
# Den oordnade

- ”High Chaparral”
- Dysfunktionell, kaotisk
- Ingen systematik
- Teststrategi saknas
- Utvecklarna testar (i bästa fall) sina egna saker
- Ingen/liten spårning av vad som testats
- Ingen/liten spårning av buggar
- Release gör man när projektet är maximalt försenat (”Vi *KAN* inte vänta längre..!”)
- Framgång beror på slumpen och/eller individuella hjältedåd



# Den agila

- Fokus på kunden, och på att addera maximalt värde
- Fokus på samarbete, i gruppen och med kunden
- Iterativ & Inkrementell
- Ständig ”releasebarhet”; gör saker *färdiga*
- Korta iterationer (1-4 veckor)
- Nytt produkt-/funktionalitetsinkrement efter varje iteration
- Tvärfunktionella team
- Samarbete och självorganisation
- Testning integrerad del i programmering



# Bakgrund

> Axis
  - Nätverksvideokameror, global marknadsledare
  - Totalt ~1100 anställda, 3 miljarder omsättning 2010
  - Huvudkontor och hela R&D i Lund

> Mitt team
  - Mestadels webbteknik (PHP), en platform för hosting av tusentals nätverkskameror
  - Vi är ansvariga för underhåll och vidareutveckling
  - Proof-of-concept, första installation hos kund cirka 2004
  - 2-3 personer under 2004, 8 personer slutet av 2008, 20+ personer slutet av 2011
  - Intern beställare (produktchef)
  - Relativt ”isolerad” utveckling (få beroenden utanför teamet)

# Utgångspunkten (2008)

- Relativt oorganiserat, entreprenörsandra, proof-of-concept, genvägar, otydlig styrning
- 8 engagerade, högmotiverade självgående PHP- & C-programmerare, inga testare
- Hög frihetsgrad och hög ”innovationskänsla”
- Liten grad av samarbete; alla hade sin ”agenda”/favoritområde
- Direktkontakt med kunder
- Ingen strukturerad testning
  - Och absolut ingen testautomatisering
- Inget processförbättringsarbete
- Åldrande kodbas, ständigt ökande teknisk skuld
- Varken krav- eller test specifikationer
- Arbete utanför R&Ds policies och processer (inga processer eller policies)

# Önskelistan 2008/2009

> Bibehållen eller ökad **effektivitet**

> Bibehållen kreativitet/**engagemang**, känsla av **frihet** & **inflytande** hos ingenjörerna

> Möjlighet för en ”**beställare**” att ansvara för produktens riktning/innehåll

> Pålitligt/**upprepbart leveransresultat**

> Möjlighet till att hålla **överblick**

> Sluta öka den **tekniska skulden**

> Delvis **refaktorisering** av arkitektur/kodbas

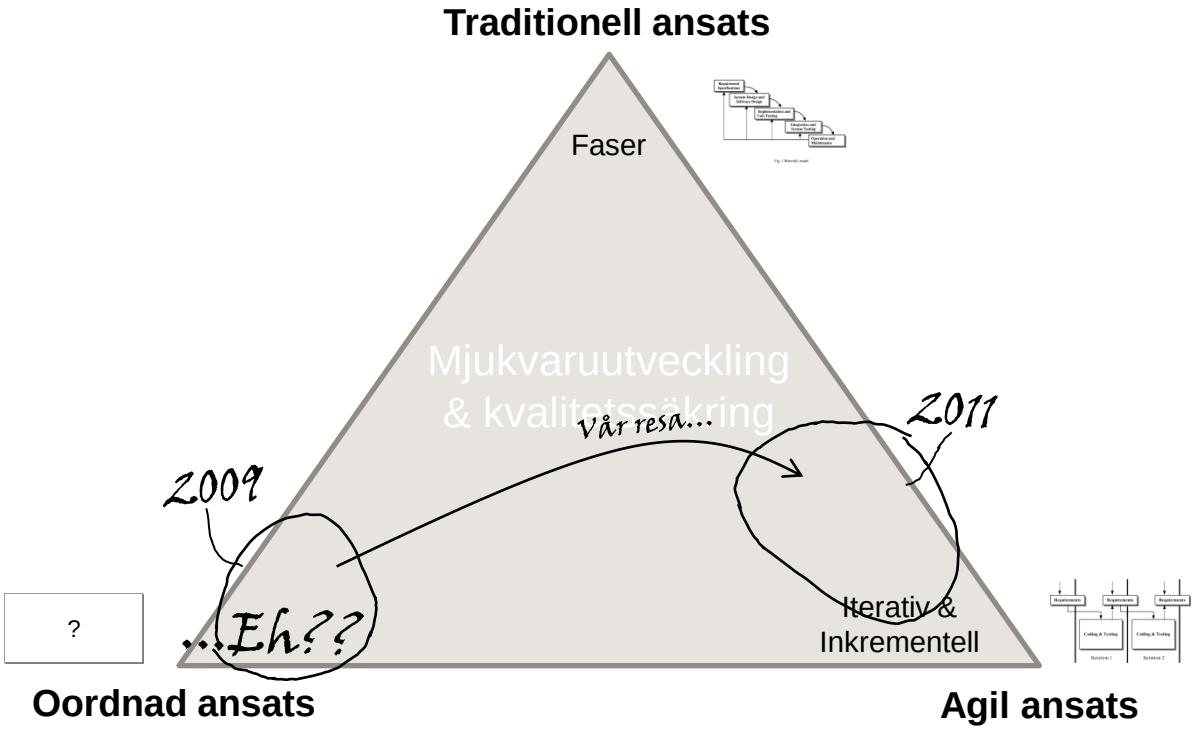
> Börja med **systematisk testning** av något slag

> Dra nytta av de **QA-resurser** som finns i företaget

> **Kundsupporten** får ta över supportfallen så programmerare får fokusera

> Följ R&Ds övergripande **projektmodell**

# Vår förändring



# Så här arbetar vi idag (1/2)

> 1...2...4 tvärfunktionella Scrumteam
> Alla sitter tillsammans
> 5-7 personer/team
> Självgående team, hög grad av samarbete
> 1 Scrum Master
> 1 Scrum Product Owner, ”proxy” för Produktchef
> 9 dagar/sprint
> Estimering i Storypoints → Velocity → Releaseplan

# Så här arbetar vi idag (2/2)

- Hög grad av testautomatisering
  - Ca 1350 automatiska funktionstester
  - Ca 100 automatiska ”GUI-tester”
  - Ca 2534 automatiska unittester
  - Ca 110 manuella testfall
- Små fungerande produktinkrement varje sprint
- Mål vara ständigt releasebar men ej där ännu
- Naturligt & ständigt förbättringsarbete
- Också fokus på att
  - rätta gamla buggar,
  - ersätta manuella tester med automatiska
  - skriva unittester för gamla (legacy) units

# Reflektionsdags…

> Vilka saker har varit avgörande för våra möjligheter att arbeta agilt?

# Det behövs en uttalad & tydlig policy om testning

> All in i vårt fall. Tydligt att verkligen satsa på den agila modellen

> *All* ny funktionalitet ska testas *inom* iterationen

> *Alla* nya buggar ska fixas *inom* iterationen

> ”Gamla” buggar som hittades lade vi på hög så länge; fokus på att lära oss hantera nya

  – Har på senare tid börjat hantera dessa ”parallellt” med sprintarna (maintenance team)
  – Kanske hade kunnat lyfta in dem som storypointestimerade items i sprinten istället?

# Det behövs en strategi för hur hantera ”teknisk skuld”

> Uttalat mål att sluta öka vår *tekniska skuld**

> Sluta ta genvägar för att spara tid

> Det måste vara okej att det tar längre tid att göra rätt

> Innebär inte nödvändigtvis ett fokus på att *minska* den tekniska skulden (refaktorisering)

* *The dirty that remains long after the quick has been forgotten.*

# Testning måste bli viktigt och naturligt

> Upp med det på dagordningen!

> Vi hade *ingen* vana av att ”testa”

> Vi hade möjlighet att bemanna teamet med Testingenjörer

> Vad är Testingenjörens ansvar jmf med Programmerarens?

- Osäkerheten uppstår då rollerna betraktas ur ett traditionellt perspektiv
- En ”agil teammedlem” inte samma som traditionell ”testare” eller ”programmerare”

# Testarna måste sitta ihop med Programmerarna

> Vi ”fryser” (skriver i sten) aldrig detaljkrav innan implementation

> Därmed finns ingen stabil ”kravspecifikation”

> Därmed finns ingen ”testspecifikation”

> …Läskigt?

> Förståelse för vad och hur testa uppstår *samtidigt* som analys, design och implementation

> Mycket hög grad av samarbete i teamet = kräver att samtliga sitter tillsammans rent fysiskt

> En organisatorisk utmaning? I vårt fall buy-in från R&D-ledning och QA-chef

# Våga investera i automatiserad testning

> Automatisera så mycket av testningen som möjligt

> Även om initialt dyrare än manuell

> ”Alla” nya units (funktioner i koden) ska åtföljas av unittestfall

  – Individuella kodfunktioner/metoder testas

> ”All” ny funktionalitet ska åtföljas av funktionstester, *helst* automatiserade

  – Hela funktioner (features) testas, som blackbox

  – I vårt fall har vi ett API som man kan göra ”det mesta” genom, som vi automattestar

> Uthållighet! Var beredd på att det tar tid innan en automatiska testsvit är ”tillräckligt” heltäckande, dvs innan de gör skillnad

# Uppfinn inte hjulet för att automatisera testning

- Automatisera funktionstester, unittester och i viss mån även ”GUI-tester” (automatiserade ”klicktester”)
- Undersök existerande ramverk för testautomatisering
- Ruby blev grunden för våra automatiserade funktionstester och ”GUI-tester”
- PHPUnit blev basen för våra automatiserade unittester
  - JUnit, CUnit, CPPUnit, NUnit, osv

# Satsa på datadriven testning från början

> Datadrivna tester* förenklar testfallskodbasen

> Återanvändning av kod; avsevärt mindre duplicering

> Tidsvinst vid tillägg av nya testfall och vid underhåll av gamla

> Vi fick gå tillbaks och refaktorisera testkod när vi upptäckte DDT

** Likartade testfall har gemensam kod för setup, exekvering, resultatverifiering och teardown, med en gemensam datafil som definierar de olika specifika testfallen.*

# Skapa en strategi för hur testa legacykod

- Legacy = gammal kodbas med gamla ”genvägar”
- Poängen med automatiserad testning är att den ska vara så heltäckande som möjligt
- Man ska vara trygg i att automattesterna fångar regression i systemet
- = Det räcker alltså ej att införa testautomatisering endast på *NY* kod/funktioner
  - Men det är en bra början!

# Lägg tid på att faktiskt hantera legacy-units

> Analysera kodbasen och identifiera existerande units (funktioner)

> Rangordna dem utifrån hur ofta de exekveras och efter hur komplexa de är

> Investera tid i att skapa unittester för gammal kod utifrån ovan ordnade lista

> Kanonbra arbete för sommarjobbare! :-)

> I vårt fall började det med ett exjobb; ”Hur införa unittestning i ett legacy PHP-system”

# Arbeta aktivt med konvertera testfall till automatiska

> Att ha manuella testfall är kostsamt

> Ha strategi för att ersätta manuella testfall med automatiska

> Utgå ifrån att du måste ha manuella testfall, men *sikta gärna på att inte ha det*

> Mål att ha en *verkligt* releasebar produkt efter varje sprint

> Idag har vi 100+ manuella testfall, men arbetar aktivt med att skriva automatiska testfall som ersätter manuella

# Vinsterna av & riskerna med automatisk testning?

> Sparar tiden det tar att exekvera ett manuellt testfall igen och igen
> Det är en baggis att testa av en förändring och fånga regression
> Främjar god (testbar) design
> Steg mot ”ständig releasebarhet”

> Övertro: testerna kan också innehålla fel → falska positiva & falska negativa utfall
> Underhållbarheten påverkas; ändringar i systemet kräver ändringar i testkoden
> Balans mellan effort på nya features och effort på testkod

# Rollerna förändras i ett tvärfunktionellt team

- Stark fokus på automatisering förändrar vad rollen ”Testare” innebär
- Ett automatiskt test blir aldrig bättre än ingenjören som utformar det
- ”Testare” är inte längre en person exekverar manuella testfall
  - Någon som behärskar konsten att identifiera och utforma automatiska testfall där manuella tester tidigare vore det naturliga valet
  - Vem som helst i ett Scrumteam
- Även den traditionella rollen ”Programmerare” förändras; gränserna suddas ut
- En agil teammedlem är både ”Testare” och ”Programmerare”

# Svårt att förändra traditionell roll

- Utmaning att få ”Programmerare” att acceptera att dom också kan vara ”Testare”
- Återkommande diskussioner kring ”vi-och-dom”
- Noga att alltid bemöta sådana diskussioner omedelbart och konsekvent
  - Programmerare KAN och MÅSTE skriva automatiska funktionstestfall
- Det tar tid att vänja
- Typiskt motargument: ”Men testarna kan ju inte hjälpa programmerarna att koda funktionerna, varför ska vi då hjälpa dom att skriva testfallen?”
  - Men alla har samma mål och alla måste bidra med allt dom kan
  - En agil teammedlem måste vara flexibel och våga gå utanför sitt traditionella område

# Ta in ”Testare” med programmerarprofil

- An ”agil testare” i ett utvecklingsteam måste ha stor förståelse för programmering, och absolut för testautomatisering
- Vi har (hittills) haft förmånen att nyrekrytera in alla våra testingenjörer och har därmed kunnat välja personer med programmerarprofil
  - Slipper hantera gamla ”vanor”?

# Behovet av utbildning & förändringsvilja

> Viktigt att alla förstår: Vad är Scrum & Agile? Vilka problem löser det? Och hur?

> Gemensam förståelse förutsättning för att alla ska dra åt samma håll

> För tung börda för en ensam champion

> Alla måste inte vara övertygade, men alla måste *vara öppna* för förändring

# Vikten av att ha Scrum-champion(s)

> Förändring kräver passion och övertygelse!

> Under press är det lätt att falla tillbaks till gamla rutiner (i vårt fall; High Chaparral)

> Kunskap och erfarenhet krävs för att utbilda och för att ”hålla riktningen”

> Hitta Scrum-champion(s) som kan leda förändringsarbetet

# Avgörande med stöd från omgivningen

> Förutom pengar/resurser krävs flexibilitet, experimentvilja, förståelse, tålamod, osv

> Produktledning; vårt arbetssätt förändrar beställarens roll jmf övriga beställare i org.

> QA-chef(er); vi har förändrat både teststrategin och testarens roll i ett utvecklingsteam

> R&D-chef & ”Projektkontor”; vi har tillåtits avvika ifrån etablerade processer

> Avdelningschef; vårt arbetssätt skiljer sig från övriga delar av avdelningen och programmerarens roll är förändrad

# En strävan efter ständig releasebarhet

> Beslutet om release flyttas från *utvecklingsteamet* till *produktägaren*

> Man exponerar ständigt bristerna i sin utvecklingsprocess → lockar till förbättringsarbete

> Minskad risk → hela utvecklingsapparaten prövas ständigt

> Minskad potentiell waste → Färdigställer (verkligen)

> En release är ingen big deal

> Högre kvalitet (färre buggar)?

# Tydlighet i Definition of Done

> Specifiera i din Definition of Done vad som ingår i ”Releasebar”
  - Färdigkodad?
  - Dokumenterad? Vad för dokumentation?
  - Unittestad?
  - Funktionstestad? Automatisk?
  - Granskad?

# Versionshantering

- Att arbeta agilt ställer krav på versionshantering
- Ständig releasebarhet kräver att man kan skilja på kod som är ”Done” och kod som är ”In Progress”
- Resulterar ofta i många brancher
  - välj ett verktyg som är ”bra” på brancher och på merge
- Flera team på samma kodbas ställer ännu fler krav på versionshantering
- Vi använder Git (infördes för ca ett halvår sedan, tidigare CVS)
- Vi lär oss fortfarande, och förändrar vår policy

# Vad gör en Testare i början av en iteration?

- Samma som övriga teammedlemmar, dvs
  - Vara delaktig i nedbrytning och estimering
  - Var delaktig i analys och design
  - Identifiera tester som behövs för de påbörjade och kommande user stories i iterationen

- Explorativ testning
- Konvertera befintliga manuella testfall till automatiska
- Rätta buggar i automattestramverket
- Förbättra automattestramverket

# Tappar man oberoendet vid integration av utv/test?

> Ja, oberoende kan man inte vara om man ingår i teamet som står för leveransen

> Men… Det kanske kan vara värt det eftersom:
  - Med en agil modell fokuserar hela teamet och hela processen på kvalitet
  - Ständigt förbättringsarbete där alla är inblandade
  - Regelbundet kvitto på kvalitetsnivån (varje iteration)
  - Bygger på förtroende för att varje teammedlem kan och vill värna om kvalitet

> ”Kvalitetspolis” känns som ett föråldrat begrepp

# Det måste vara enkelt att köra automattester

- Automattester blir mycket centrala i agil utveckling
- Därför viktigt att det är enkelt att köra dem
  - Varje teammedlem enkelt kunna initiera automatiska tester på ”sin” kod
- Viktigt att testresultaten är deterministiska
  - Så de alltid ger samma resultat
  - Work in progress här för oss…

# Testning är nu en självklarhet

> Alla förväntar sig att vi arbetar med ”testning”

> Alla teammedlemmar är delaktiga och drivande; både ”testare” och ”programmerare”

> Naturlig fokus på att vidareutveckla vår automatiska testning

> Gemensamt mål att nå fullständig releasebarhet; alla vill dit

# Frågor?

# Kontakta mig gärna

> Kontaktuppgifter:
  - richard.kronfalt@gmail.com
  - http://www.linkedin.com/in/kronfalt
  - http://www.scrumftw.com

> Kontakta mig gärna om Du har frågor om eller synpunkter på innehållet i denna presentation, och/eller vill veta mer om Scrum & Agile.


Axis
competence
worldwide
leader
HDTV
network video
convergence
intelligent
open
camera
megapixel
easy installation
partner network
Thank you!
safe
innovation
outdoor
environment
leader
thermal
global
ease of use
H.264
protect
integration
image usability
focus
video encoder